摘要

大学生机器人大赛是一个集科技创新与竞技比赛于一身的对抗性项目，机器人系统开发过程主要涉及到智能控制理论、嵌入式系统、计算机算法、人工智能等多个学科，参加该项赛事是一个培养自动化、信息科技人才，展示高科技成果，推进产业化与实用化的新途径，对于机器人科学与技术的普及，机电技术与智能控制的研究与应用有着极大的促进作用。

本课题是以第十四届亚太大学生机器人大赛为背景，其目的是帮助场上的机器人判断羽毛球的落点，使得机器人能够对飞来的羽毛球进行准确击打，击败对面的机器人从而赢得比赛，最终获得亚太冠军。

本课题利用数字图像处理技术，在摄像头读取到的图像中，利用帧差法，选取出感兴趣区域，然后进一步用特征提取的方法，从感兴趣区域中找到羽毛球，得出羽毛球的二维坐标，接下来，用双目三维重建技术，将两个摄像头分别得到的二维坐标通过矩阵运算转化为三维坐标，接着利用已知的羽毛球飞行的轨迹坐标，利用空气动力学方程，预测羽毛球的接下来的飞行轨迹，并通过基于已知位置的初速度矫正方法，对开始预测的轨迹坐标的初始状态进行校正，从而得到较准确的飞行轨迹。通过蓝牙通信系统，将得到的目标高度的 xy 坐标以及飞行的时间发给机器人，使得机器人能跑到指定地点进行击打。

关键字：双目视觉；图像处理；特征提取；通信；轨迹预测

The Research and Design of The system about tracking badminton based on double camera

Abstract

Robotics Competition is a kind of confrontation project containing high-tech innovation and athletics. This project involves variety of area including mechanical design and manufacture, theory of intelligent controlling, singleship, computer algorithm, artificial intelligence and is a new way of training elite in the domain of Automation and Information Technology, showcasing high-tech product promoting industrialization and functionization. Holding Robotics Competition impulse immensely popularization of robotic science of technology and promotion of research about electromechanical and intelligence controlling.

This topic is based on the 12th Asia-Pacific Robot Contest of College Students, which help the robots on the match to judge the track of badminton so that the robots can bit it back accurately, defeat opponent and finally win the Asia-Pacific champion.

Using image processing, this topic dispose the image from the camera based on the method of frame difference and extract the ROI. And then, we can get the badminton from the ROI using feature extraction so that we can calculate the two-dimension position of badminton and translate the posiont into three-dimension position using matrix operating. The next step, we can predict the track of badminton using kinetic equation based on the three-dimension position we have gotten from the double camera. In order to improve the precision, this system adopt a method which corrects the initial speed of the badminton based on the current position we have gotten. When the computer have gotten the predicted track, the system can send the destination of xy and the bit time to the robot through the system of Bluetooth so that the robot can move the destination and bit back the badminton.

Key words: double camera; image processing; feature extracting; communication; tracking predicting

目录

引言

本次竞赛解决的主要问题就是研究、设计并调试出一个能针对场上环境做出稳定快速动作的机器人系统。针对这一问题，所做的工作涉及到自动控制原理、图像处理技术、无线通信技术、人工智能控制、液气压传动理论以及硬件电路基础等知识与技能。参加比赛的队员需要分工合作，对这些领域的知识进行深刻的理解，并综合考虑多方面的因素，共同完成一个融合多领域知识的机器人操作系统。

在对机器人的研究、设计过程中，充分利用不同软件的功能来解决遇到的难点与问题也是一个非常重要的途径。用 OpenCV 来实现系统的运行，用 Excel 来记录并定量求解落点误差，用 MATLAB 来标定摄像机和对数据的仿真、模型的建立。将这些软件恰到好处运用，有助于对机器人从理论研究到实践的实现。

本文的第一章主要阐述了参加机器人大赛的意义以及概述了对羽毛球进行跟踪判断涉及的主要技术。第二章介绍了针对这次比赛所需要的硬件以及将二维坐标转换为三维坐标的重建技术。第三章讨论了羽毛球追踪系统之间的通信技术，包括计算机与计算机之前的局域网技术以及计算机与机器人之间的蓝牙技术。第四章详细地介绍了有关羽毛球识别的相关技术，包括图像预处理、运动目标检测、特征提取与特征判断。第五章构建了羽毛球的运动方程，计算羽毛球的运动轨迹，并用基于位置的初速度调整算法以及线性拟合去除噪点的方法使得结果更加准确。

1 绪论

1.1 课题背景及研究的意义

1.1.1 国内与国际比赛背景分析

“亚太大学生机器人大赛国内选拔比赛”是中国中央电视台组织的比赛，其目的是要在全国高校之间选拔出代表队，代表中国队参加“亚广联亚太地区机器人大赛”，这个比赛可以成为亚广联的预算赛。

而亚太大学生竞技机器人大赛早在 2002 年，由亚广联（ABU）倡导并开始举办，通过各个国家自主选拔出冠军队之间的争夺，从而选出亚太地区的机器人大赛的冠军队。

1.1.2 本届规则背景与分析

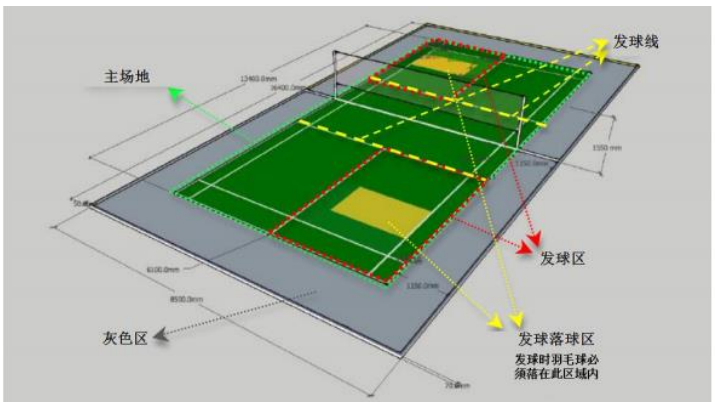


图 1.1 比赛场地的透视图

今年的比赛题目是“羽球双雄”，既是两个机器人为一组，进行羽毛球双打比赛。

规则：本届比赛关于羽毛球对打的规则较为简单，首先由一方机器人发球，将球发到指定区域，然后对方机器人要将球击打回来，双方进行击打直到有一方机器人不能将球成功击打回来，则对方机器人赢得一分。率先获得 5 分而对方不足 4 分或者率先获得 7 分的一方获得比赛胜利。

那么如何实现自动击打羽毛球的任务呢。这就需要计算机视觉的辅助作用，如果要成功回击羽毛球，必须在羽毛球飞行的时候，就要对其的下落到目标高度的羽毛球 xy

的位置做出预判，因此，我们选用摄像头来对飞行的羽毛球进行检测、跟踪，像人类的眼睛一样，让机器人对飞来的羽毛球能够进行提前的位置预判，为准确击打羽毛球做出保证。

羽毛球的飞行时间总体很短，所以，在设计系统的时候，实时性是首要因素，然后对羽毛球的检测、飞行轨迹的确定也是不容忽视的关键因素，这就需要通过大量的实验来调整系统的系数，从而保证轨迹预测的准确性。

系统设计的第一个难点是如何保证检测羽毛球的准确性。因为比赛背景是动态的，在复杂背景下，如何能够有效的滤除掉噪音，准确的提取出羽毛球，是关键的一步。如果不能有效的检测出羽毛球，那么在后期的构建的轨迹模型中，必然会造成极大地偏差，最坏的情况，可能直接导致机器人冲出场地导致损坏。

系统设计的第二个难点是如何调整参数来保证轨迹模型的准确性。羽毛球飞行的过程中，会受到重力、空气阻力、空气粘滞力、马格努斯力等众多的力，如何在在这些力的影响下，找到一条准确的飞行轨迹。就算羽毛球检测的再准，如果轨迹不够准确的话，那么检测出的羽毛球就如同一个个孤立的点，对机器人的预测毫无帮助。本论文主要围绕这两个关键问题来设计系统。

1.2 参加机器人大赛的意义

对机器人的研究，从很早就已经开始了，而且从来就没有间断过，机器人技术的发展能够促进工农业的发展[1]。现如今，机器人在我们生活中所占的比例越来越大，家政机器人、洗车机器人等各种各样的机器人不断涌现到人们的生活中[2]。亚太地区大学生机器人大赛，其目的是为了促进大学生开发并利用机器人。通过对机器人的不断开发与应用，促进学生的动手能力，编程能力以及思维逻辑能力，并且不断促进机器人技术的进步与发展，使得机器人能够完成越来越多的复杂任务。机器人的研究与开发使得机器人越来越有能力完成人类下达的复杂指令，并且能更好的完成一些危机人身安全的工作，同时也能够代替人类完成一些日常繁琐的工作，减轻人类的劳动，便于人类进行更具有创新的活动。所以研究机器人的意义十分重大，通过这次比赛，可以让大学生充分理解机器人是如何运行的。

亚太地区大学生机器人比赛是一个影响范围很广的比赛，它是学校科技活动的重要一部分，在过去的比赛中，北京科技大学都取得了不错的成绩，不光为学校带来了荣誉，也促进了学校科技活动的发展。学校对此项比赛的大力支持，有助于提升大学生的动手、学习能力。

而对于个人而言，参加这个比赛，不仅使自己在图像处理方面学到的理论进行了实践，而且也拓展了自己在这个领域的视野、提高了自己的技能。作为一个本科生，能有这么多的资源让自己学习，对自己能力的提升有着很大的帮助。同时，在这个团队，有很多在各自领域有着深入研究的人，跟他们一起工作，使自己学到了终身受用的知识。

1.3 研究内容、预期目标及研究方法

1.3.1 基于双目视觉的三维重建

对双目摄像头三维重建技术的学习与研究[3]，并利用 MATLAB 标定工具箱完成标定任务。MATLAB 标定工具箱是将张正友标定原理与方法[4]进行函数的封装，降低了标定难度。并在现有的双目摄像头三维重建技术的基础上，进行改进，以便能够快速、准确地进行标定，以适应比赛的要求。设计一个机械结构，为的是能够在移开摄像头后再次找准前一次的标定位置。

1. 3. 2 通信系统设计

竞赛机器人的通信系统主要分为两个模块，即摄像机之间的通信模块、摄像机与机器人之间的通信模块。因为规则原因，摄像机之间的通信用的是局域网技术，而摄像机与机器人之间的通信用的是蓝牙技术。通信模块是系统不可或缺的一部分，只有保证了通信逻辑的正确性，才能使得系统稳定地运行。

1.3.3 基于 OpenCV 的羽毛球的检测与跟踪

在 vs2010 的平台下，用 OpenCV 对图像进行初步处理[5]，利用帧差法[6]，首先从整个图像中寻找可能会出现先运动的羽毛球所在的感兴趣区域，然后，利用图像处理技术，对这些感兴趣区域进行噪声的滤除，然后利用图像匹配技术[7]对羽毛球进行准确的检测、识别。

1.3.4 对羽毛球飞行轨迹建模

利用空气动力学模型[8]，对羽毛球受到的各种力力进行详细分析，然后根据受力分析建立空气动力学方程，计算出羽毛球在 x、y、z 三个方向的加速度，利用迭代方法模拟出羽毛球飞行轨迹，获得羽毛球在击打高度的 x、y 坐标与时间。并用 MATLAB 来画出落点图，进行系统参数调整。

2 三维重建

2.1.硬件的选型

2.1.1 摄像机的选型

关于摄像机的选型有三个参数是比较重要的[9]。

（1）有效像素。这个参数是决定了摄像机拍摄到图像所能达到的最大分辨率。因为需要考虑实时性，大的分辨率虽然细节更加清楚，但是也意味着计算量将增大，因此，摄像机的分辨率不宜太大。

（2）成像帧率。帧率即是单位时间内摄像机能获取到的图像的个数。帧率越高，则在有限的时间内，获得更多的图片，在羽毛球的飞行过程中，才能获得更多的信息。

（3）成像颜色。市面上的摄像机分为彩色和黑白两种，黑白的所占内存少，处理速度快，虽然黑白图像会丢失一些信息，但是，经试验发现，这些信息不会对羽毛球提取构成很大影响。

综上所述，我们选用的迈德卫视 GIGE 工业相机。其个别参数如表 2.1 所示。

表 2.1 摄像头参数

有效像素	640*480（30 万）
像元尺寸	5.6um*5.6um
曝光时间	0.0167ms~30000ms
帧率	140fps（最高）
成像颜色	彩色
支持开发语言	C++、C#、VB、Delphi6、Labview

2.2.2 镜头的选型

镜头的选型主要是有两个参数比较重要[10]：

镜头成像尺寸：镜头的尺寸一定要不能大于摄像机的 CCD 的靶面尺寸。否则，在图像中会出现黑边的效果。

焦距：镜头主要分为两种，一种是可变焦镜头，另一种是定焦镜头。镜头的焦距要跟实际的工作距离和所需视野范围来确定。

选取的 1/4“CCD 靶面尺寸为宽 3.2mm x 高 2.4mm，假设工作距离为 D，在 D 处视场大小要求为宽 H，高 V，则焦距为 f = 3.2 x D/H，或 f = 3.2 xD/V。在此需要注意，f 的计算只能是一个大致的估计。我们在选取镜头的时候，会发现只有一些固定焦距或焦距范围的镜头。若想要定焦镜头，则应当选用与计算值 f 焦距最靠近的镜头，如果选变焦镜头，则使 f 包含在镜头焦距范围内即可。

2.2 标定原理

2.2.1 内外参标定原理

摄像机的内部参数简称为内参[11]，这个参数取决于摄像机的固有的内部结构。其包括了 3 个主要参数，1）像素点的坐标（u_0, v_0），摄像机的尺度变换因子（f_u, f_v），3）透镜的倾斜因子。

摄像机的外部参数简称为外参，这个参数取决于两个摄像机的位置信息。外参由旋转矩阵 R 和平移向量 t 来共同表示，外参包含了 6 个独立的参数，其中，旋转矩阵 R 和平移向量各 3 个。

设该点在三维世界的坐标为$M=[X,Y,Z,1]$，设该点在二维相机的平面坐标为$m=[u,v,1]^T$。那么，对应的关系如式 2.1 所示。

$$sm = A[R,t]M = A[r_1,r_2,t]\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2.1}$$

为了简化运算，设该点在一个与摄像机平面平行的三维平面即令 Z=0 的平面中，那么则有$M=[X,Y,1]^T$，该式表示的是 Z=0 平面的齐次坐标。$m=[u,v,1]^T$则是该平面投影到图像平面的对应的齐次坐标。设相机的内参矩阵如下：

$$A=\begin{bmatrix}\alpha,r,u_0\\0,\beta,v_0\\0,0,1\end{bmatrix}$$

记平移矩阵为R，旋转矩阵为t，缩放因子标量为s。

把式2.1简化一下，如式2.2所示：

$$sm=HM \tag{2.2}$$

其中 $H=A[r_1,r_2,t]$。

$$H=[h_1,h_2,h_3]=\lambda A[r_1,r_2,t] \tag{2.3}$$

其中，λ 是一个缩放的因子变量，$\lambda=\frac{1}{s}$，那么，$r_1=\frac{1}{\lambda}K^{-1}h_1$，$r_2=\frac{1}{\lambda}K^{-1}h_2$。

通过研究旋转矩阵的性质，则有结论 $r_1^Tr_2=0$ 以及 $\|r_1\|=\|r_2\|=1$，那么对每一幅图像俩说，可以得到一个关于此的约束方程：

$$h_1^TK^{-T}K^{-1}h_2=0 \tag{2.4}$$

$$h_1^TK^{-T}K^{-1}h_1=h_2^TK^{-T}K^{-1}h_2 \tag{2.5}$$

我们知道，当待标定的图像数目大于或等于3的时候，就可以产生6个及以上的方程，那么，就可以唯一的求解出含有5个未知变量的内参数矩阵A。

$$B=A^{-T}A^{-1}=\begin{bmatrix}B_{11},B_{12},B_{13}\\B_{21},B_{22},B_{23}\\B_{31},B_{32},B_{33}\end{bmatrix}=\begin{bmatrix}\frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma-v_0\beta}{\alpha^2\beta}\\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2}+\frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma-v_0\beta)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2}\\ \frac{v_0\gamma-v_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma-v_0\beta)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2} & \frac{(v_0\gamma-v_0\beta)^2}{\alpha^2\beta^2}+\frac{v_0^2}{\beta}+1\end{bmatrix} \tag{2.6}$$

由式2.6可以看出，B矩阵是一个对称阵，那么，就意味着就可以把它表示成一个六维向量的形式，如式2.7所示：

$$b=[B_{11},B_{12},B_{22},B_{13},B_{23},B_{33}]^T \tag{2.7}$$

我们把矩阵H改写成列向量的形式，如式2.8所示：

$$h_i = [h_{i1}, h_{i2}, h_{i3}]^T \tag{2.8}$$

根据式 2.7 和式 2.8，可以改写成另一种形式，如式 2.9 所示：

$$h_i^T B h_j = v_{ij}^T b \tag{2.9}$$

其中，$v_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]$

然后，根据式 2.4 和式 2.5 可以得到式 2.10：

$$\begin{bmatrix} v_{12}^T \\ (v_{12} - v_{22}) \end{bmatrix} b = 0 \tag{2.10}$$

从式 2.9 可以看出，矩阵 V 是一个 2 行 6 列的矩阵，那么，可以知道，可以为每张照片建立两个方程组，其中共包含了 6 个方程组。我们知道，要解出所有的未知数，那么就需要与未知数同等数量的非线性方程，也就是 6 个方程组。那么，每张照片能建立两个方程组，那么就需要至少 3 张照片。b 解出之后，就能得到摄像头的内参数矩阵 A，然后，根据式 2.3，就可以解得每张照片的外部参数 R、t；

$$\begin{cases} r_1 = \lambda A^{-1} h_1 \\ r_2 = \lambda A^{-1} h_2 \\ r_3 = r_1 \times r_2 \end{cases} \tag{2.11}$$

$$t = \lambda A^{-1} h_3 \tag{2.12}$$

其中，$\lambda = \dfrac{1}{\left\| A^{-1} h_1 \right\|} = \dfrac{1}{\left\| A^{-1} h_2 \right\|}$。

2.2.2 三维重建原理

用双目摄像头观察物体[12]，那么空间上的点在两个摄像机中的投影应该是一一对应的，即每一个空间坐标点，在两个摄像机中投影出两个唯一的点。如图 2.1 所示，用两个摄像头去观察空间中的一个点。

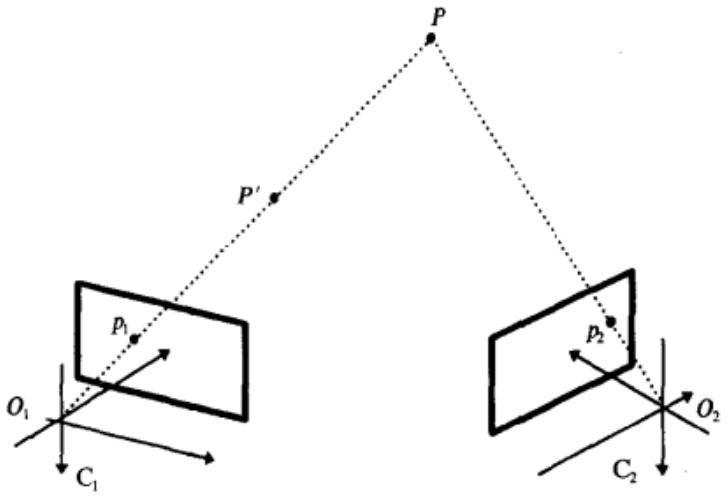


图 2.1 用双摄像头去观察空间点

因为是一一对应关系，那么，反过来，用两个摄像头中观察到的空间点的投影，那么就能唯一确定这一空间点。而三维重建就是利用这种可逆的关系。

在接下来的研究中，我们假设我们已经从两个摄像机中检测到了空间点 P 分别在它们的投影点 p1 和 p2。另外，也假设摄像机已经标定完毕，即它们的投影矩阵已经知道了，分别设为 M1 和 M2。那么，就有：

$$Z_{c1}\begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}^1 & m_{12}^1 & m_{13}^1 & m_{14}^1 \\ m_{21}^1 & m_{22}^1 & m_{23}^1 & m_{24}^1 \\ m_{31}^1 & m_{32}^1 & m_{33}^1 & m_{34}^1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.13}$$

$$Z_{c2}\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}^2 & m_{12}^2 & m_{13}^2 & m_{14}^2 \\ m_{21}^2 & m_{22}^2 & m_{23}^2 & m_{24}^2 \\ m_{31}^2 & m_{32}^2 & m_{33}^2 & m_{34}^2 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.14}$$

其中，$(u_1, v_1, 1)$ 和 $(u_2, v_2, 1)$ 分别是点 p1 和 p2 在各自摄像机中的的齐次坐标。而 $(X, Y, Z, 1)$ 则是 P 点在世界坐标系下的齐次坐标。m_{ij}^k 分别是 $\boldsymbol{M}_k$ 的第 i 行第 j 列的元素。根据求投影矩阵已经得出的结论，

$$\begin{cases} X_{v_i}m_{11}+Y_{v_i}m_{12}+Z_{v_i}m_{13}+m_{14}-v_iX_{v_i}m_{31}-v_iY_{v_i}m_{32}-v_iZ_{v_i}m_{33}=v_im_{34} \\ X_{u_i}m_{11}+Y_{u_i}m_{12}+Z_{u_i}m_{13}+m_{14}-u_iX_{u_i}m_{31}-u_iY_{u_i}m_{32}-u_iZ_{u_i}m_{33}=u_im_{34} \end{cases} \tag{2.15}$$

可以得到有关于 X、Y、Z 的 4 个线性方程：

$$\begin{aligned} (u_1m_{31}^1-m_{11}^1)X+(u_1m_{32}^1-m_{12}^1)Y+(u_1m_{33}^1-m_{13}^1)Z=m_{14}^1-u_1m_{34}^1 \\ (v_1m_{31}^1-m_{21}^1)X+(v_1m_{32}^1-m_{22}^1)Y+(v_1m_{33}^1-m_{23}^1)Z=m_{24}^1-v_1m_{34}^1 \end{aligned} \tag{2.16}$$

$$\begin{aligned} (u_2m_{31}^2-m_{11}^2)X+(u_2m_{32}^2-m_{12}^2)Y+(u_2m_{33}^2-m_{13}^2)Z=m_{14}^2-u_2m_{34}^2 \\ (v_2m_{31}^2-m_{21}^2)X+(v_2m_{32}^2-m_{22}^2)Y+(v_2m_{33}^2-m_{23}^2)Z=m_{24}^2-v_2m_{34}^2 \end{aligned} \tag{2.17}$$

根据解析几何的知识可以知道，世界坐标系的平面方程是满足线性关系的线性方程，两个平面方程的联立，能构成一个直线方程。而两个直线方程，则一定能构成一个点的方程。

如图 2.1 所示，空间点 P 是O_1p1和O_2p2的交点，那么，这一点的空间坐标必然满足式 2.16 和式 2.17 这两个方程组，因此，通过联立这两个方程组，就可以求出 P 点的世界坐标（X,Y,Z）。4 个方程，只有 3 个独立方程，而未知数的个数为 3，那么，此方程组必然有解且唯一。然而，在实际应用中，数据的采集并不是那么理想，数据往往伴随着噪声，所以，通常我们用最小二乘法来求解 X、Y、Z。

2.3 标定基本步骤

2.3.1 内参标定步骤

内参是矫正摄像头畸变的关键参数。因为这些原理已经很普遍了，为了提高效率，并不需要去编写代码完成。这里所进行的标定工作，是基于 MATLAB 里的 calibration toolbox。

（1）经过摄像头，先取出左摄像头的 25 张图片。然后进行标定。首先打开 MATLAB，然后打开 toolbox 标定工具箱。界面如图 2.2 所示。

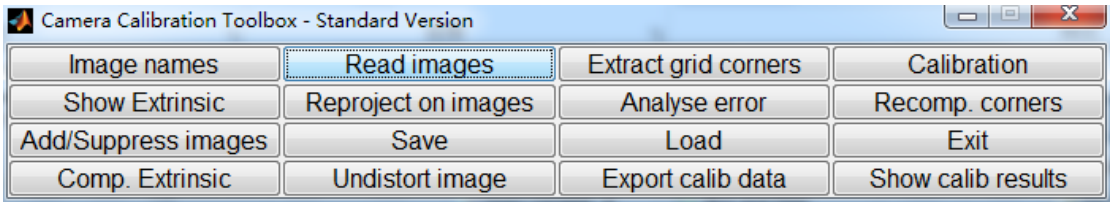


图 2.2 toolbox 标定界面

（2）进入图片所在目录，读入图片。读入图片结果如图 2.3 所示：

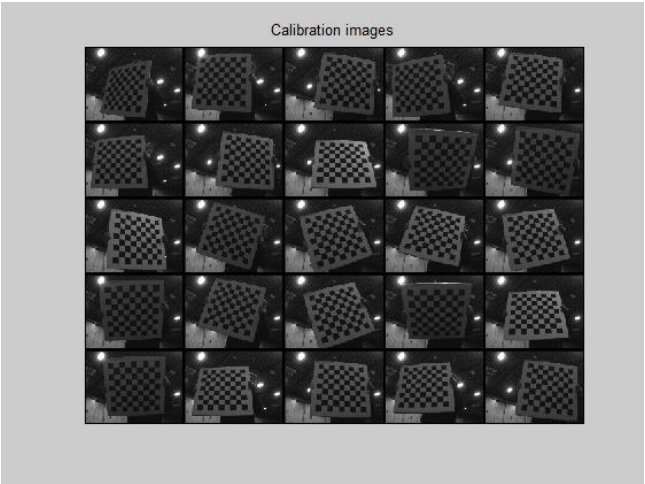


图 2.3 标定用所有图片

（3）开始提取棋盘格角点。一次找寻棋盘的最外面 4 个角点，然后，工具箱自动找到这 4 个角点范围内所有的角点。结果如图 2.4 及 2.5 所示。

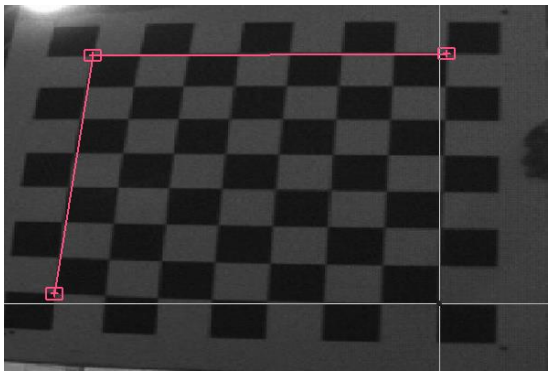


图 2.4 取角点的过程

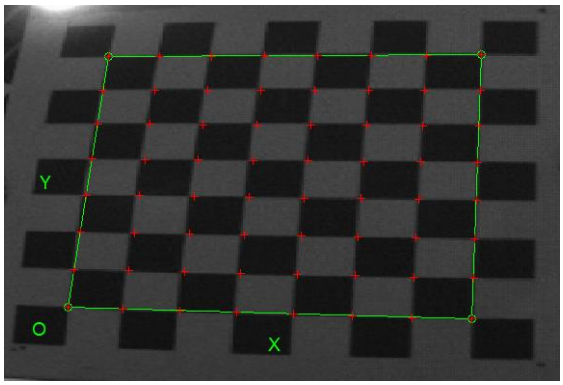


图 2.5 取角点的结果

（4）按照这个方法，依次找到 25 张棋盘的图片。然后运行 Calibration，MATLAB 便开始运行计算得出摄像机的内参。结果如图 2.6。不仅如此，标定工具箱还能为我们画出每张图片的矫正信息，如图 2.7。然后将结果保存下来。

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 765.66651   768.00883 ] ? [ 3.35447   3.64046 ]
Principal point:       cc = [ 362.25784   217.06160 ] ? [ 3.15185   3.09146 ]
Skew:             alpha_c = [ 0.00000 ] ? [ 0.00000  ]   => angle of pixel axes = 90.00000 ? 0.00000 deg
Distortion:            kc = [ -0.39666   0.29761   0.00043   -0.00018  0.00000 ] ? [ 0.01174   0.06547
Pixel error:          err = [ 0.18852   0.18373 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).


Number(s) of image(s) to show ([] = all images) =
Pixel error:      err = [0.18852   0.18373] (all active images)
```

图 2.6 标定内参结果

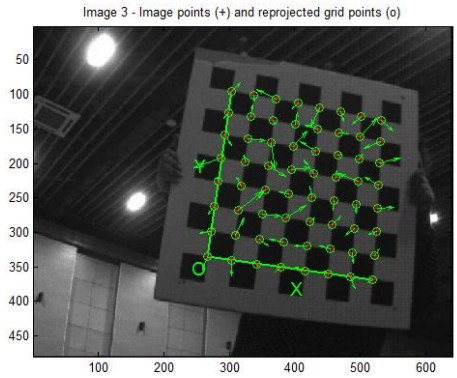


图 2.7 原图像经标定后的像素点的矫正图片

（5）利用同样的方法，标定右摄像机的内参，并将结果保存下来。结果如图 2.8 所示。

```
    fc = [ 724.81546   724.85480 ] ? [ 6.13905   5.99094 ]
    cc = [ 361.63039   284.98337 ] ? [ 4.99012   5.75512 ]
lpha_c = [ 0.00000 ] ? [ 0.00000  ]   => angle of pixel axes = 90.00000 ? 0.00000 degrees
    kc = [ -0.40199   0.24396   -0.00245   0.00165  0.00000 ] ? [ 0.01808   0.09114   0.00162   0.00166  0.00000 ]
   err = [ 0.21066   0.23757 ]
```

图 2.8 右摄像头内参结果

2.3.2 外参标定基本步骤

外参是确定图片上的二维像素点与三维坐标点一一对应关系的关键参数。

（1）先载入上一阶段计算出的摄像机的内参信息。然后再运行 calibration，这样，摄像机的内参信息就被读入到 MATLAB 中，然后开始计算摄像机的外参，运行 Comp.Extraction。读入在左摄像头获取的外参标定新照片，如图 2.9 所示。

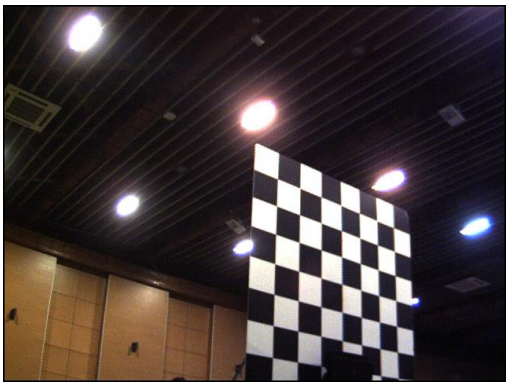


图 2.9 左摄像机外参标定图片

（2）然后输入棋盘格的长宽，得出摄像机的外参。将外参保存成 left_ext.mat;

（3）同样的，标定右边摄像头的外参，保存成 right_ext.mat。标定图片如图 2.10。

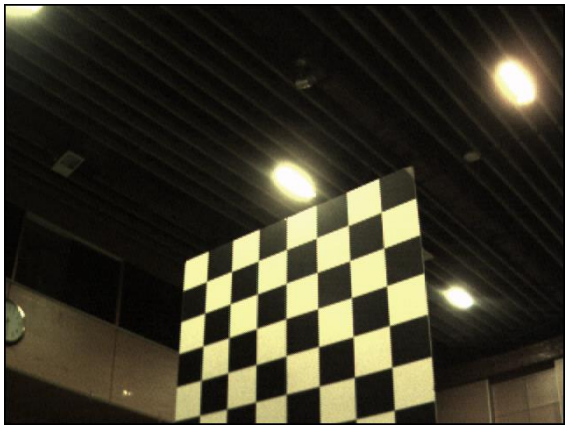


图 2.10 右摄像机外参标定图片

2.3.3 三维重建

分别读入左摄像机和右摄像机的外参，然后编写 MATLAB 程序，进行二维点到三维点的转换。算法的实现：

```
double R[3][3] = RotationMatrix;
double T[3] = Transform;
double fc_left[2] = fc_l;
```

```
double fc_right[2] = fc_r;
double cc_left[2] = cc_l;
double cc_right[2] = cc_r;
double kc_left[5] = kc_l;
double kc_right[5] = kc_r;
double Rinv[3][3] = RotationMatrixext;
double T_ext[3] = Transformext;
double O_w[3] = { -1750.0,1742.0, 0.0 };//  此处数值为昨晚那个桌高+1000.0
double Rme[3][3] = RotationMatrixcorr;
double Tme[3] = Transformcorr;
//data sheet
//first L/R normalmaping
//turn to real image point
double xyccL[2], xyccR[2];
xyccL[0] = (uvLx - cc_left[0]) / fc_left[0];
    xyccL[1] = (uvLy - cc_left[1]) / fc_left[1];
    xyccR[0] = (uvRx - cc_right[0]) / fc_right[0];
    xyccR[1] = (uvRy - cc_right[1]) / fc_right[1];
    //undishortion
    double r_2[2], k_radial[2], delta_x[2], delta_y[2], xdccL[2], xdccR[2];
    xdccL[0] = xyccL[0];
    xdccL[1] = xyccL[1];
    xdccR[0] = xyccR[0];
    xdccR[1] = xyccR[1];
    for (int kk = 0; kk < 20; kk++)
    {
```

```
            r_2[0] = pow(xdccL[0], 2) + pow(xdccL[1], 2);
            k_radial[0] = 1.0 + kc_left[0] * r_2[0] + kc_left[1] * r_2[0] * r_2[0] + kc_left[4] *
r_2[0] * r_2[0] * r_2[0];
            delta_x[0] = 2.0*kc_left[2] * xdccL[0] * xdccL[1] + kc_left[3] * (r_2[0] + 2 *
pow(xdccL[0], 2));
            delta_x[1] = kc_left[2] * (r_2[0] + 2 * pow(xdccL[1], 2)) + 2 * kc_left[3] *
xdccL[0] * xdccL[1];
            xdccL[0] = (xyccL[0] - delta_x[0]) / k_radial[0];
            xdccL[1] = (xyccL[1] - delta_x[1]) / k_radial[0];
            r_2[1] = pow(xdccR[0], 2) + pow(xdccR[1], 2);
            k_radial[1] = 1.0 + kc_right[0] * r_2[1] + kc_right[1] * r_2[1] * r_2[1] +
kc_right[4] * r_2[1] * r_2[1] * r_2[1];
            delta_y[0] = 2.0*kc_right[2] * xdccR[0] * xdccR[1] + kc_right[3] * (r_2[1] + 2 *
pow(xdccR[0], 2));
            delta_y[1] = kc_right[2] * (r_2[1] + 2 * pow(xdccR[1], 2)) + 2 * kc_right[3] *
xdccR[0] * xdccR[1];
            xdccR[0] = (xyccR[0] - delta_y[0]) / k_radial[1];
            xdccR[1] = (xyccR[1] - delta_y[1]) / k_radial[1];
        }
        xyccL[0] = xdccL[0];
        xyccL[1] = xdccL[1];
        xyccR[0] = xdccR[0];
        xyccR[1] = xdccR[1];
        //second tripoint
        double uL[3];
        for (int uli = 0; uli < 3; uli++)
```

```
        uL[uli] = R[uli][0] * xyccL[0] + R[uli][1] * xyccL[1] + R[uli][2];
    double n_xtL2, n_xtR2;
    n_xtL2 = pow(xyccL[0], 2) + pow(xyccL[1], 2) + 1;
    n_xtR2 = pow(xyccR[0], 2) + pow(xyccR[1], 2) + 1;
    double DD = n_xtL2*n_xtR2 - pow((uL[0] * xyccR[0] + uL[1] * xyccR[1] + uL[2]),
2);
    double dot_uT = uL[0] * T[0] + uL[1] * T[1] + uL[2] * T[2];
    double dot_xtR2T = xyccR[0] * T[0] + xyccR[1] * T[1] + 1 * T[2];
    double dot_xtR2u = uL[0] * xyccR[0] + uL[1] * xyccR[1] + uL[2];
    double NN1 = dot_xtR2u*dot_xtR2T - n_xtR2*dot_uT;
    double NN2 = n_xtL2*dot_xtR2T - dot_uT*dot_xtR2u;
    double ZtL = NN1 / DD;
    double ZtR = NN2 / DD;
    double X1[3], X2[3], Rr[3];
    for (int xi = 0; xi < 2; xi++)
    {
        X1[xi] = xyccL[xi] * ZtL;
        Rr[xi] = xyccR[xi] * ZtR - T[xi];
    }
    X1[2] = ZtL;
    Rr[2] = ZtR - T[2];
    for (int ri = 0; ri < 3; ri++)
        X2[ri] = R[0][ri] * Rr[0] + R[1][ri] * Rr[1] + R[2][ri] * Rr[2];
    double Pl[3], Pr[3], Pr1[3], P[3],Pc[3];
    for (int pi = 0; pi < 3; pi++)
        Pl[pi] = 0.5*(X1[pi] + X2[pi]);
```

```
        for (int pir = 0; pir < 3; pir++)
        {
            Pr[pir] = R[pir][0] * Pl[0] + R[pir][1] * Pl[1] + R[pir][2] * Pl[2] + T[pir];
            Pr1[pir] = Pr[pir] - T_ext[pir];
        }
        for (int piw = 0; piw < 3; piw++)
            P[piw] = Rinv[piw][0] * Pr1[0] + Rinv[piw][1] * Pr1[1] + Rinv[piw][2] * Pr1[2] +
O_w[piw];
        //Rectify
        //for (int pic = 0; pic < 3; pic++)
            //Pc[pic] = Rme[pic][0] * (P[0]+3050.0) + Rme[pic][1] * (-P[2]+6700.0) +
Rme[pic][2] * P[1] + Tme[pic];
        pw[0] = P[0]+3050.0;
        pw[1] = -P[2]+6700.0;
    pw[2] = P[1];
```

2.4 本章小结

本章主要内容是双目摄像机的标定技术与三维重建技术，这是羽毛球检测系统的第一步，也是关键一步，只有三维重建模型准确了，后面的工作才是有用的。本章首先阐述了双目视觉的硬件选型，从摄像机的原理分析，确定了摄像头以及镜头的型号。然后阐述了摄像机标定原理以及三维重建的原理，解释了三维重建的可能性。最后，本文通过 MATLAB 标定工具箱，完成摄像机的标定工作，并用 C++编程实现了二维点到三维点的转换工作。

3 通信系统设计

3.1 蓝牙通信模块

3.1.1 蓝牙技术概述

因为比赛规定，与机器人无线通信的各个部件除红外、蓝牙外不能采用其他方式，因此，本系统选用蓝牙进行数据之间的传输。

蓝牙是一种用于设备之间进行短距离通信的无线电技术[13]。国际上规定，使蓝牙在2.4GHz 频段下进行工作，其传输数据的速率为 1Mbps。蓝牙技术能将许多个蓝牙节点连接起来，组成微微网。在微微网中，只能有一个节点作为主站点，主站点能连接并控制多达 7 个蓝牙从站。将多个微微网连接起来，就可以组成一个散射网；从安全上考虑，每一个蓝牙模块儿，都分别分配了唯一一个遵循 IEEE802 标准的设备地址，从而保证数据能够进行安全有效的传输。而且蓝牙具有协议开放、抗干扰能力强、功耗低的优势，所以，这也是此系统选用蓝牙进行数据传输的一个重要原因。

3.1.2 蓝牙协议概述

蓝牙协议栈[14]就是蓝牙协议的集合，在协议栈里，各种协议层次分明，分工明确。如图 3.1 所示。

蓝牙协议可按照不同的功能，可以被划分成四个类别：

（1）蓝牙核心协议主要包括基带 BB、链接管理 LMP、链接控制和适配 L2CAP、服务发现 SDP 协议。BB 实现了蓝牙数据与帧之间的传输。传输的方式主要有分组交换、电路交换；LMP 可以实现链路链路的建立、加密、控制等；L2CAP 的功能是桥接上、下层协议之间长度不同的分组、有拆装数据、协议服用、控制服务质量等功能。SDP 协议是上层发现和解释网络中可用的协议。

（2）蓝牙串口替代协议：串口仿真协议是在串口基础之上，建立起的传统应用的特定接口环境。

（3）蓝牙选用协议：有 UDD/TCP/IP、OBEX、Vcal 等。

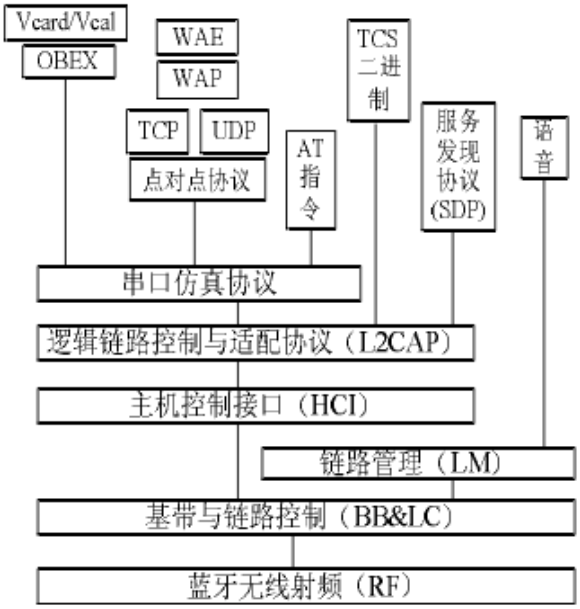


图 3.1 蓝牙协议模型

3.1.3 蓝牙通信

蓝牙接收和发送信号的完整过程如图 3.2 所示。发送前必须对传输数据进行载波调制，在接收之后，对收到的高频信号进行解调制。

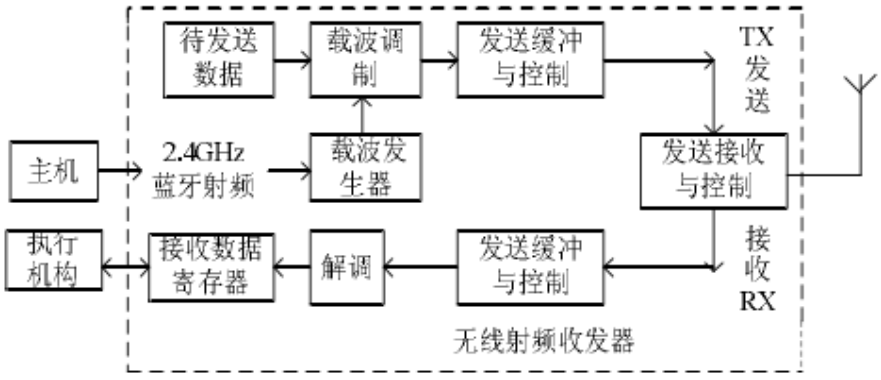


图 3.2 蓝牙通信过程

在蓝牙数据接收与发送中，有效数据信息和有效的控制信息可以同时在信道上进行传送，但却分别使用特定的接口。其中，控制信息主要是用于控制无线射频收发器的收发动作。在此过程中，发射时的控制信息用于控制信息的载波频率、数据信息流向、发

射的功率级别等，而接收时，主要控制接收到的信号的解调制、分析信噪比的大小、接受信息强度等。

由于蓝牙通信是基于无线技术的，所以，在发送、接收数据的时候，会出现漏数的情况，因为发送的格式是规定好的，那么，如果在发送、接收过程中漏数，会直接导致转换的结果错误，从而使得机器人会跑向未知的地方，发生碰撞。因此，在蓝牙通信过程中，此系统增加用于校验的包头、包尾。只有当包头、包尾都正确且都在正确的位置上（即在固定的数组中）时，那么，才认为接收的数据是正确的。另外，根据蓝牙的发送接收格式只能是 8 位二进制码，所以任何数据都要进行拆分，因为此系统涉及到的数字在 0~6000 之间，所以只需要拆分成两个 8 位二进制码即可。核心代码如下，将目标位置 dstx、dsty 以及击打时间 dsttime 分成高八位和低八位分别发送。

```
unsigned char d = 0x66;                    port.WriteData(&d,1);
d = (unsigned short)(dstx>>8);       port.WriteData(&d,1);
d = (unsigned short)(dstx);          port.WriteData(&d,1);
d = (unsigned short)(dsty>>8);       port.WriteData(&d,1);
d = (unsigned short)(dsty);          port.WriteData(&d,1);
d = (unsigned short)(dstime>>8);     port.WriteData(&d,1);
d = (unsigned short)(dstime);        port.WriteData(&d,1);
d = 0x77;                                  port.WriteData(&d,1);
d = 0x88;                                  port.WriteData(&d,1);
cout<<"发送成功："<<endl;
```

3.2 网络通信模块

3.2.1 网络通信概述

由于双目摄像头架在场外，所以双目摄像头的数据传输不会受到大赛规则的限制。考虑到 PC 机的 USB 的接口有限，不能全接上蓝牙，另外，蓝牙的传输速度并不是很快，综合考虑，决定用组建局域网[15]进行数据传输。

局域网的产生与发展：

随着计算机网络的发展，LAN 逐渐成为一个活跃且重要的计算机领域。LAN 从 20 世纪 70 年代开始发展的。1972 年加州大学研制了 Newhall loop 网，Xerox 公司研究中心首次提出了总线结构的实现版的以太网。1974 年，剑桥大学计算机实验室创建了剑桥环。20 世界 80 年代，LAN 的衍生版纷纷出现并开始投入市场。

超大规模集成电路（VLSI）技术的快速发展极大地推动了计算机技术的发展，从而导致了微型计算机价格的大幅度下降，包括 LAN 网络接口卡以及其他联网设备的价格也随之下降，这使得计算机局域网 PC-LAN 的发展。

目前，LAN 技术发展非常迅速，其中最著名的要属以太网（Ethernet），经过 30 年的发展，其速度已经从最初的 10Mb/s 提高到了现在的 10Gb/s。如今，以太网已经成为 LAN 的主流网络，全世界的网络基本上都是以太网。

千兆以太网的结构：

IEEE802.3 以太网技术的发展催生了千兆以太网，IEEE 制定的 1000Base-X 光纤和 1000Base-T 双绞线两种千兆以太网的物理层标准。其协议结构如图 3.3 所示。

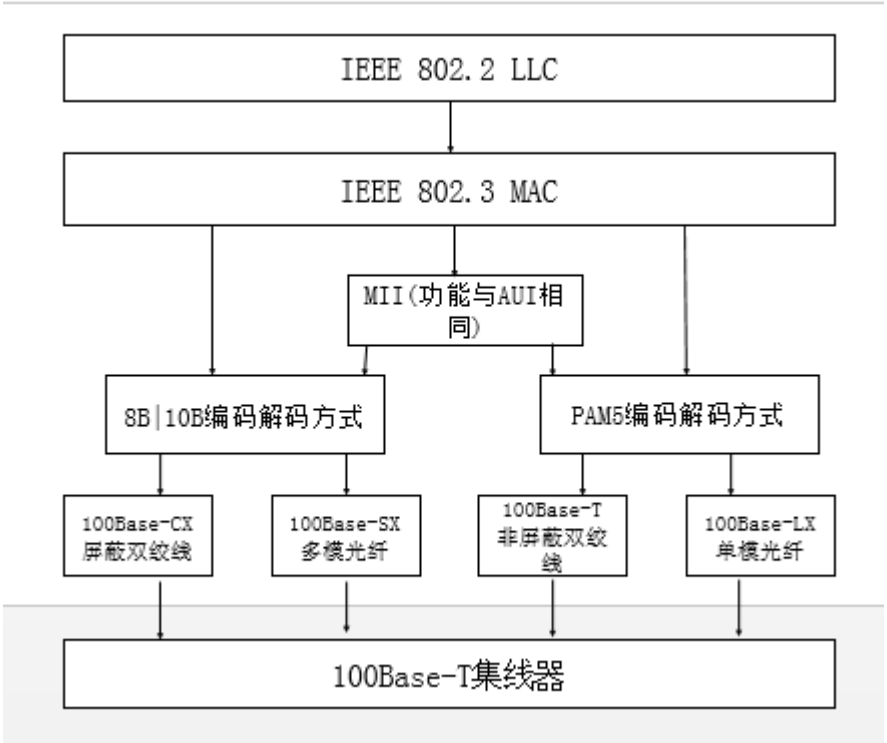


图 3.3 千兆以太网协议结构

（1）1000Base-X 包括 3 种不同的媒体

1）1000Base-LX:长波长光纤网段，工作在波长为 1270~1355nm 的长光纤激光传输器。可用 10um 的纤芯的单模光纤和 62.5、125um、50、125um 的多模光纤。

2）1000Base-SX：短波长光纤网段，工作在波长为 770~860nm 的光纤激光传输器。可用于 62.5、125um、50、125um 的多模光纤。

3）1000Base-CX:用的是高质量的 STP。

（2）1000Base-T 的媒体

这个标准定义了 4 对 5 类 UTP，支持 100m 距离的数据传输。千兆以太网在全双工和半双工的模式下的的各种物理层的链路长度表。

表 3.1 千兆以太网在各物理层的链路长度

物理层标准	50umMMF	62.5umMMF	10umSMF	150STP	5 类 UTP
	半双工/全双工	半双工/全双工	半双工/全双工	半双工/全双工	半双工/全双工
1000Base-SX	110/550	110/275			
1000Base-LX	110/550	110/550	110/550		
1000Base-CX				25/25	
1000Base-T					100/100

表 3.1 中可以看到，1000Base-CX 的链路长度却只有 25m，这个原因是因为该链路的自身媒体的限制。

3.2.2 网络协议

计算机网络就是有多个点相互连接，组成一个网络，节点之间能够进行数据的交换与信息的控制。要保证点与点之间的数据传输做到有条不紊，必须规定一种统一的协议，这个协议必须明确规定用来交换的数据的时序与格式。这些被用来规定网络数据传输与交换的协议、约定就被称为网络协议。一个完整的网络协议通常是由三个要素组成：（1）语法，是规定用户数据交换与信息控制的格式与结构，还包括了数据出现的意义与顺序。（2）语义，它解释了数据所占的每一比特流的含义，规定了信息控制、动作完成以及做出的响应。（3）时序，它是对事件发生顺序的说明与规定。由此可以看出，计算机网络的组成不能缺少网络协议这部分。

TCP/IP 协议是物理网上的一个比较完整的网络协议。其特点有：（1）开放的标准，网络协议不依赖于某种特定的计算机硬件或者操作系统。（2）不依赖于网络硬件，在局域网、广域网、互联网都可以很好地应用。（3）统一分配网络地址，这使得在计算机网络中，每个 IP 都能被分配到唯一的地址，防止数据传输冲突。（4）标准化的高层协议，能支持多种可靠的用户服务。

其模型包含了 4 个部分，如图 3.4 所示。

图 3.4 TCP/IP 分层模型

（1）应用层：这一层主要负责跟用户的直接交互，为用户的应用程序提供服务，包括一些高层协议，如文件传输协议、域名系统服务等。

（2）传输层：主要负责两台主机之间的端到端的通信服务，这一层有两个协议，一个是 TCP，它是一种面向连接的可靠的通信协议。通过此协议，可以将数据准确的从

一端传输到另一端。另一个是 UDP，它是一种非面向链接的通信协议，类似于广播，不能确定是否另一端成功接收，优点是能进行一对多的传输。

（3）互连网络层：主要处理为传输层分组的发送请求、协议数据加上报文与解析报文，另外能处理互联路径、数据流量控制以及拥塞情况。

（4）网络接口层：主要负责通过网络路径发送以及接受 IP 数据报。

3.2.3 Windows 下的 socket 编程实现

（1）套接字概述[16]

对于一般的系统而言，应用程序与系统程序工作在不同的保护模式下，应用程序通常下不被允许访问系统资源，这样做的目的是防止一些应用程序非法访问与修改系统资源。为此，系统会为应用程序提供接口，对于网络系统而言，需要为网络应用程序提供网络编程接口实现计算机之间的网络通信。套接字就是操作系统为应用程序提供的一种网络编程接口。图 3.5 直观的显示了套接字在 TCP/IP 协议中的位置。从图 3.5 可以很明显的看出，套接字是在传输层之上，也就是说，套接字有效地屏蔽了底层协议，降低了编程的难度。这就犹如打电话，无论电话之间的传输有多么复杂，只要拿起听筒，那么两个人就能实现通话，套接字也是一样，通过套接字接口，我们能将数据从一台电脑传输到另一台电脑，而不用管传输过程的复杂性。

（2）Socket 编程

这里用的是 TCP 通信协议来进行编程。Socket 编程模型如图 3.5 所示。

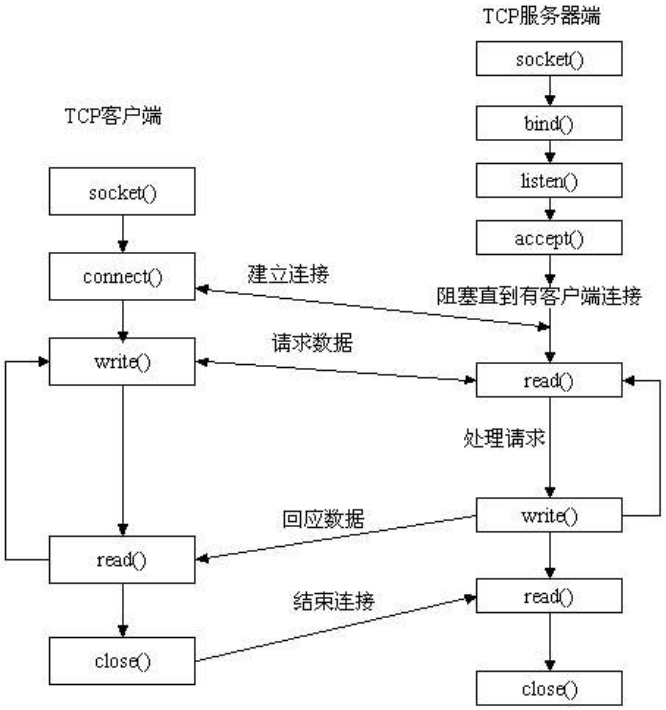


图 3.5 TCP 协议 Socket 编程模型

首先，服务器创建 socket，函数如下：

SOCKET WSAAPI socket(_In_ int af,_In_ int type,_In_protocol);

第一个参数是协议类型，可以使 IPV4、IPV6 协议或者是一些其他更底层的协议。第二个参数是套接字接口类型。一般选择 SOCK_STREAM。第三个参数是选择底层协议，比如 TCP 协议或者 UDP 协议。

然后是绑定 socket 接口，函数如下：

int bind(int sockfd, const struct sockaddr *addr,socklen_t *addrlen)

第一个参数是 socket()创建的套接字。第二个参数是套接字的数据信息。一般包括协议的端口、通信协议等。第三个参数是套接字的数据大小，一般用 sizeof 来获取。

最后客户端连入服务器，函数如下：

int connect(int sockfd, const struct sockaddr* server_addr, socklen_t addrlen)

参数与 bind()差不多，但是第二个参数的端口信息要写上要连接的服务器的信息。

下面就是数据的传输了。其函数是：

int recv(int sockfd,void *buf,int len,int flags)

int send(int sockfd,void *buf,int len,int flags)

两个函数相近，recv 是用于接收数据的，而 send 则用于发送数据。

第一个参数是创建的套接字接口，第二个是要发送或者接收数据的存储数组。

Socket 编程的通信实现：

服务器核心代码：

```
WORD sockVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    if(WSAStartup(sockVersion, &wsaData)!=0)
    {
        return 0;
    }
    //创建套接字
    if(slisten == INVALID_SOCKET)
    {
        printf("socket error !");
        return 0;
    }
    slisten= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    //绑定 IP 和端口
    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(8888);
    sin.sin_addr.S_un.S_addr = INADDR_ANY;
    if(bind(slisten, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET_ERROR)
    {
```

```
        printf("bind error !");
    }
    //开始监听
    if(listen(slisten, 5) == SOCKET_ERROR)
    {
        printf("listen error !");
        return 12;
    }
    printf("等待连接...\n");
    sClient = accept(slisten, (SOCKADDR *)&remoteAddr, &nAddrlen);
    if(sClient == INVALID_SOCKET)
    {
        printf("accept error !");
    }
    printf("对方 ip：%s \r\n", inet_ntoa(remoteAddr.sin_addr));
```

客户端核心代码：

```
WORD sockVersion = MAKEWORD(2,2);
    WSADATA data;
    if(WSAStartup(sockVersion, &data) != 0)
    {
        return 0;
    }
    if(sclient == INVALID_SOCKET)
    {
        printf("invalid socket !");
        return 0;
```

```
    }
  sclient= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in serAddr;
    serAddr.sin_family = AF_INET;
    serAddr.sin_port = htons(8888);
    serAddr.sin_addr.S_un.S_addr = inet_addr(ServerAdress);
  cout<<"44"<<endl;
    if (connect(sclient, (sockaddr *)&serAddr, sizeof(serAddr)) == SOCKET_ERROR)
    {
        printf("connect error !");
        closesocket(sclient);
      while(1);
        return 0;
    }

  char recData[3],*sendData="2";
  int ret=0;
  while(!recv(sclient,recData,1,0));
  if(recData[0]='1')
  {
     GetImage();
  }
```

通过上述程序搭建起的局域网，使得两个摄像机之间能够进行数据的交换，从而将一个从摄像机中得到的羽毛球的位置信息发送给主摄像头，主摄像头利用三维重建技术，准确求得飞行中的羽毛球的位置信息。

核心程序如下：

```
while(!recv(sClient,recData,200,0));
point2D_l.x=(recData[1]-48)*100+(recData[2]-48)*10+(recData[3]-48);
point2D_l.y=(recData[4]-48)*100+(recData[5]-48)*10+(recData[6]-48);
```

局域网传输速度很快，因此，此系统利用局域网技术，使得主摄像头能够控制两个摄像头能够同时对场景进行拍摄，从而确保得出的位置信息的可靠性。

核心程序如下：

```
char * sendData = "1";（主摄像头）
send(sClient, sendData, strlen(sendData), 0);
while(!recv(sclient,recData,1,0));（从摄像头）
    if(recData[0]='1')
    {
        GetImage();
    }
```

3.3 本章小结

本章主要介绍了系统所用的两种通信模式：蓝牙通信和网络通信。通信系统是羽毛球检测系统的相互连接的重要部分，有了通信模块，分开的各个模块才能变成一个整体。本章第一部分，讲述了蓝牙的原理和以及蓝牙的选择，接着讲述蓝牙配对的实现可能。本章第二部分，讲述了局域网的概况，并解释了局域网的原理，然后利用 Windows socket 实现了了局域网通信的实现。

4 羽毛球检测与特征提取

4.1 形态学处理原理及效果

4.1.1 形态学概念

形态学[17]一词通常表示生物学的一个分支，该分支主要研究动植物的形态和结构。而在图像处理领域，表示数学形态学的内容，将数学形态学作为工具从图像中提取表达和描述区域形状的有用图像分量，如边界、骨架、凸壳等。

4.1.2 形态学基本方法

（1）腐蚀

像素集合 A 和 B 属于Z^2，用$A\Theta B$表示为 B 对 A 的腐蚀作用，表达式如式 4.1。

$$A\Theta B=\{z|(B)_z\subseteq A\} \quad (4.1)$$

从上式中可以看到，B 对 A 的腐蚀作用，其实就是用 Z 平移的 B 包含进 A 中的所有点 z 的集合.

下面用一个像素为 100*100 的集合 B 来对 200*200 的矩形集合 A 进行腐蚀操作。图 4.1 是集合 A 未经处理的图像，图 4.2 是经过腐蚀之后的图像。可以很明显的看出来，腐蚀后的图像，矩形区域明显变小。

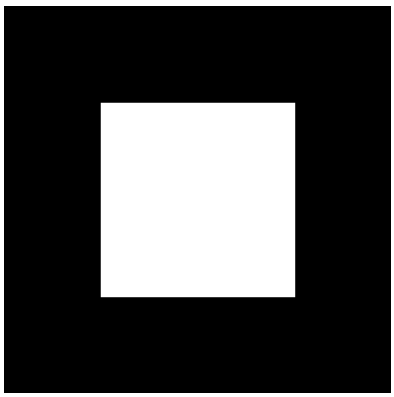


图 4.1 集合 A

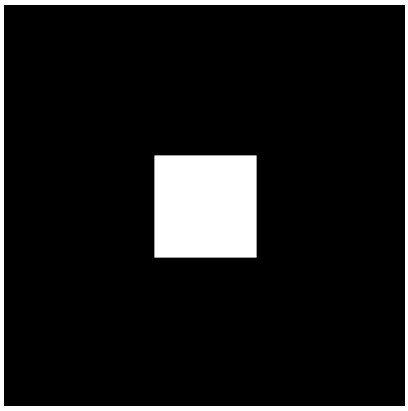


图 4.2 腐蚀后的集合 A

（2）膨胀

像素集合 A 和 B 属于Z^2，用$A\oplus B$来表示 B 对 A 的膨胀作用，表达式如式 4.2

$$A \oplus B = \left\{ z \mid (\hat{B})_z \cap A \neq \varnothing \right\} \tag{4.2}$$

从上式中可以看到，B 关于原点对称的集合$(\hat{B})$，用 z 平移的$(\hat{B})$为基础，包含进所有 z 平移的集合，这样$(\hat{B})$和 A 之间至少有一个像素元素是重叠的。

下面用一个像素为 100*100 的集合 B 来对 200*200 的矩形集合 A 进行膨胀操作。图 4.3 是一个未经处理的矩形集合 A，图 4.4 是经集合 B 膨胀后的集合 A，很明显可以看出，矩形区域明显增大。

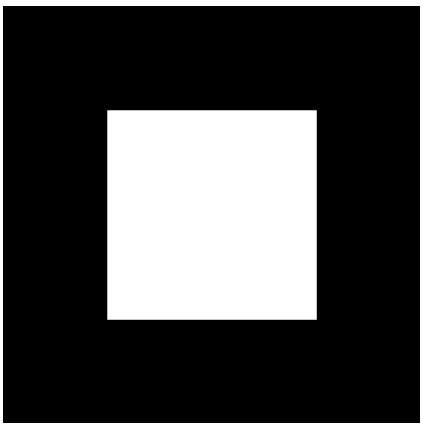


图 4.3 集合 A

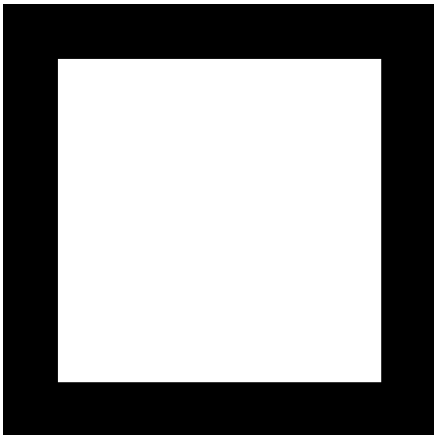


图 4.4 集合 A 经过膨胀之后的图像

（3）开操作

正如上边所讨论的那样，膨胀操作会扩大图像的组成部分，而腐蚀操作正好相反，会缩小图像中的组成部分。这里讨论的开操作和闭操作，是以这两个基本操作为基础的，进行组合运算。

结构元集合 B 对 A 进行开操作，表示为$A\circ B$，其表达公式如式 4.3。

$$A\circ B=(A\Theta B)\oplus B \tag{4.3}$$

正如公式所表示的，开操作是用 B 对 A 进行腐蚀操作，然后在进行膨胀操作。从操作的过程来看，先缩化，再粗化，则可以有断开细小的连接部分的功能。

下面用一个像素为 17*17 的结构元集合 B 对集合 A 进行开操作。图 4.5 是未经处理的原图像，可以看到，图像当中，两个矩形之间，有一条细线进行连接。而图 4.6 是经过开操作处理后的图像，可以看到，两个矩形之间的细线被消除了。

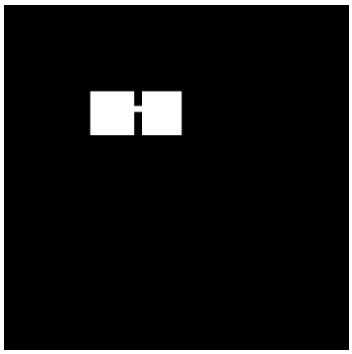


图 4.5 集合 A 的原图像

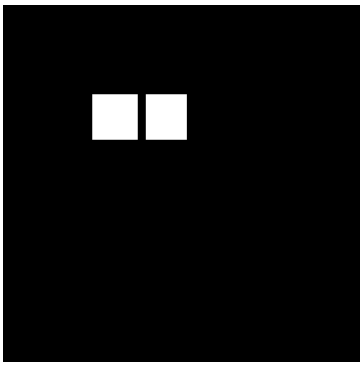


图 4.6 集合 A 经过开操作之后的图像

（4）闭操作

结构元集合 B 对 A 进行闭操作，表示为$A\bullet B$，其表达公式如式 4.4.

$$A \bullet B = (A \oplus B) \Theta B \tag{4.4}$$

正如公式显示的，闭操作是开操作的反运算，先进行膨胀操作，再进行腐蚀操作。从这个过程来看，闭操作有强化细小连接部分的功能。

下面用一个像素为 17*17 的结构元集合 B 对对集合 A 进行闭操作。其中，集合 A 的原图像如图 4.5，经过处理后的图像如图 4.7 所示，正如前面定义的那样，由于细线的膨胀，使得两个矩形区域合并成一个更大的区域了。

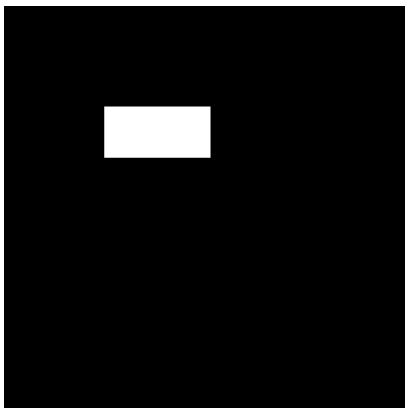


图 4.7 集合 A 经过闭操作之后的图像

这些滤波器常常用在检测物体方向。开操作是将被误分割成碎片的物体重新连接起来了，而开操作是将图像噪点引起的小像素块消除。综上分析，在视频序列中应用开闭操作通常能达到理想的效果。如果我们用于测试的二值图像相继进行闭、开操作，获取到的图像中将只显示场景中的主要目标物体。如果优先处理噪点，也可以先开后闭运算，但这个方法可能会去除一些分散的目标。

4.1.3 形态学 OpenCV 的实现

有关膨胀的函数是[18]：

void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue()).

其中，src 是输入图像，这个便是上述说的集合 A，dst 是输出图像，kernel 是膨胀因子这个即为上述说的集合 B，anchor 指的是 kernel 的算子中心的位置，这里默认为（-

1，-1），意味着是算子从中心开始的。Iterations 指的是迭代次数，迭代次数越多，那么膨胀的越厉害。

有关腐蚀的函数是：

void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue())

参数与膨胀是一样的。

然而，膨胀腐蚀只是一种处理手段，我们通常想得到是对原来图像的增强图像。也就是说，我们往往要将膨胀腐蚀一起来用，也就是上述提到到开操作和闭操作。

有关操作的函数：

void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue())

其中，src、dst、iterations 和开闭操作是一样的，op 是指操作的类型，有 MORPH_OPEN（开操作），MORPH_CLOSED（闭操作），MORPH_GRADIENT（求梯度操作），MORPH_TOPHAT（礼帽操作），MORPH_BLACKHAT（黑帽操作）。由于梯度操作和礼帽、黑帽操作本系统没有涉及，这里就不赘述了。

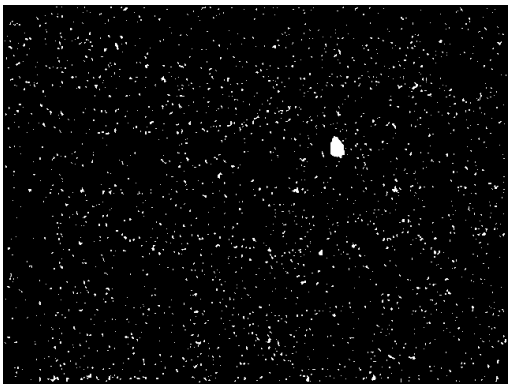


图 4.8 未经处理的羽毛球图像

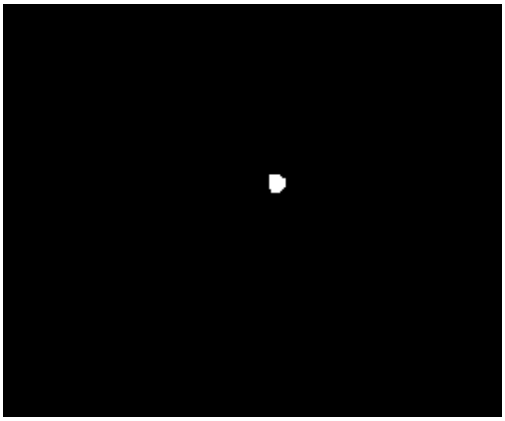


图 4.9 形态学操作之后的羽毛球图像

如图 4.8 所示，提取出的羽毛球图像没有经过形态学处理的原始图像，可以看出，由于摄像头硬件，环境变化等原因，产生的噪音非常的多，如果直接对此图片进行处理，可想而知，计算量会是非常的大，无法满足实时性的要求。图 4.9 是经过开操作之后的图像，很明显的看出，羽毛球周围的噪声没有了，凸显出了羽毛球一个物体的特征，这对后面的轮廓提取，特征判断有了很大的帮助。因此，形态学操作是很有必要的。以下列出了膨胀腐蚀以及开、闭操作的核心代码。

膨胀腐蚀核心代码：

```
int main()
{
    Mat img(400,400,CV_8UC1,Scalar(0));
    rectangle(img,Point(100,100),Point(300,300),Scalar(255),-1);
    Mat img_erode = img.clone();
    Mat img_dilate = img.clone();
    Mat temp;
    Mat kernel(100,100,CV_8U,cv::Scalar(1));
    erode(img,img_erode,kernel);
    imshow("img_erode",img_erode);
    dilate(img,img_dilate,kernel);
```

```
        imshow("img_dilate",img_dilate);
        imshow("img",img);

        waitKey(0);
}
```

开、闭操作核心代码：

```
cvtColor(img,img_next,CV_BGR2GRAY);
Mat img_contour_no = img_pre.clone();
morphologyEx(img_pre,img_contour,MORPH_OPEN,element);
morphologyEx(img_pre,img_contour,MORPH_CLOSED,element);
```

4.2 运动目标检测

4.2.1 光流法

光流法[19]最先是 Gibson 在 1950 年第一次提出来的。光流就是观察物体在成像表面上引起的像素发生位移的瞬时速度，主要是应用了相邻两帧之间的关联性找到帧与帧之间的一一对应关系以及图像序列中的像素在时间轴上的变化，进而得出两帧之间物体发生的位移信息的一种方法。通常来讲，光流就是由相机运动、前景物体自身的移动或者是这两个原因共同产生的运动。研究光流场主要目的是能够从图片序列中得到不能直接通过摄像机得到的的运动场。运动场，说白了，就是目标物体在真实的三维世界中的运动，而光流场，就是运动场投影到二维平面上的结果。

光流法是用于目标检测与目标分割的经典算法之一。在平滑性的约束条件下，光流法能根据图像序列的时间和空间梯度进行估算得出运动场，通过计算、分析运动场的变化，将目标物体与背景分离开来。

光流法的假设前提是（1）前后两帧图像之间的亮度恒定；（2）前后帧的读取时间是连续或者两帧之间目标位移较小；（3）在子图像中，对应当像素点的运动是相同的。将这三个条件作为假设前提，随着时间轴的不断延伸，目标物体就会表现出理想的光流特性。

光流法目标检测的基本原理：在图像系列中，每一帧中的像素点都对应这一个速度矢量，它们共同构成了一个运动矢量场。在某一时刻，三维目标物体上的空间点与投影到二维图像上的像素点是一一对应的。根据二维图像中所表现的速度矢量特征，就能分析出图像的动态特性。如果图像中出现运动着的物体，则前景与背景之间必然会发生相对运动，因此，前景在二维图像上产生的速度矢量一定会与背景速度矢量是不一样的，因此，根据这个变化，可以较准确地求出运动目标。

光流方程的数学表达式如式 4.5。

$$I(x(t),y(t),t)\approx I(x(x+\Delta t),y(t+\Delta t),t+\Delta t) \tag{4.5}$$

$I(x(t),y(t),t)$ 是一幅图像中连续空间的灰度函数.而将式 1-1 按泰勒级数展开，就能得到光流约束方程如式 4.6 以及式 4.7

$$I(x(x+\Delta t),y(t+\Delta t),t+\Delta t)\approx I(t)+\frac{dI(t)}{dt}\Delta t+O^2(\Delta t) \tag{4.6}$$

$$\frac{dI}{dt}=0 \tag{4.7}$$

其中，$O^2(\Delta t)$ 是二阶项，因为其值是高阶无穷小，所以忽略不计。应用链式法则，就可以得到光流方程如式 4.8

$$I_x u+I_y v+I_t=0 \tag{4.8}$$

其中，I_x 和 I_y 是空间梯度 ΔI 的分量，I_t 就是指对时间的微分的分量。u，v 是图像在运动场的位移的偏移量，光流就是每一个像素点在运动场的速度分量。

光流法可以检测出运动物体比较完整的运动信息，而且不用预先掌握背景的任何信息，可以用于当摄像机运动时对运动目标的检测。但是，光流法对环境依赖性比较高，类似于阴影、多光源、遮挡等原因，使得光流法在应用时计算量非常大。而在现有的硬件条件下，无法保证运动目标检测的实时性，因此很难应用到此系统中。

OpenCV 的实现

void calcOpticalFlowFarneback(InputArray prev, InputArray next, InputOutputArray flow, double pyr_scale, int levels, int winsize, int iterations, int poly_n, double poly_sigma, int flags)

其实，关于光流法，OpenCV 提供了许多函数，但是，相比较之下，calcOpticalFlowFarneback 函数所用这个算法运行速度还算比较快，较适合我们这个比赛，但是，正如图 4.10 所示，由于摄像机噪声比较大，因此，引起的光流变化比较多，因而覆盖了羽毛球引起的光流变化。

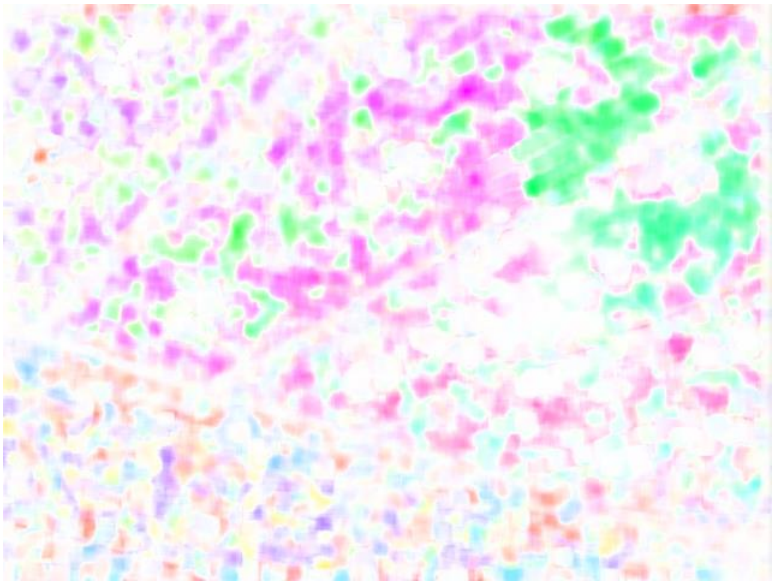


图 4.10 calcOpticalFlowFarneback 作用的结果

利用 OpenCV 实现的代码：

```
calcOpticalFlowFarneback(prevgray, gray, flow, 0.5, 3, 11, 3, 5, 1.1, 0);
motionToColor(flow, motion2color);
imshow("flow", motion2color);
```

4.2.2 背景差分法

背景差分法[20]常用于背景静止时对目标的检测。主要原理就是用当前帧的图像与事先得知的背景图像进行差分运算，然后经过简单的处理就可以很容易的得到运动目标。这个方法很适合应用于背景无变化或者变化不大的场景中。

背景差分法速度快，因此可以满足系统实时性的要求，但是，很多实际应用的背景并不是一直不变的，因此，如果应用这个方法，产生的噪声比较多，处理过程比较麻烦。为此，解决此问题的最好方法就是对背景图像进行建模，通过背景模型的自主学习与自动更新来适应背景的变化。很多学者提出了不同的背景模型，Stauffer 和 Grimson 根据自适应混合高斯模型将图像中的像素点进行建模，这样就可以很好地抑制了因背景、光照等变化产生的噪声影响。由于混合高斯模型能够很好额对复杂背景环境进行描述，且对变化着的环境有很好地自适应能力，，因此，在目标检测中得到了广泛的关注。但是，这个方法要对每一个像素点建立多个高斯分布函数，算法复杂且计算非常巨大，不太适合对实时性要求较高的系统。

4.2.3 帧间差分法

针对不同时刻、同一背景下的多幅图像，对这多幅图像进行比较，就能反应一个运动物体的运动结果。比较直接的方法就是对图像进行差分[21]运算即比较，差分后，灰度不变的部分就被去除掉，剩下的就是运动物体留下的“痕迹”。

帧间差分法的公式如式 4.9.

$$g(x,y)=\begin{cases}1, \left| f_k(x,y)-f_{k-1}(x,y)\right| \geq T_0 \\ 0, \left| f_k(x,y)-f_{k-1}(x,y)\right| \prec T_0\end{cases} \tag{4.9}$$

其中T_0是设定的阈值。

从式 4.2.5 中可以看出，对于邻近两帧来说，由于背景变化不大，而运动目标变化较大，那么，就可以通过比较得出目标物体的轮廓，而帧差法只作用在前后两帧图像上，所以与背景变化无关。

由于羽毛球场地背景变化比较复杂，看台上的人头攒动，空中无人机的盘旋等背景变化等等都加剧了背景的复杂性，而且，对图像处理的实时性要求比较高，从对方击球到我方击球，时间不到 900ms。同时，羽毛球飞行速度比较快，帧间差法可以有效的过滤掉运动目标的重叠现象。所以，这里的系统采用帧间差法。

4.2.4 运动检测的 OpenCV 实现

试验结果分析：如图 4.11 所示，这是一幅羽毛球飞行的原图像，图 4.12 是经过帧差法之后的图像，可以很明显的看出，帧差法可以有效地提取出羽毛球区域而滤除掉背景。达到快速提取的目的。

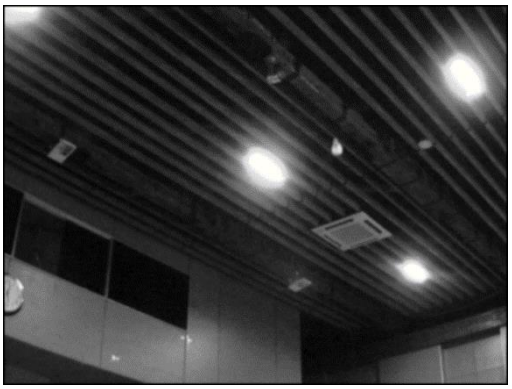


图 4.11 羽毛球飞行图像

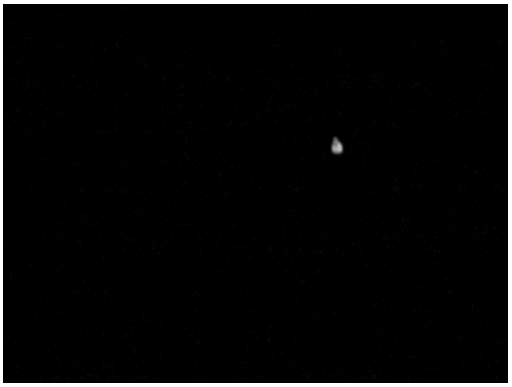


图 4.12 帧差之后的图像

系统中所有的 OpenCV 帧差法的核心代码：

```
capture>>img;
cvtColor(img,img_next,CV_BGR2GRAY);
GaussianBlur(img_next,img_next,Size(3,3),1);
subtract(img_next,img_pre,img_pre);
```

4.3 阈值化处理

4.3.1 概述

当完成很多处理过程之后，通常我们希望想要观察的物体能够从图像中提取出来，包括对像素点做一些决策或者直接剔除高于或者低于目标像素值得像素点，因此，给定一个数组（即图像）和一个高低阈值[22]，然后判断数组中每个元素的值是否在阈值范围之内。这个过程，在图像处理的领域里，就叫做阈值化过程。阈值化技术主要用于图像的分割，将前景目标与背景分割开来，方便后面对目标物体的分析。

4.3.2 阈值化方法

（1）基于聚类的阈值化方法

这类方法中，将具有相同特征或者相似特征的区域分类划分，用多个高斯混合模型表示多个目标物体和背景。在聚类的方法中，Otsu 方式是阈值化技术中，表现比较好的算法。

Otus 方法是基于统计的理论去寻找最佳阈值。设 N 是像素点个数的综合，令 nl 是灰度级为 l 的像素点个数。则 $N=\sum_{l=0}^{L} nl$ 。

图像直方图分布概率的计算公式如式 4.10。

$$Pl=\frac{nl}{N},\sum_{l=0}^{L} pl=1 \tag{4.10}$$

而最佳阈值 T 的计算公式如式 4.11

$$T=\arg\max_{0\le l\le L}\left\{\sigma_B^2(l)\right\} \tag{4.11}$$

其中， σ_B^2 为类间方差。

Otsu 方法总是可以给出满意的阈值，尤其是当类间点数很接近的时候。因此，Otsu 方法也被公认为是最有用的全局阈值化方法。

（2）基于直方图的阈值化方法

这种方法的原理就是利用图像的直方图。通过得出的直方图对图像的整体特征进行判断归类，进而进行分割。比如图像二值化过程，就是在直方图上寻找两个波峰，通过波峰波谷的关系，对图像进行自适应的分割处理。一些作者对此提出了改进办法。其中一种是利用直方图的凹凸度来寻找最佳阈值。具体的过程就是通过函数的计算，设凸函数为 f_t，而概率密度函数是 p_t，计算两者的差的绝对值。当计算 p_t 凸度的时候，最深的那个凸点可以看做是候选的阈值。如果设物体的光滑度为 s 的话，那么凹点的选取一般是通过一些目标的属性得到的反馈来确定的。确定最佳阈值的公式如式 4.12 所示。

$$T = \arg\max_{0\le t\le L}\left\{\max\left|p_t - f_t\right|, s\right\} \tag{4.12}$$

（3）基于熵的阈值化的方法

这一类的算法讨论、研究主要是基于图像灰度级的熵的分布，这通常被称为熵函数。最大化一个熵值说明阈值化的过程中获取到了最大的信息传输。Kapur 等提出的用熵函数阈值化的方法中规定了前后景作为两组不同的信息来源。当这两类熵值达到最大值时，就能得到最优的阈值。其中，前景熵与后景熵的定义分别如式 4.13 和式 4.14 所示。

$$H_1(t) = -\sum_{t=0}^{t}\frac{p_t}{p_1(t)}\ln\frac{p_t}{p_1(t)} \tag{4.13}$$

$$H_2(t) = -\sum_{t=t+1}^{L}\frac{p_t}{p_2(t)}\ln\frac{p_t}{p_2(t)} \tag{4.14}$$

因此，确定的最佳阈值的计算式如式 4.15 所示。

$$T = \arg\max_{0\le t\le L}\left\{H_1(t) + H_2(t)\right\} \tag{4.15}$$

（4）基于空间的阈值化方法

这个方法主要是利用物体与背景像素之间的空间信息。比如关联函数、概率、上下文信息等信息。

Pal 等人提出的方法是利用两个图像拥有同样的直方图，但是却得到了不同的 n 阶熵。作者考虑到哪些具有相同的灰度值的图像发生的概率，从垂直以及水平两个方向进行统计。Pal 提出两个共生的像素概率，定义式 4.16 和式 4.17 所示。

$$T = \arg\max_{0\le t\le L}\left\{H_a(t)+H_b(t)\right\} \tag{4.16}$$

$$T = \arg\max_{0\le t\le L}\left\{H_c(t)+H_d(t)\right\} \tag{4.17}$$

第一个公式是想得到一个二值图像，它拥有了尽可能多的从前景到后景、后景到前景的过度信息。第二个公式是利用相邻的几个像素是否属于同一类别的概率来进行判断提取的。

（5）局部自适应阈值化方法

使用局部自适应阈值化方法来求阈值，其计算过程涉及到每一个像素的特点。其主要根据范围、表面参数、方差或者一些逻辑参数的组合等信息。

White 等人提出的非线性动态窗口阈值化的方法是利用比较相邻像素的平均灰度值以及比较全局像素的灰度值进行判断的。如果此像素的像素值比平均像素像素值低许多的话，它就被认为是感兴趣像素，否则，就被视为是背景。用 a(I,j)来表示像素的平均灰度值，r 来表示一个纠正值。其计算公式如式 4.18 所示。

$$V(i,j)=\begin{cases}1, if a(i,j)<I(i,j)\\ 0, otherwise\end{cases} \tag{4.18}$$

4.3.3 阈值化 OpenCV 实现

OpenCV 里面有两个函数，一个是全局阈值化函数：

double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)

其中，thresh 是阈值下限，maxval 是对阈值上限，type 是阈值化的方式。

THRESH_BINARY：

$$dst(x,y)=\begin{cases}\max val, if src(x,y)>thresh\\ 0, othrewis\end{cases} \tag{4.19}$$

THRESH_BINARY_INV：

$$dst(x,y)=\begin{cases}0, if src(x,y)>thresh\\ \max val, otherwise\end{cases} \tag{4.20}$$

THRESH_TRUNC：

$$dst(x,y)=\begin{cases}threshold, if src(x,y)>thresh\\ src(x,y), otherwise\end{cases} \tag{4.21}$$

THRESH_TOZERO：

$$dst(x,y)=\begin{cases}src(x,y), if src(x,y)>thresh\\ 0, otherwise\end{cases} \tag{4.22}$$

另一个函数是自适应阈值：

void adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C)

其中，adaptiveMethod 是自适应阈值方法，这里提供了两个方法，一个是 CV_ADAPTIVE_THRESH_MEAN_C，这个方法的原理是对区域内的所有像素点进行加权平均计算。另一个是 CV_ADAPTIVE_THRESH_GUSSIAN_C，这个方法的原理是对区域内的中心像素周围的像素点利用高斯函数，根据它们离中心点的距离来进行加权计算的。

由于是在体育场内，羽毛球飞行的时候没有物体进行遮光操作，在利用帧差法之后，经观察，我们得到的图片，基本上没有因光线亮暗变化而产生多变的阈值，因此，为了加快系统的运行速度，这里用全局阈值法。

羽毛球系统中所用的阈值化核心代码：

```
cvtColor(frame_temp, gray, CV_BGR2GRAY);
subtract(gray,pre_gray,pre_gray);
morphologyEx(pre_gray,img_contour,MORPH_OPEN,element);
threshold(img_contour,img_contour,10,255,THRESH_BINARY);
```

4.4 轮廓提取

4.4.1 概述

很多时候，如果仅从图像的轮廓[23]来看的话，便可以得出很多大量的目标物体的信息。通常来讲，图像的轮廓中要包含比其他特征包含更多的信息。因此，轮廓提取在很多视觉系统里面尤其是模式识别过程中，被认为是极其重要的过程。

4.4.2 Freeman 链码原理

这个轮廓提取方法最初是由 Freeman[24]在 1961 年提出来的，沿着二维样条的不同的趋势，产生 8 个方位，分别用{0,1,2，…，7}这 8 个数字标志来表示。任意连续的样条图案，如果被微分后，都可以近似地用{0,2，…，7}这 7 个方向来表示。那么，假设从同一个方向出发，对样条进行描述的话，一幅二维图像的轮廓就可以用 Freeman 链码表示。

依照 Freeman 链码的定义，可以先用一个像素点（设为 P）的 8 邻域进行初步的试探识别，如果该像素被判定为边缘像素，则该像素点（设为 Q）便是 P 点的后继边缘点。如图 4.13 所示。

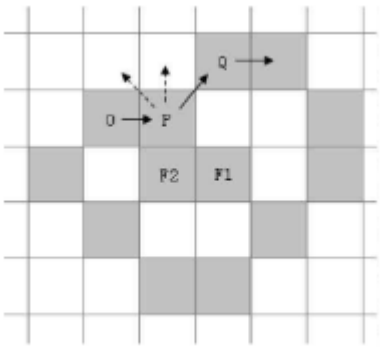


图 4.13 寻找后继点

但是，从图中可以观察出，像素点 P 的 8 邻域像素点 O 与像素点 Q 是此目标的轮廓，但是，像素点 F1 和 F2 是非轮廓点，我们寻找轮廓的关键就是屏蔽掉这两个点。因此，关于这个问题，提出了一种解决方法：

第一步：先要确定跟踪轮廓的方向，顺时针、逆时针都可以。如图 4.14 所示

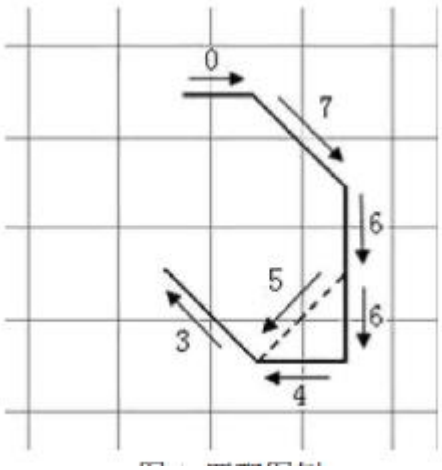


图 4.14 确定轮廓方向

第二步：从此像素点出发，这里选择的是顺时针方向，从 0 到 7 开始跟踪目标轮廓，如果找到的点的像素值不为零，那么就把此点设为轮廓点，并停止对其 8 邻域的其他像素点的搜索。这样，像上面提到的 F1，F2 点就不会被搜索到。

第三步：如果找到目标像素点，那么，就把此点储存起来，然后，重复第二步，对它的后继节点继续进行除已经储存的轮廓像素点除外的 8 邻域判断，找到目标像素点。如果没有感兴趣点可以被搜索，那么停止搜索，已经储存的像素点的集合就是目标物体的轮廓。

4.4.3 轮廓提取的 OpenCV 实现

轮廓提取函数：

void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

其中，contours 是用来存放在 image 中寻找轮廓的点的集合，用 vector<vector<Point>>contours 创建，mode 是轮廓点排列的序列方式。

CV_RETR_EXTERNAL：只检测最外面的轮廓.

CV_RETR_LIST：检测所有的轮廓，并将它们保存在列表中。

CV_RETR_CCOMP：检测出所有的结构并将它们按双层结构老保存。其中，顶层边界是所有成分的最外边界。第二层是孔的边界。

CV_RETR_TREE：检测出所有的轮廓并且重新建立网状轮廓结构.

具体的排列如图 4.15 所示。

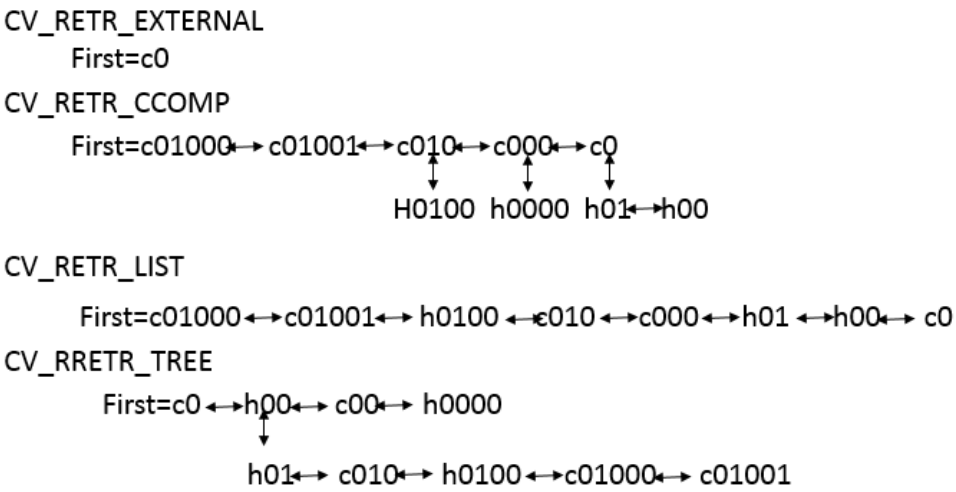


图 4.15 序列排列方式

而 method 是寻找轮廓的方式：

CV_CHAIN_CODE：用 Freeman 链码输出轮廓。

CV_CHAIN_APPROX_NONE：将链码编码中的所有点转换为 Point。

CV_CHAIN_APPROX_SIMPLE：压缩水平、垂直或者倾斜的部分，并且只保存最后一个点。

CV_CHAIN_APPROX_TC89_L1：使用 Tec-Chin 链逼近算法。

由于我们羽毛球是实心的，所以轮廓的排列方式选用的是 CV_RETR_EXTERNAL，这样，对一些孔等噪音等得到一定的滤除。而方法则用了 Freem 链码寻找轮廓的方式，输出则是选用了 CV_CHAIIN_APPROX_NONE，因为后面的特征提取还需要各个点的位置信息，所以，这里要输出每个点的坐标。图 4.16 是用轮廓提取找到的结果。这里用红线画出。

图 4.16 寻找到的羽毛球

系统轮廓提取算法：

findContours(img_contour,contours,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);

drawContours(img,contours,-1,Scalar(0,0,255));

4.5 特征提取

4.5.1 概述

特征是目标物体区别于其他物体的显著性或者标志性特点，图像特征[25]就是指图像中目标物体所拥有的特征。辨别不同目标类别的依据主要是靠特征来判断，能作为图像特征的这些因素都应该具有集中性、可区分性以及重复性等特点，而且，面对亮度、尺度、旋转的影响，应该有一定的适应能力。（基于视觉信息的图像特征提取算法研究），而基于形状特征的检测是对图像目标最好的描述方式，在实际应用中，通常利用形状特征来进行目标特征的提取。

4.5.2 特征提取方法

（1）找圆心

这里用的是最小二乘法拟合圆心。公式推导如式 4.23 所示。

$$R^2 = x^2 - 2Ax + A^2 + y^2 - 2By + B^2 \tag{4.23}$$

令 a=-2A,b=-2B $c = A^2 + B^2 - R^2$ 可得到另一种形式的圆曲线方程，如式 4.24：

$$x^2 + y^2 + ax + by + c = 0 \tag{4.24}$$

那么，只要求出参数 a,b,c 就可以求得圆心半径的参数，如式 4.25。

$$A=-\frac{a}{2},B=-\frac{b}{2},R=\frac{1}{2}\sqrt{a^2+b^2-4c} \tag{4.25}$$

样本集（X_i,Y_i）中点到圆心距离为d_i，如式 4.26，

$$d_i^2=(X_i-A)^2+(Y_i-B)^2 \tag{4.26}$$

通过迭代，可以求出 abc 使得上式的最小值。令

$$C=(N\sum X_i^2-\sum X_i\sum X_i)$$
$$D=(N\sum X_iY_j-\sum X_i\sum Y_j)$$
$$E=(N\sum X_i^3+N\sum X_iY_j^2-\sum(X_i^2+Y_j^2)\sum X_i)$$
$$G=(N\sum Y_i^2-\sum Y_i\sum Y_i)$$
$$H=(N\sum X_i^2Y_j+N\sum Y^3-\sum(X_i^2+Y_j^2)\sum Y_j)$$

可解的：

$$Ca+Db+E=0$$
$$Da+Gb+H=0$$
$$a=\frac{HD-EG}{CG-D^2}$$
$$b=\frac{HC-ED}{D^2-GC}$$
$$c=-\frac{\sum(X_i^2+Y_i^2)+a\sum X_i+b\sum Y_i}{N}$$

OpenCV 实现：

void minEnclosingCircle(InputArray points, Point2f& center, float& radius)

其中，points 是输入的轮廓上的像素点，center 则是求出来的拟合圆的中心，而 radius 则是 求出来的半径。图 4.17 是处理后的效果图。

图 4.17 画出羽毛球的最小拟合圆

通过观察羽毛球的整个飞行过程，以及输出拟合圆的半径可以发现，羽毛球的半径都小于 30 像素点，而大于 30 像素点的，基本上都是一些干扰噪音。因此，对最小拟合圆的半径加以限制，可以有效地去除一些明显的噪声，达到更好的寻找羽毛球的目的。

（2）长宽比例

长宽比例就是通过拟合包围羽毛球的最小矩形框，在有的时候，虽然找到的最小拟合圆满足设置的条件，但是，由于摄像头从我方场地看对方场地，那么避免不了看到对方比赛人员的缓慢移动，经过前后两帧的差分后，很有可能会留下一些细长的区域，但是这篇区域符合拟合圆的条件，因此，这里通过对感兴趣区域进行最小矩形的拟合，判断长宽比例，有效去除上述等原因留下的噪声。

OpenCV 实现：

Rect boundingRect(InputArray points)

其中这个参数是通过轮廓提取找到的那些点的集合。返回的是一个 Rect 结构的值，这个结构能够存储矩形的左上点（x,y 坐标），长（height），宽（width）。

（3）形状匹配

形状匹配能够有效的克服目标物体因旋转、缩放、亮度变化而导致的特征提取错误。形状匹配主要是用到了不变距的概念。所谓不变距，就是利用 n 阶距不变的特性。

不变距[27]： 1962 年美籍华人 Hu 提出了一种方法叫二维距不变量理论，并首次将此方法用到了形状匹配的领域。

对于一个图像中的二维函数 f(x,y)，它的（p+q）阶距的定义可表示成黎曼积分形式如式 4.27 所示。

$$m_{p,q}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}x^{p}y^{q}f(x,y)dxdy \tag{4.27}$$

式中，p，q=0,1,2 …。并根据唯一性定理，若 f(x,y)分段的并且是连续的，那么，只要在平面 xy 上出现非零指 ，那么，该函数就存在任意阶距，并且距的序列$\{m_{p,q}\}$能够唯一的被 f(x,y)所确定。同样的，反过来推导，f(x,y)也能够唯一的确定$\{m_{p,q}\}$。

将上面所提到的特征量进行归一化位置的运算，那么，就能够得到二维连续图像的 f(x,y)的中心距如式 4.28 所示。

$$\mu_{p,q}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(x-x_c)^{p}(y-y_c)^{q}f(x,y)dxdy \tag{4.28}$$

式中，$x_c=\frac{m_{1,0}}{m_{0,0}},y_c=\frac{m_{0,1}}{m_{0,0}}$

$$m_{0,0}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}f(x,y)dxdy,m_{0,1}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}yf(x,y)dxdy,m_{1,0}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}xf(x,y)dxdy$$

而对于实际的图像来说，都是用像素值来表示的离散值，所以，用双重求和的方式来计算离散值。点(x,y)处的(p+q)阶距可以表示如式 4.29

$$m_{p+q}=\sum_{x}\sum_{y}x^{p}y^{q}f(x,y) \tag{4.29}$$

零阶距表示为：$m_{0,0}=\sum_{x}\sum_{y}f(x,y)$ $m_{0,0}$表示的该区域的像素点数，也就是目标物体投影到摄像机的面积。而一阶距则表示为$m_{1,0}=\sum_{x}\sum_{y}xf(x,y)$，

$m_{0,1}=\sum_{x}\sum_{y}yf(x,y)$。同样的，根据连续函数的公式可以看出，二维离散图像的重心坐

标是：$x_c=\frac{m_{1,0}}{m_{0,0}}, y_c=\frac{m_{0,1}}{m_{0,0}}$。二维离散图像(p+q)阶的中心距的定义公式为：

$\mu_{p,q}=\sum_x\sum_y(x-x_x)^p(y-y_c)^q f(x,y)$。

考虑到目标物体的平移运动，设平移量为(a,b)，则平移之后，目标物体的点的坐标就是(x+a,y+b)，平移之后，物体的区域用R'，$m'_{0,0}$和$m_{0,0}$分别用来表示R'和R的面积，那么，通过的推导可以得出平移之后的重心目标如式 4.30.

$$x'_c=\frac{m'_{1,0}}{m'_{0,0}}=\frac{\sum_{(x,y)\in R'}x'}{m'_{0,0}}=\frac{1}{m_{0,0}}\sum_{(x,y)\in R}(x+a)=x_c+a \tag{4.30}$$

同理可得：

$$y'_c=y_c+b \tag{4.31}$$

因此，可以得到中心几何距$\mu'_{p,q}$在变换后的公式如 4.32.

$$\mu'_{p,q}=\sum_{x'}\sum_{y'}(x'-x'_c)^p(y'-y'_c)^q f(x',y')=\mu_{p,q} \tag{4.32}$$

从式 4.5.6 可以看出，中心几何距具有平移不变的特性。

当目标物体的尺寸被放缩 K 倍的时候，那么，像素坐标变成：$x'=kx, y'=ky$，因此，新的重心坐标为：$x'_c=\frac{m'_{1,0}}{m'_{0,0}}=\frac{1}{k^2m_{0,0}}\sum_{(x',y')\in R'}x'=\frac{1}{k^2m_{0,0}}\sum_{(x+y)\in R}kx=kx_c$，同样，也可以推导出$y'_c=ky_c$。上式中，由于$(x',y')\in R'$变成$(x,y)\in R$，所以 R 的面积也改变了$k^2$倍，因此，(p+q)阶中心几何矩就可以这样推导：

$$\mu'_{p,q}=\sum_{(x,y)\in R}k^{p+q+2}(x-x_c)^p(y-y_c)^q f(x,y)=k^{p+q+2}\mu_{p,q} \tag{4.33}$$

使用归一化中心几何矩$\mu_{p,q}=\mu_{p,q}\Big/\mu_{0,0}^r$使得$\mu'_{p,q}=\mu_{p,q}$，其中，$r=(p+q+2)\Big/2$。综上所述，中心几何矩也符合尺度不变的特性

而形状匹配正是基于不变距的基础上发展的。 通过计算匹配函数，可以得出最接近模板图像的图像。从而能得到理想的目标物体-羽毛球。

4.5.3 特征提取的 OpenCV 系统的实现

```
findContours(img_contour,contours,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE
);
        pre_gray=gray.clone();
        itc = contours.begin();
        count_contour = 0;
        while(itc != contours.end())
        {
            minEnclosingCircle(Mat(*itc),center,radius);
            area = contourArea(Mat(*itc));
            arc=arcLength(Mat(*itc),1);
            contour_rect = boundingRect(Mat(*itc));
            contour_scale = (double)contour_rect.width/contour_rect.width;
            if(contour_scale>1.4||contour_scale<1 || area>400 || area<50 || radius>20 )
                itc = contours.erase(itc);
            else
            {
                distance[count_contour] =
matchShapes(contours1[0],*itc,CV_CONTOURS_MATCH_I2,0);
                if(distance[count_contour]>1.2){itc = contours.erase(itc);}
                else
                {
                    itc++;
                    count_contour++;
                }
            }
```

```
}
if(contours.size()>1)
    for(int i=0;i<count_contour;i++)
    {
        if(distance[i]<min_distance)
        {
            min_distance=distance[i];
            min_index=i;
        }
    }
if(contours.size()>1)
    {
        itc = contours.begin();
        while(itc != contours.end())
        {
            if(index_temp!=min_index || min_distance>0.7 )
                itc = contours.erase(itc);
            else {itc++;index_temp++;}
        }
        }
```

4.6 本章小结

本章讲述了关于羽毛球图像处理以及检测的内容。这部分是系统的核心部分，按照先后顺序，具体介绍了形态学操作、运动目标检测、阈值化操作、轮廓提取以及特征判断五个部分，并展示了部分核心代码以及作用的效果图。

5 羽毛球轨迹模型建立与落点预测

5.1 模型的建立

5.1.1 羽毛球受力分析

（1）重力：羽毛球的飞行高度基本上是从 0 米到 6 米的范围内变化，所以，羽毛球质心受到的重力基本上没有大的变化，这里把重力看做常量 g=9.8m/s。因此，羽毛球的受到的重力公式如式 5.1。

$$G = mg \tag{5.1}$$

其中，m 为羽毛球的质量，这里取 m=50g。受力分析如图 5.1。

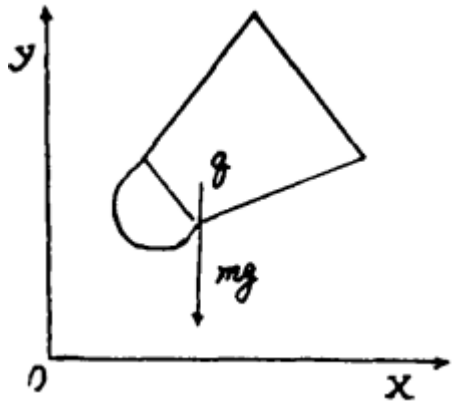


图 5.1 羽毛球受重力示意图

（2）空气阻力[27]：空气阻力可分为迎面阻力，摩擦阻力，涡流阻力三个部分。迎面阻力是指羽毛球在飞行过程中正面所受到的空气压力。它的方向始终与羽毛球飞行的方向相反，如图 5.2 所示。摩擦阻力是指在羽毛球飞行过程中，由于空气属于流体，具有粘滞性，因此，空气将在羽毛球的表面附着一部分空气，使得运动较快的一层空气带动运动较慢的一层空气，这一过程产生的动能是由羽毛球分离出来的，因此羽毛球的动能减少，速度降低，如图 5.3 所示。涡流阻力是指由于羽毛球的前小后大的特征，使得羽毛球前部的流线比羽毛球后部的密，因此，后部产生的压力大于前部，同时使得气流从前到后逐渐减少，甚至气流的速度反向，而前部的气流流速仍未改变，所以这种方向相反的逆流互撞产生一种涡流。如图 5.4 所示。

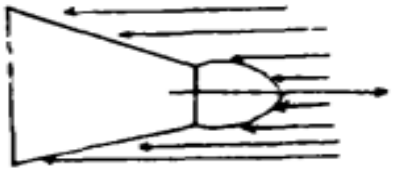


图 5.2 羽毛球受到迎面阻力

图 5.3 羽毛球受到摩擦阻力

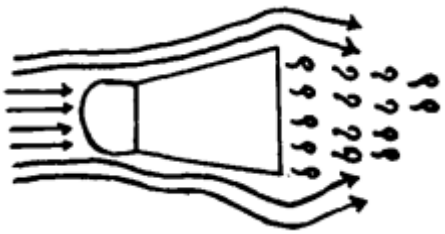


图 5.4 羽毛球受到涡流阻力

（3）马格努斯力[28]：在羽毛球的飞行过程中，因为羽毛球自身发生旋转，所以，根据流体力学的知识，可以知道，羽毛球还受到马格努斯力。马格努斯力的方向遵循右手螺旋法则，中指所指的方向即为马格努斯力的方向，因此，马格努斯力的方向总是与羽毛球飞行的方向垂直，从而导致羽毛球在飞行的过程中，会偏离原本的轨迹方向。根据茹科夫斯基环流理论，可以推导出马格努斯力的公式，如式 5.2 所示：

$$E = -\rho v w h$$

其中，p 为空气密度，v 为羽毛球的飞行速度，w 为羽毛球的旋转角速度，h 为羽毛球的体积。由于本次比赛是机器人之间的对打，其飞行速度，以及旋转角速度较正常比赛要低，经计算，产生的马格努斯力较重力以及空气阻力小几个数量级，所以这里马格努斯力忽略不计。

经过上述力学分析，此系统采用的空气动力学方程式是：

$$F = \frac{1}{2} K_f V^2 \qquad (5.2)$$

其中，F 是羽毛球受到的阻力，Kf 是指阻力系数，V 是指羽毛球飞行的速度。

5.1.2 确定阻力系数

通过查找资料，发现论文关于阻力系数主要分为三种 k1=0.008，k2=0.01146，k3=0.0122。这里，通过做测量此系统下羽毛球实际落点与预测的误差，来选取最合适的阻力系数。其实验结果如表 5.1、表 5.2 及表 5.3 所示。

表 5.1 kf=0.0122 下预测落点与实际落点

3600	2750	3430	3082	170	-332
3600	2750	3271	3104	329	-354
3600	2750	3359	2987	241	-237
3600	2750	3304	3022	296	-272
3600	2750	3236	3084	364	-334
3600	2750	3382	3076	218	-326
3600	2750	3268	3170	332	-420
3600	2750	3423	3045	177	-295
3600	2750	3393	3027	207	-277
3600	2750	3384	3093	216	-343
3600	2750	3243	3062	357	-312
3600	2750	3361	3058	239	-308
3600	2750	3268	3052	332	-302
3600	2750	3373	3019	227	-269
3600	2750	3296	3128	304	-378
3600	2750	3308	3157	292	-407
3600	2750	3276	3141	324	-391
3600	2750	3372	3119	228	-369

表 5.2 kf=0.01146 下预测落点与实际落点

实际落点		预测落点		误差	
3600	2750	3683	2716	-83	34
3600	2750	3642	2763	-42	-13
3600	2750	3652	2817	-52	-67
3600	2750	3587	2822	13	-72
3600	2750	3546	2694	54	56
3600	2750	3634	2782	-34	-32
3600	2750	3687	2699	-87	51
3600	2750	3614	2806	-14	-56
3600	2750	3596	2821	4	-71
3600	2750	3608	2689	-8	61
3600	2750	3692	2748	-92	2
3600	2750	3667	2826	-67	-76
3600	2750	3557	2817	43	-67
3600	2750	3676	2765	-76	-15
3600	2750	3612	2795	-12	-45
3600	2750	3692	2769	-92	-19
3600	2750	3673	2697	-73	53
3600	2750	3658	2727	-58	23
3600	2750	3562	2782	38	-32

表 5.3 kf=0.008 下预测落点与实际落点

实际落点		预测落点		误差	
3600	2750	4156	2230	-556	520
3600	2750	4132	2341	-532	409
3600	2750	4087	2304	-487	446
3600	2750	4057	2311	-457	439
3600	2750	4068	2207	-468	543
3600	2750	4123	2279	-523	471
3600	2750	4151	2331	-551	419
3600	2750	4132	2194	-532	556
3600	2750	4082	2301	-482	449
3600	2750	4193	2299	-593	451
3600	2750	4029	2248	-429	502
3600	2750	4055	2257	-455	493
3600	2750	4134	2267	-534	483
3600	2750	4165	2331	-565	419
3600	2750	4103	2281	-503	469
3600	2750	4094	2328	-494	422
3600	2750	4113	2199	-513	551
3600	2750	4059	2331	-459	419
3600	2750	4117	2315	-517	435

通过比较发现，当阻力系数为 0.0122 的时候，实际落点比预测落点远了 300mm 左右。由此可以知道，阻力系数过大，从而使得计算的羽毛球飞行的速度下降比较快，所

以飞行的距离较近，因此速度下降较快。当阻力系数为 0.008 的时候，实际落点比预测落点近了将近 500mm 左右，由此可以知道，阻力系数过小，从而不能使羽毛球按照实际的速度下降规律下降，因此速度下降较慢。当阻力系数为 0.01146 的时候，预测落点在实际落点的周围，因此系统的阻力系数选取 0.01146 最为合适，误差在直径为 200mm 之间。

5.2 落点预测

5.2.1 模型求解

因为羽毛球飞行涉及到三个坐标系的问题，受到的阻力与飞行的方向有关，所以，直接求解羽毛球的模型涉及到矢量的计算，对于积分运算的话，过程有些复杂，但是，根据微分的思想，当把整个过程分割成无限小的路程时，那么，通过无限次的迭代计算，就能无限逼近于整个过程。因此，这里所用的模型是将羽毛球飞行轨迹按照 2ms 的时间间隔，分成多个匀加速过程。运动方程为：

$$\begin{bmatrix} x_j \\ y_j \\ z_j \end{bmatrix} = \begin{bmatrix} x_{j-1} \\ y_{j-1} \\ z_{j-1} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{5.3}$$

其中，

$$\Delta = v_0 t + \frac{1}{2} a t^2 \tag{5.4}$$

通过式 5.3，式 5.4 可以知道，对于这个模型的求解，初速度的求取是至关重要的，初速的的准确与否直接关系到预测落点的准确性。通过查阅资料，比较可行的方法就是求两点的平均速度，作为两点中点的速度。计算机的处理速度为 15ms 左右，因此，两点之间只经历了 15ms，所以，运动距离不超过 100mm，那么，在 100mm 的距离内，平均速度理论上可以逼近中点速度。但是，大量实现发现，这种方法的误差在直径为 800mm 的范围内，已经大大超出了机器人的容差范围。实验证明，这种方法求得的初速度虽然能逼近中点速度，可是，要预测 2000mm 以上距离的时候，这一点儿误差被极大地放大，导致最后的落点的不准确。

为了解决这个问题，本系统采用的是基于位置的负反馈的方法，因为羽毛球的位置，在某一时刻是确切知道的，那么当加速度一定的情况下，都那么从一个点运动到另一个点的轨迹就是唯一确定的。因此，利用在其后的点来检验第一个点的初速度，并进行校正，能达到准确求取第一个点的初速度的目的。其反馈的方程为：

$$\vec{e} = \vec{p}_{np} - \vec{p}_n \tag{5.5}$$

$$\begin{bmatrix} v'_{x0} \\ v'_{y0} \\ v'_{z0} \end{bmatrix} = \begin{bmatrix} v_{x0} \\ v_{y0} \\ v_{z0} \end{bmatrix} - k \begin{bmatrix} x_{np} - x_n \\ y_{np} - y_n \\ z_{np} - z_n \end{bmatrix}^T \tag{5.6}$$

式 5.6 是校正方程，利用最初算得的初速度进行轨迹的预测，预测到经过第 n 个点的时刻时，预测的羽毛球的位置 $\vec{p}_{np}$，将 $\vec{p}_{np}$ 与实际的位置 $\vec{p}$ 进行比较，通过一定的加权值 k 将这种差别反馈到初速度上，从而构成新的初速度，再进行对目标点的预测，反复这个过程，直到满足终止条件，即预测点无限逼近目标点，那么，这个初速度就可以被认为是第一个点的初速度。其反馈流程如图 5.5 所示。

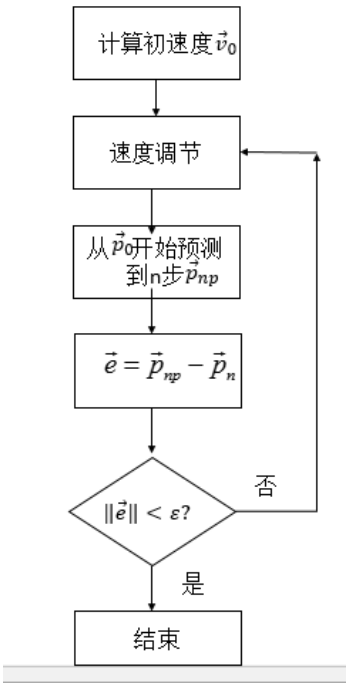


图 5.5 基于位置的速度反馈

试验结果如表 5.2 所示，可以看到，落点基本上在直径为 200mm 的区域范围内，这个范围在机器人的容差范围之内。产生误差的原因一方面是摄像机的像素不够高，不能有效地提取到羽毛球的质心，另一方面，在提取点的过程中，不能保证每次找到的羽毛球上的点都是一样的，这样就造成了对羽毛球位置的错误观察，从而引起误差。但是误差在可接受范围，因此此轨迹预测模型可用。利用 MATLAB 进行仿真，取其中三组数据，如图 5.6、5.7、5.8 所示。其中不同颜色表示在不同的位置点进行预测，得出的不同的预测轨迹。其具体数据见附录 C。

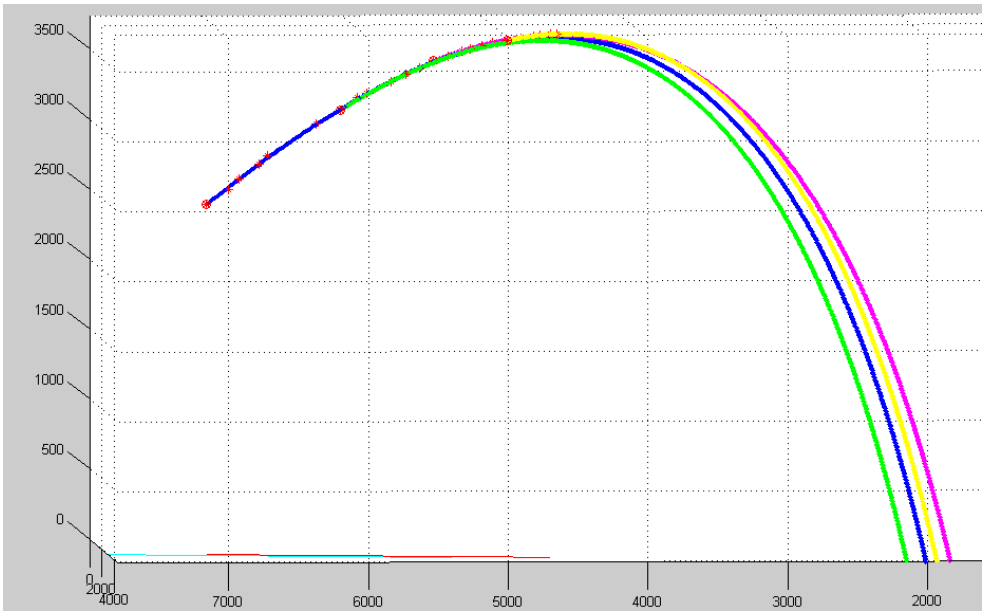


图 5.6 第一组数据

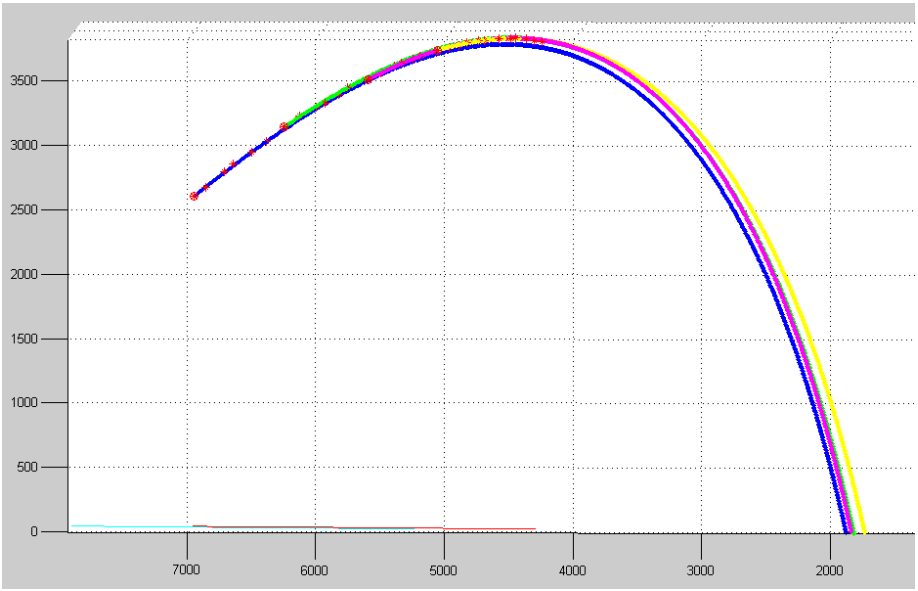


图 5.7 第二组数据

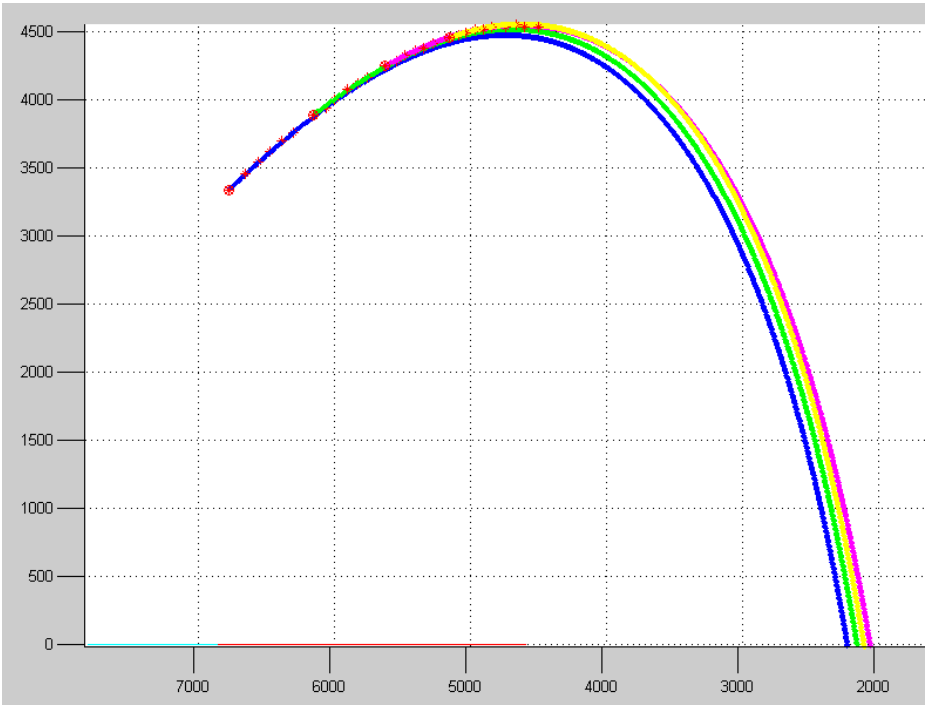


图 5.8 第三组数据

其核心函数程序如下：

void CalPointFinal(double f_p1_x,double f_p1_y,double f_p1_z,double f_p2_x,double f_p2_y,double f_p2_z,double f_t,int f_dst_height,int &f_dstx,int &f_dsty,int &f_dsttime)

```
{

    //这里假设用另外ＲＡＮＳＡＣ已经去除了杂点。
    double f_new_p[2][3];
    f_new_p[0][0]=f_p1_x*0.001;f_new_p[0][1]=f_p1_y*0.001;f_new_p[0][2]=f_p1_z*0.001;
    f_new_p[1][0]=f_p2_x*0.001;f_new_p[1][1]=f_p2_y*0.001;f_new_p[1][2]=f_p2_z*0.001;
    double f_v_x=0.0,f_v_y=0.0,f_v_z=0.0;
    f_v_x = (f_new_p[1][0] - f_new_p[0][0])/f_t;
    f_v_y = (f_new_p[1][1] - f_new_p[0][1])/f_t;
    f_v_z = (f_new_p[1][2] - f_new_p[0][2])/f_t;

    double distance_diff = 0.0;
    double distance_delt = 675;
    double Km = Kf/Mass;

    double f_delt_x = 0.0,f_delt_y = 0.0,f_delt_z = 0.0;
    double f_width = 0.0,f_height=0.0,f_delt_w = 0.0,f_delt_h = 0.0;

    double f_new_v_xy = 0.0;
    double f_new_v_xy2=0.0;
    double f_new_v_z = 0.0;
    double f_new_v = 0.0;
    double f_new_v2 = 0.0;
    double f_angle_x = 0.0,f_angle_y = 0.0,f_angle_z = 0.0,f_angle_xy = 0.0;
    double f_a_all = 0.0,f_a_xy = 0.0,f_a_z = 0.0;
```

```
double time_span = 0.0005;//0.5ms;
double time_span2 = pow(time_span,2)*0.5;
int f_time_count = 0;

double f_predict_x=0.0,f_predict_y=0.0,f_predict_z=0.0;

int flag_stop =0;
int count_iterate    = (int)(f_t/(time_span*1000));

while(1)
{
    ++flag_stop;//停止标志
    f_new_v_xy2 = pow(f_v_x,2) + pow(f_v_y,2);
    f_new_v2 = pow(f_v_z,2) + f_new_v_xy2;
    f_new_v_xy = sqrt(f_new_v_xy2);
    f_new_v = sqrt(f_new_v2);
    f_new_v_z = f_v_z;
    f_angle_x = f_v_x / f_new_v_xy;//计算角度
    f_angle_y = f_v_y / f_new_v_xy;
    f_angle_xy = f_new_v_xy / f_new_v;
    f_angle_z = f_new_v_z / f_new_v;

    f_width = 0.0;f_height = 0.0;//初始化
    cout<<f_new_v<<"    "<<f_new_v_xy<<"     "<<f_new_v_z<<endl;

    for(int i=0;i<count_iterate;i++)
```

```
{
    f_a_all = -Km * f_new_v2;//这里将单位转化一下  1m/s*s = 1000mm/ms*ms .
    f_a_xy = f_a_all * f_angle_xy;
    f_a_z = f_a_all * f_angle_z - 9.8;
    f_delt_h = f_new_v_z * time_span + f_a_z * time_span2;
    f_delt_w = f_new_v_xy * time_span + f_a_xy * time_span2;

    f_width = f_width + f_delt_w;//计算运动距离。
    f_height = f_height + f_delt_h;

    f_new_v_xy = f_new_v_xy + f_a_xy*time_span;//更新速度
    f_new_v_z = f_new_v_z + f_a_z*time_span;
    f_new_v_xy2 = pow(f_new_v_xy,2);
    f_new_v2 = pow(f_new_v_z,2) + f_new_v_xy2;
    f_new_v_xy = sqrt(f_new_v_xy2);
    f_new_v = sqrt(f_new_v2);

    f_angle_xy = f_new_v_xy / f_new_v;
    f_angle_z = f_new_v_z / f_new_v;

}

f_predict_x = f_new_p[0][0] + f_width * f_angle_x;
f_predict_y = f_new_p[0][1] + f_width * f_angle_y;
f_predict_z = f_new_p[0][2] + f_height;
```

```
f_delt_x = f_new_p[1][0] - f_predict_x;
f_delt_y = f_new_p[1][1] - f_predict_y;
f_delt_z = f_new_p[1][2] - f_predict_z;

distance_diff = pow(f_delt_x,2) + pow(f_delt_y,2) + pow(f_delt_z,2);


if(distance_diff < distance_delt )//如果满足条件，那么就开始预测。
{
    cout<<"distance:"<<distance_diff<<endl;

    f_new_v_xy2 = pow(f_v_x,2) + pow(f_v_y,2);
    f_new_v2 = pow(f_v_z,2) + f_new_v_xy2;
    f_new_v = sqrt(f_new_v2);
    f_new_v_xy = sqrt(f_new_v_xy2);
    f_new_v_z = f_v_z;
    f_angle_x = f_v_x / f_new_v_xy;//计算角度
    f_angle_y = f_v_y / f_new_v_xy;
    f_angle_xy = f_new_v_xy / f_new_v;
    f_angle_z = f_new_v_z / f_new_v;

    f_width = 0.0;f_height = f_new_p[0][2];

    while(1)
    {
```

```
++f_time_count;

f_a_all = -Km * f_new_v2 * 1000;//这里将单位转化一下  1m/s*s =
1000mm/ms*ms .
f_a_xy = f_a_all * f_angle_xy;
f_a_z = f_a_all * f_angle_z - 9.8;

f_delt_h = f_new_v_z * time_span + f_a_z * time_span2;
f_delt_w = f_new_v_xy * time_span + f_a_xy * time_span2;

f_width = f_width + f_delt_w;//计算运动距离。
f_height = f_height + f_delt_h;
if( f_new_v_z<0 && f_height < f_dst_height)
{
    f_dsttime = (int)(f_time_count * time_span*1000);
    f_dstx = (int)((f_new_p[0][0] + f_width*f_angle_x)*1000);
    f_dsty = (int)((f_new_p[0][1] + f_width*f_angle_y)*1000);
    break;
}

f_new_v_xy = f_new_v_xy + f_a_xy*time_span;//更新速度
f_new_v_z = f_new_v_z + f_a_z*time_span;
f_new_v_xy2 = pow(f_new_v_xy,2);
f_new_v2 = pow(f_new_v_z,2) + f_new_v_xy2;
f_new_v_xy = sqrt(f_new_v_xy2);
f_new_v = sqrt(f_new_v2);
```

```
                f_angle_xy = f_new_v_xy / f_new_v;
                f_angle_z = f_new_v_z / f_new_v;
            }

            break;

        }
        else
        {
            if(flag_stop<50)
            {
                f_v_x = f_v_x + correct_k * f_delt_x;//修正初速度
                f_v_y = f_v_y + correct_k * f_delt_y;
                f_v_z = f_v_z + correct_k * f_delt_z;
            }
            else
            {
                cout<<"sha bi!!!"<<endl;
                break;
            }
        }
    }
}
```

5.2.2 去除噪点

由于摄像机的像素为 30 万，羽毛球在图像中显示的基本上是一团有着近似三角形的轮廓，特征不明显，无法百分之百的提取出羽毛球，为了应付复杂背景下对落点的准确估计，在取点的时候，需要加上对落点的判断。

此系统去除噪点的算法基于稳健回归（M 估计）方法的基础上进行的。M 估计[29]（Maximum Likelihood Type Estimates）最早是由 Huber 在 1964 年提出来的，同时也可以叫做广义最大似然估计。M 估计基本的思想就是基于迭代加权的最小二乘法[30]来估计系统的回归系数，根据各个回归残差的值来确定每一个的权重从而达到稳健的结果。为了减少那些“不可能点”，该算法对残差值小的给予大的权重，给残差值大的给予晓得权重，然后，每个点根据不同的权重，进行最小二乘拟合，反复跌打以改进各个点的权重系数，直到权重系数的改变值小于允许的误差范围。

OpenCV 里有 M 估计拟合直线的算法实现。

void fitLine(InputArray points, OutputArray line, int distType, double param, double reps, double aeps)

其中，第一个参数是用于拟合所需要的点，第二个参数是用于存放直线的信息。Line[0],line[1]存放的是 x,y 的与斜边的比值，line[2],line[3]存放的是在拟合直线上的一个点。第三个参数是用于拟合直线所用的方法，其值可以是：CV_LIST_I2,CV_DIST_L1,CV_DIST_L12,CV_DIST_FAIR,CVDIST_WELSCH,CV_DIST_HUBER..

此系统的算法是先利用 M 估计，拟合出一个最佳直线，然后计算各个点到直线的距离，然后剔除掉超过阈值的点，接着利用剩下的点再次进行直线拟合，反复迭代直到所有用于直线拟合的点都满足条件，那么所得到的直线就是最接近羽毛球飞行轨迹在 xy 平面投影的直线。

直线拟合以及去除噪点的核心函数程序：

void pointClearNoise(double *f_x,double *f_y,double *f_z,double *f_t,double &f_t_all,double &f_angle_x,double &f_angle_y)

```
{
    vector<Point2f> f_points;
    Vec4f f_lines;
    float f_k=0.0,f_b=0.0;
    float f_distance = 0.0;
    float f_distance_delt = 500.0;//可容许的误差范围
    bool f_flag_iterate = true;
    int f_point_count = 6;
    vector<Point2f>::const_iterator f_itc;


    for(int i=0;i<f_point_count;i++)//将所有的点放到直线拟合的容器里。
    {
        f_points.push_back(Point2f(f_x[i],f_y[i]));
    }

    while(f_flag_iterate)//当所有的点都满足容许的范围之内时。迭代结束。这些点可以作为使用点。
    {
        f_flag_iterate = false;
        //for(int i=0;i<f_point_count;i++)//将所有的点放到直线拟合的容器里。
        //{
        //    f_points.push_back(Point2f(f_x[i],f_y[i]));
        //}

        f_itc = f_points.begin();
```

```
fitLine(Mat(f_points),f_lines,CV_DIST_L2,0,0.01,0.01);



for(int i=0;i<f_point_count;i++)
{
    if(f_lines[0] < 0.001)
    {
        f_distance = abs(f_lines[2] - f_x[i]);
    }
    else
    {
        f_k = f_lines[1]/f_lines[0];//0 代表的是 cos，1 代表的是 sin
        f_b = f_lines[3] - f_lines[2] * f_k ;
        f_distance = abs(f_k * f_x[i] - f_y[i] + f_b)/sqrt(pow(f_k,2) + 1);//点到直线距
离公式。
    }
    if(f_distance > f_distance_delt)
    {
        f_flag_iterate = true;
    if(i==0)//这里如果学会用 vector 的话，那么就不用再一个一个赋值了。直接
对 vector 操作。
        {
            int j = i;
            for(;j<f_point_count-2;j++)
            {
                f_x[j] = f_x[j+1];
```

```
            f_y[j] = f_y[j+1];
            f_z[j] = f_z[j+1];
            f_t[j] = f_t[j+1];
        }
        f_x[j] = f_x[j+1];
        f_y[j] = f_y[j+1];
        f_z[j] = f_z[j+1];
        //f_t[j] = 0.0;
        //f_x[j+1]=0.0;
        //f_y[j+1]=0.0;
        //f_z[j+1]=0.0;
    }
    else
    {
        int j=i;
        for(;j<f_point_count-2;j++)//如果不等于零，那么，将前一次的时间与这一次的时间相加，赋给前一个时间数组。
        {
            f_x[j] = f_x[j+1];
            f_y[j] = f_y[j+1];
            f_z[j] = f_z[j+1];
            f_t[j-1] = f_t[j-1] + f_t[j];
        }
        f_x[j] = f_x[j+1];
        f_y[j] = f_y[j+1];
        f_z[j] = f_z[j+1];
```

```
                //f_t[j] = 0.0;
                //f_x[j+1]=0.0;
                //f_y[j+1]=0.0;
                //f_z[j+1]=0.0;
            }
            --f_point_count;
            f_itc = f_points.erase(f_itc);
        }
        else
        {
            ++f_itc;
        }
    }
}
f_angle_x = f_lines[0];//j 将拟合的直线角度传出
f_angle_y = f_lines[1];
--f_point_count;
if(f_angle_x<0.001)
{
    f_x[1] = f_x[0];
    f_y[1] = f_y[f_point_count];
}
else
{
    double f_k_vertical = -1/f_k;
```

```
        f_x[1] = (f_y[f_point_count] - f_k_vertical * f_x[f_point_count] - f_b) / (f_k -
f_k_vertical);
        f_y[1] = f_k * f_x[1] + f_b;
    }
    for(int i=0;i<f_point_count;i++)
    {
        f_t_all = f_t_all + f_t[i];
    }
}
```

5.3 本章小结

本章主要介绍了对羽毛球的受力进行了分析，从重力以及空气阻力方面进行讨论，并在此基础上，建立羽毛球的轨迹模型，并根据实际情况，提出了基于位置负反馈的初速度矫正方法以及基于 M 估计的直线拟合与噪点去除，从而使得轨迹预测变得更加准确。

结论

本文是以第十四届亚太大学生机器人大赛为背景，主要研究了双目视觉的三维重建，系统之间的蓝牙通信以及局域网通信，羽毛球的特征提取与识别和羽毛球飞行轨迹的预测。经过 4 个月的学习与研究，其设计与实现的系统具有如下功能：

（1） 双目视觉系统能够将两个摄像机中的两个点通过矩阵变换得到较准确的三维点。

（2） 计算机与机器人之前的通信能够保证不错传数，在绝大多数情况下，不漏数。摄像头之间能够保证对羽毛球飞行过程中进行同时拍摄，以及发送数据的准确性。

（3） 对羽毛球能够进行较准确的识别与提取。

（4） 通过对初速度的矫正，使得系统对羽毛球的落点能够有较准确的预测，其误差控制在机器人的容差范围之内，保证机器人的回击。

但是，由于经费问题，不能重新配置高端摄像头。30 万像素的意味着，即使理论准确性比较高，但是，重建出来的三维坐标精度还是不够，所以在轨迹预测方面，还不能实现单拍击打。另外，像素低意味着特征不明显，因此导致了此系统对一些特别像羽毛球的物体不能进行更好的识别。

参 考 文 献

[1]李云江，机器人概论[M]，北京；机械工业出版社 2011

[2]曹文祥，冯雪梅 工业机器人研究现状及发展[J] 机械制造 2011,49(558),41-43

[3] 石继雨.机器人双目立体视觉技术研究[D] 黑龙江自动化学院 2003

[4] Weng J,Cohen P,HemiouM. Camera calibration with distortion models and accuracy evaluation [J],IEEE Trans.PAMI,1992,14(12):965-980

[5]秦小文，温志芳 基于 OpenCv 的图像处理[J] 电子测试 2011,7(7), 39-41

[6]田鹏辉 视频图像中运动目标检测与跟踪方法研究[D] 长安大学 2013

[7]张耀宗 图像匹配算法的研究与应用[D] 太原理工大学 2010

[8]张晶华 羽毛球空气动力学的若干问题研究[D] 广东工业大学 2014

[9]闫龙 摄像机成像过程仿真技术研究[J] 山东大学学报(工业版) 2011,41(3),67-71

[10]福赛斯 计算机视觉：一种现代方法[M] 水利电力出版社 2004

[11] 舒娜 摄像机标定方法的研究[D] 南京理工大学 2014

[12]张正友 计算机视觉：计算理论与算法基础 科学出版社 1998

[13]白云飞 蓝牙通信模块的设计与实现[J] 工业控制学报 2006,6,90-93

[14]杨菲 基于 ARM 的蓝牙无线通信模块的设计与实现[D] 武汉科技大学 2011

[15]金志刚 计算机网络[M] 西安电子科技大学出版社 2009

[16]董向志 Windows 网路编程案例教程[M] 清华大学出版社 2014

[17] Rafael C. Gonzalez, Digital Image Processing(Second Edition)(M) .电子工业出版社，2007

[18]Robert L. OpenCV2 计算机视觉编程手册[M] 科学出版社 2014

[19]Simon Baker, A Database and Evaluation Methodology for Optical Flow[M] Int J Comput Vis 2011

[20]王秀芬，王汇仁基于背景差分法和显著性图的海底目标检测方法[J] 山东大学学报(工学版) 2011,41(1),12-16

[21] 肖本贤，陆诚.基于帧差法与不变矩特征的运动目标检测[C] .第二十七届中国控制会议论文集，2008，578-581

[22]董立菊 图像阈值化技术综述、分类及评价[J] 2004,16(4),8-11

[23]邹柏贤 图像轮廓提取方法研究[J] 计算机工程与应用 2008,44(25),161-165

[24]姚文君 基于 Freeman 链码二维图像轮廓的提取与匹配[J] 宁波职业技术学院学报 2006,10(5),24-26

[25]刘鹏宇 基于内容的图像特征提取算法的研究[D] 吉林大学 2004

[26]Ruliang Zhang, An Image Matching Evolutionary Algorithm Based on Hu Invariant Moments[J] IEEE 2011,113-117

[27]林传潮 羽毛球空气力学分析[J] 体育科技资料 1980,13,27-33

[28]张晶华 四个坐标系下羽毛球飞行运动学模型与仿真[J] 自动化与信息工程 2013,34(1),6-10

[29]王海娜 线性回归模型的若干稳健估计方法以及应用实例[D] 山东大学 2013

[30]Steven Gillan,Pan Agathoklis Image registration using feature points,Zernike moments and an M-estimator[J] IEEE 2010,434-437

附录 A：外文翻译

原文：

Image Registration Using Feature Points, Zernike Moments and an M-estimator

Steven Gillan
Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC V8W3P6
Email: swgillan@ece.uvic.ca

Pan Agathoklis
Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC V8W3P6
Email: pan@ece.uvic.ca

***Abstract*—This paper presents recent improvements in a feature based image registration technique [23, 24]. This image registration technique is based on finding a set of feature points in the two images and using these feature points for registration. This is done in four steps. In the first, images are filtered with the Mexican hat wavelet to obtain the feature point locations. In the second, the Zernike moments of neighbourhoods around the feature points are calculated and compared in the third step to establish correspondence between feature points in the two images and in the fourth the transformation parameters between images are obtained using an iterative least squares technique to eliminate outliers.**

The proposed method is illustrated with examples of images with partial overlap, differences in scale, various affine distortions and contamination with noise.

I. Introduction

Image registration is the process of estimating the parameters of the geometric transformation model that can be used to map a given (target) image to an original (reference) image. An overview of image registration techniques are found in [4, 21, 25]. In general, image registration techniques fall within two methodologies: area based and feature based.

Area based methods estimate the transformation between two images by analyzing pixel intensities of an image using various properties such mutual information, Fourier transforms etc. Mutual information is a concept from information theory which is the measure of dependence between two images based on the assumption that the gray values are of maximum dependence when the images are correctly aligned. Recent applications of using mutual information for image registration has been presented in [7, 13]. Mutual information is a popular method for medical image registration due to the generality of the algorithm [17]. Other area based methods come from the Fourier transform [20]. While FFT based methods have difficulty with cases where scale changes between the images exist, methods such as the ones presented in [8] and [14] have shown improvements on the amount of scaling that can be achieved while also greatly reducing the computation time of the algorithm.

Feature based methods use the extraction of feature points of an image to provide a means of correlation of areas between the two images which are similar and are then used to obtain the parameters for transformation. The methods based on this technique use image characteristics such as edges, corners, feature points, line segments, contours and curvature of image intensity [1, 2, 10, 12, 22]. A popular method in computer-vision applications is the scale invariant feature transform (SIFT) which is based on the feature point detection in scale-space [15]. This technique has shown to be very robust for difference in scale and affine distortions applied to the images [16].

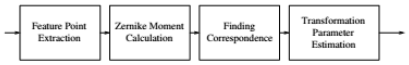


Fig. 1. Summary of the Image Registration Algorithm.

An interesting feature based technique was recently developed which is based on the scale interaction of Mexican-hat wavelets [23, 24]. This Mexican-hat wavelet based feature extraction method was inspired by the method presented in [19] utilizing Gabor wavelets. In [23, 24] feature points extracted from the images are then used to provide a correspondence between an original (reference) image and a second (target) image by comparing the magnitudes of the Zernike moment calculated using neighboorhoods around these feature points [3, 9]. These matching feature points are then used to provide the transformation parameters required to transform the images to a standard form and remove any outlier feature points using an iterative weighted least squares minimization. The properties of this technique make it effective for registering images that have partial overlap, scale differences, affine distortions (rotations, skew, sheer, etc) and image degradations such as noise or blurring. The technique has been so far successfully applied to face recognition [5], aerial image registration [23, 24], and in the future will be considered for content based image retrieval and image mosaicing.

The paper is organized as follows. In section II, the current version of the algorithm is presented and in section III, the performance of the proposed algorithm is demonstrated using various images.

II. Image Registration Method

The image registration algorithm involves four steps as shown in Figure 1. This method was originally presented in

978-1-4244-7773-9/10/$26.00 ©2010 IEEE

[23, 24] and here it has been extended to use a more efficient method for choosing the scale parameters for filtering and to use an M-Estimator for the iterative weighted least squares step. Further, part of the algorithm has been implemented in C to significantly speed up execution. In the rest of this section the steps are briefly described.

A. Feature Point Extraction

Feature points are locations in an image that represent a local maxima of a certain function within a neighborhood in an image. In image registration, these feature points become locations which can be used to determine whether two images have matching points and also to estimate the transformation parameters required to register the two images.

There are many approaches to find feature points. The method presented in this paper uses a two-step process [23, 24]:

1) Filtering an image with the Mexican-hat wavelet.
2) Searching for local maxima of the response.

The response obtained by filtering the same image at different scales using the Mexican hat wavelet will be different. This could result in a failure to correctly extract corresponding feature points from two images which have common areas but are at different scales. In order for a method to work for images at different scales, the feature point extraction step is applied multiple times using a range of values for the Mexican-hat scaling parameter s_{p_i} which will compensate for the difference in scale between the two images.

B. Zernike Moment Calculation

With the feature points for each image found as described in the previous section, the next step is to find matching points in the two images using the Zernike moments of circular neighboorhoods around the feature points. The advantage of using Zernike moments to determine correspondence is two fold. First, the magnitudes of the Zernike moments are rotationally invariant. Second, the Zernike moments of an image, I, is related to the Zernike moments of the scaled image, $\hat{I}$ by the squared scaling parameter s_{p_i}.

Experiments showed that using descriptor vectors for the feature points with the magnitude of Zernike moments up to order 10 provides a reasonable compromise between noise sensitivity and information on fine details in an image.

C. Correspondence

Once the Zernike moments for each feature point are calculated they can be compared to the moments of each feature point on the other image using a correspondence matrix C. Each entry c_{ij} is the $\ell_1 - norm$ of the difference between two descriptor vectors of feature points in the first and second image. A correspondence between two feature points is detected when the minimum value along a row is also the minimum value on the associated column in C. In order to ensure that the true minimum value is found, a threshold between the lowest value, and the next lowest value is used.

D. Transformation parameter estimation

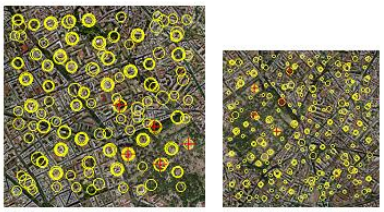
The matching transformations considered are 2D affine transformations such as scaling, rotation, skewing, translation, etc. Based on the correspondence of the feature points between the two images from the previous step, a set of transformation parameters can be found such that points $\mathbf{x}_i = (x_1, x_2)$ of image I can be mapped to the points $\hat{\mathbf{x}}_i = (\hat{x}_1, \hat{x}_2)$ of image $\hat{I}$. These parameters are the solution of an overdetermined set of equations obtained by applying the affine transformation to all pairs of corresponding feature points. This can be solved using an iterative weighted least-square minimization method similar to the ones in robust statistics [6]. A diagonal weighting matrix is iteratively updated to distinguish between inlier and outlier points. The weights are determined at each step using the residual error for each feature point pair and thus small residuals will be heavily weighted while large residuals, which indicate a non-matching pair, will have very little affect on the least square solution, or will be ignored completely. The type of robust estimator method used here is a popular form of *M-Estimation* proposed by Huber [18].

III. Illustrative Examples

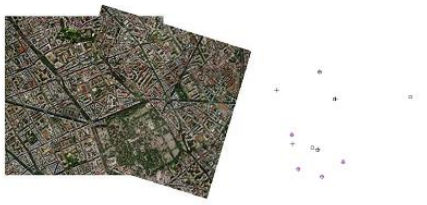
The properties of the registration method described in the previous section will be illustrated here with some examples. These examples include cases where the two images have only partial overlap, affine distortions, differences in scale and noise contamination. The registration process in each case is illustrated with a figure containing four images. The two top images are the ones to be registered with the feature points found overlaid on them. These feature points are indicated with yellow circles of different radius representing different scale factors. The correspondence between them is found using Zernike moment. Some of these feature points are further indicated with red crosses representing inliers detected by the iterative weighted least squares minimization. The final registered image is shown as well as a plot showing the fit of the inlier and outlier feature points after the transformation parameters are applied to the target image and overlaid on the reference image.

Figure 2 shows the registration of two satelite images acquired from Google Earth. The target image, Figure 2b, and reference image Figure 2b are obtained at different scale factors. The feature points found in the two images are shown in Figure 2a and 2b. The use of the iterative weighted least-squares optimization leads to the transformation parameters required to register the two images as shown in Figure 2c. Figure 2d shows the correspondence between the features points and which ones of them are inliers and outlier. It can be seen that there is a large number of outliers which are rejected by the iterative weighted least-squares minimization using the M-estimator.

In the example in Figure 3 the registration algorithm has been used for face recognition [5]. The matching feature point pairs are superimposed on Figures 3a and 3b. The registered image in Figure 3c shows the target image transformed and

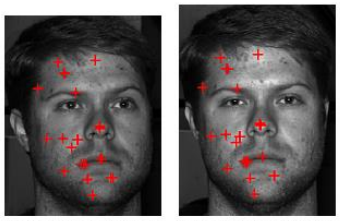


(a) First image with feature points (b) Second image with feature points

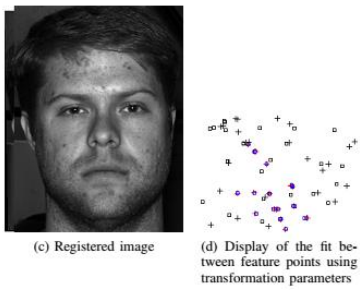


(c) Registered image (d) Display of the fit between feature points using transformation parameters

Fig. 2. Aerial image of Paris, scale differences and affine distortions.

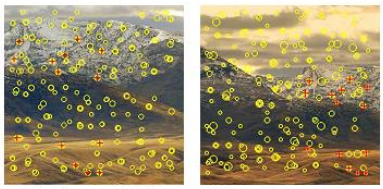


(a) First image with feature points (b) Second image with feature points



(c) Registered image (d) Display of the fit between feature points using transformation parameters

Fig. 3. Face Recognition Application



(a) First image with feature points (b) Second image with feature points

(c) Registered image (d) Display of the fit between feature points using transformation parameters

Fig. 4. Mountain Landscape images which has differences in color and scale

superimposed on the reference image. In Figure 3d the location of the feature points, inliers and outliers are shown.

The third example with two images in significantly different scale and some difference in color is shown in Figure 4 [11]. Performing feature point extraction using a range of different scale parameters results in the correspondence of matching feature point pairs. These points are superimposed on Figure 4a and 4b. After estimating the transformation parameters, the target image is transformed and superimposed on the reference images, shown in Figure 4c and the matching of the feature points is shown in Figure 4d. It can be seen that it works well and since the registration technique is using the luminance it is not affected by difference in the chrominace of the two images.

These brief examples illustrate that this registration technique works well for images with partial overlap and the use of the M-Estimator has the effect of eliminating large number of outliers. Further, the technique can deal with images with different scales and affine distortions as well as degradations like noise contamination etc.

IV. Conclusion

An image registration technique is presented in this paper. This technique is based on scale interactions of Mexican-hat wavelets for feature point extraction, on magnitudes of Zernike moments around circular neighboorhoods of the feature points for finding correspondence between points in the two images and on an iterative weighted least squares minimization algorithm to determine the transformation parameters. This registration technique is briefly illustrated with examples indicating

that this technique works well for images with partial overlap, can deal with images with different scales, affine distortions as well as degradations like noise contamination etc. These properties make this registration technique a good candidate for use in applications such as aerial image registration, face image recognition and image mosaicing.

ACKNOWLEDGMENT

This work has been supported by the National Science and Engineering Council of Canada (NSERC) and the Province of British Columbia through the Ministry of Advanced Education. The authors would like to thank Mohammed Yasien for helpful discussion and an earlier version of the registration algorithm.

REFERENCES

[1] Y. Bentoutou, N. Taleb, A. Bounoua, K. Kpalma, and J. Ronsin, "Feature based registration of satellite images," in *Digital Signal Processing, 2007 15th International Conference on*, July 2007, pp. 419–422.

[2] Y. Bentoutou, N. Taleb, K. Kpalma, and J. Ronsin, "An automatic image registration for applications in remote sensing," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 43, no. 9, pp. 2127–2137, Sept. 2005.

[3] Y. Bin and P. Jia-xiong, "Improvement and invariance analysis of zernike moments using as a region-based shape descriptor," in *Computer Graphics and Image Processing, 2002. Proceedings. XV Brazilian Symposium on*, 2002, pp. 120–127.

[4] L. G. Brown, "A survey of image registration techniques," *ACM Comput. Surv.*, vol. 24, no. 4, pp. 325–376, 1992.

[5] S. Gillan and P. Agathoklis, "A feature based technique for face recognition using mexican hat wavelets," in *Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on*, Aug 2009.

[6] F. R. Hampel, E. M. Ronchetti, P. J. Roussseeuw, and W. A. Strahel, *Robust Statistics: The Approach Based on Influence Functions*. John Wiley & Sons, 1986.

[7] Y. Hu, J. Tang, H. Jiang, and S. Peng, "A fast algorithm to estimate mutual information for image registration," in *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, Nov. 2008, pp. 720–724.

[8] Y. Keller, A. Averbuch, and M. Israeli, "Pseudopolar-based estimation of large translations, rotations, and scalings in images," *Image Processing, IEEE Transactions on*, vol. 14, no. 1, pp. 12–22, Jan. 2005.

[9] H.-K. Kim, J.-D. Kim, D.-G. Sim, and D.-I. Oh, "A modified zernike moment shape descriptor invariant to translation, rotation and scale for similarity-based image retrieval," in *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, vol. 1, 2000, pp. 307–310 vol.1.

[10] T. Kim and Y.-J. Im, "Automatic satellite image registration by combination of matching and random sample consensus," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 41, no. 5, pp. 1111–1117, May 2003.

[11] C. Kuhn. Chuck kuhn photography. [Online]. Available: www.pbase.com/ckuhn55

[12] H. Li, B. Manjunath, and S. Mitra, "A contour-based approach to multisensor image registration," *Image Processing, IEEE Transactions on*, vol. 4, no. 3, pp. 320–334, Mar 1995.

[13] S. Lin, X. Zhu, P. Yan, and J. zhou, "A novel registration method that incorporates template matching and mutual information," in *Automation and Logistics, 2009. ICAL '09. IEEE International Conference on*, Aug. 2009, pp. 813–818.

[14] H. Liu, B. Guo, and Z. Feng, "Pseudo-log-polar fourier transform for image registration," *Signal Processing Letters, IEEE*, vol. 13, no. 1, pp. 17–20, Jan. 2006.

[15] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157.

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[17] F. Maes, D. Vandermeulen, and P. Suetens, "Medical image registration using mutual information," *Proceedings of the IEEE*, vol. 91, no. 10, pp. 1699–1722, Oct. 2003.

[18] E. Malis and E. Marchand, "Experiments with robust estimation techniques in real-time robot vision," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2006, pp. 223–228, beijing, China.

[19] W. A. Manjunath, B. S. Manjunath, and C. Shekhar, "A new approach to image feature detection," *Pattern Recognition*, vol. 29, pp. 627–640, 1996.

[20] B. Reddy and B. Chatterji, "An fft-based technique for translation, rotation, and scale-invariant image registration," *Image Processing, IEEE Transactions on*, vol. 5, no. 8, pp. 1266–1271, Aug 1996.

[21] A. Siu and R. Lau, "Image registration for image-based rendering," *Image Processing, IEEE Transactions on*, vol. 14, no. 2, pp. 241–252, Feb. 2005.

[22] G.-J. Wen, J. jian Lv, and W. xian Yu, "A high-performance feature-matching method for image registration by combining spatial and similarity information," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 46, no. 4, pp. 1266–1277, April 2008.

[23] M. Yasein and P. Agathoklis, "A feature-based image registration technique for images of different scale," in *Proc. IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008.*, May 2008, pp. 3558–3561.

[24] M. Yasien and P. Agathoklis, "A robust, feature-based algorithm for aerial image registration."

[25] B. Zitová and J. Flusser, "Image registration methods: a survey," *Proceedings of Image Vision Computing*, no. 3, pp. 977–1000, 2003.

翻译：利用特征点，zernike 矩和 M 估计的图像配准

摘要：本论文提出了一些较新的基于图像配准技术的特征改进方法。这个图像配准技术是基于寻找两张图片的特征点并利用这些点进行配准的。这个方法可分为四步。第

一，利用墨西哥帽小波函数去获得特征点的位置。然后，计算特征点附近的 zernike 矩，第三步，通过比较建立起两张图片的特征点的对应关系。第四步，利用最小二乘迭代法去除异常值的方法获得图片间转换参数。

本文提出的方法，利用不同尺寸，不同仿射畸变，噪声污染的图片来进行证明。

一、介绍

图像配准是一个估计几何变换模型参数，将目标图片变换到原始图像的过程。一个关于图像配准技术的概论可以在[4,21,25]里查到。一般来讲，图像配准技术不外乎两个理论：基于区域和基于特征的。

基于区域的方法是通过利用互信息、傅里叶变换等来分析像素点强度来估计两张图片的转换关系。互信息是信息论的一个方面，当图片已经被排好序之后，通过估计灰度值的最大相关性来衡量两张图片的相关性。目前利用互信息来进行图像配准的应用在[7,13]。根据论文[17]可看到，互信息在医学图像配准领域很普遍。其他一些领域主要用傅里叶变换。然而，当图片的尺寸变换之后，傅里叶变换将很难应用。在论文[8],[14]里，提出的方法有效解决了这个尺度变换问题，而且大大节省了算法的计算时间。

基于特征的方法主要应用于图片的特征点的提取上，这个方法提供了一个获得相似的两张图片相关性的手段，并被用来获得转换参数。基于这个技术的方法，用到了图片的特征，诸如边缘、角点、特征点、分割线、轮廓和图片深度[1,2,10,12,22]。一个在机器视觉常用的应用是尺度不变的特征转换，这个方法是基于特征点检测。这个技术在不同尺度，不同仿射畸变的图片下展现了很强的鲁棒性。

基于尺度相关的墨西哥帽小波的技术最近开始走红。这个基于特征提取的墨西哥帽小波方法通过 Gabor 小波的提出而被发展起来。从图片中提取的这些特征点表现了原始图片和目标图片的相关性。这个相关性是通过对比计算相邻特征点的 zernike 矩的梯度得出的[3,9]。这些匹配的特征点之后会被用来计算转换参数，这些参数就是用来将图片转换为标准图片，而且，基于这些参数，利用带权重的最小二乘法去除异常点。此技术的这个特性使得对不同尺寸、不同放射畸变、图像腐蚀诸如噪声或者滤波等的图像配

准比较有效。这个技术成功应用于人脸识别、航拍图片配准，而且，将来准备考虑应用于图像搜索和图像拼接领域。

本论文的结构组织是这样的。在第二部分，列出了当前的一些算法。在第三部分，利用不同的图片来证明提出算法的表现。

二、图像匹配准算法

图像配准算法主要包括四个部分，如图 1 所示。这个方法最初在[23,24]中提出，这里提出了一个对于不同比例滤波系数选择的更有效的方法，并利用基于 M 估计的最小二乘法。而后，部分算法已经用 C 语言写好并显著提升执行速度。接下来的部分，将简单描述这些步骤。

A. 特征点提取。

特征点是图片中的一些代表邻域内最大函数值的像素点。在图像配准中，这些特征点被用来判断两幅图片是否拥有匹配点的依据，和估算被用来配准两幅图片的转换参数。这里有很多方法去寻找特征点。这个方法在[23,224]中提出，用到了两步法：

1）利用墨西哥帽小波函数对图片进行滤波。

2）寻找最大函数值的特征点。

这些对同一幅图片利用不同尺度的墨西哥小波处理而获得的代表点是不相同的。这样将导致不能从两幅具有相同区域但是不同尺度的图片中准确提取出相关的特征点。为了找到一种能在具有不同尺度的图片中有效配准的方法，特征点的提取步骤将利用过个时间下一组经墨西哥帽小波得出的参数s_{p_i}，这样就会弥补两张图片在不同尺度下的差异。

B. zernike 矩的计算

利用上个章节描述的算法，对每个图片提取到了特征点，下一步就是在特征点邻域内利用 zernike 矩寻找匹配点。用 zernike 矩去决定图片的相关性的优点有两个方面。第一，zernike 矩具有旋转不变性，第二，一张图片 I 与规定化的图片 I1 具有正比关系。

试验证明，用描述符容器来储存大量的 zernike 矩对噪声影响和图片细节信息有很好地保证。

C. 一致性

一旦对每一个特征点进行 zernike 矩的计算之后，就可以对每一个特征点与另一张图片进行对比得出一致性矩阵 C，每一个输入 Cij 都对应着一个 L1—归一化的两幅图片描述容器里的特征点的差异。在被检测的两个特征的一致性，即使矩阵 C 每一行的最小值，也是每一列的最小值。为了确定真的最小值被找到，在上一个最小值和下一个最小值之间设置一个阈值。

D. 转换参数的估计

这个配准转换矩阵，被认为是二维的仿射转换矩阵，例如尺度变换、旋转、形变、转换等。基于在上一步骤找到的基于特征点的一致性，一系列的转换参数将得到确认。例如图片 I 中的点 $x_i = (x_1, x_2)$ 与 I1 中的点 $\hat{x}_i = (\hat{x}_1, \hat{x}_2)$ 匹配。这些参数为一系列超定方程提供了解，而这些解是通过应用配对的一致性特征点得到的。这个可以用带权重的最小二乘法进行求解[6]。一个对角权重矩阵不断地迭代更新以区分内点与外点。通过每一步对每一对特征点计算的的残差值来决定权值，这样，小的残差拥有大的权值而大的残差拥有小的权值，这就说明，一对不匹配的特征点对二乘结果的影响会非常小，或者直接忽略。这个用到的具有鲁棒性估计的方法是 Huber 提出的 M 估计的一种应用比较广泛的形式。

三、试验证明

在前几节中提到的匹配算法将在这一节用几个粒子来证明。这些例子包含了仅有仿射畸变、不同尺度和噪声污染的情况。在每个例子中的配准过程会用一个包含 4 幅图片的图显示。前两幅图片是在寻找特征点的标准图片，这些特征点用黄色圈来表示，不同的半径代表不同的比例因素。它们之间的一致性使用 zernike 矩来寻找的。一些特征点又用红十字线标出，这代表着用带权重的最小二乘法计算的内点。显示的最后一张配准图同样也用一个表格表明在应用转换参数到目标图片后得出的内点和外点分布。并画在了被匹配的图片中。

图 2 从谷歌地图上得到的卫星图。图 2b 是目标图片与不同尺度下的被匹配的图片。在两张图片中分别找到的特征点显示在图 2a，和图 2b。利用迭代的带权重的最小二乘

法得出的匹配两张图片的配准参数显示在图 2c，图 2d 显示了特征点的一致性关系，它们当中有内点，也有外点。可以看出，利用 m 估计的最小二乘法拒绝了大量的外点。

在图 3 的例子中，配准算法被应用与人脸识别。匹配的特征点对添加在图 3a 和图 3b。图 3c 中的配准图片显示了转换的目标图片与添加的标准图片。在图 3d 中，显示了匹配点，内点，外点。

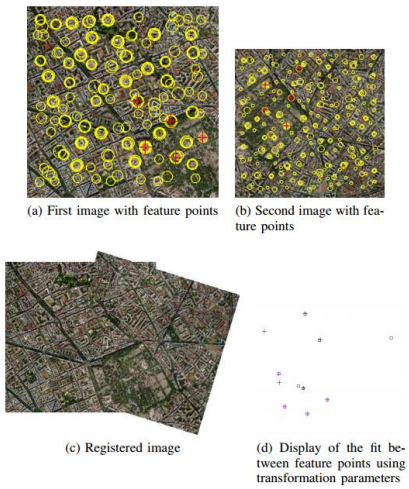


图 1 巴黎的航拍图，不同的尺度，不同的畸变。

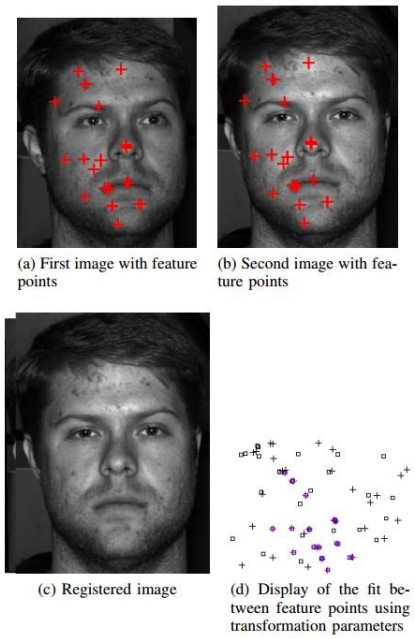


图 2 人脸识别应用

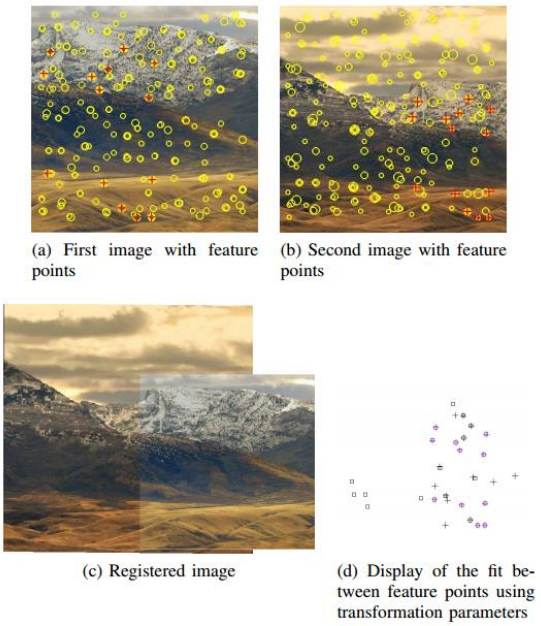


图 4 拥有不同尺度和颜色的山的风景图

在图 4 中的第三个图片，是两幅尺度颜色都不同的风景图，用一系列的不同比例的参数进行特征点的提取，得到了一致性匹配的特征点对。这些点在图 4a 和图 4b 被画出来。在估计转换参数之后，目标图片经过转换添加到标准的图片中，如图 4c 所示，匹配点如图 4d 所示。我们可以看到，这个算法效果很好，因为配准算法应用了亮度信息，所以结果并没有因为色度变化而又影响。

四、结论

本论文提出了一种图像配准的算法。这个技术应用了特征点提取的墨西哥帽小波的迭代操作，在两幅图片的特征点附近计算 zernike 矩，对特征点进行带权重的最小二乘拟合去决定转换参数。这个图像配准技术用几个例子简单的证明了一下，可以看到对不同尺度变换下，不同的仿射畸变下，不同的噪声污染下的效果很好。这个方法使得图像配准技术成为一个可以用于航拍图配准、人脸识别和图像拼接。

附录 B 比赛用机器人

附录 C 羽毛球飞行轨迹以及预测轨迹数据

表 1 第一组数据

实际点x:	y:	z:	第一阶段	x:	y:	z:
3160.931	7097.614	2498.326	速度	2.298047	-8.00843	6.537208
3198.6	6941.748	2608.451	初始点	3.160931	7.097614	2.498326
3220.139	6864.533	2685.79	落点	4620	2009	0
3248.183	6725.305	2789.549	第二阶段	x:	y:	z:
3282.112	6662.421	2849.29	速度	1.617518	-5.77011	3.648374
3384.288	6319.239	3079.211	初始点	3.432394	6.145929	3.185633
3432.394	6145.929	3185.633	落点	4553	2146	0
3468.403	6032.857	3270.787	第三阶段	x:	y:	z:
3492.991	5962.325	3308.232	速度	1.510169	-5.38331	2.121134
3535.687	5789.812	3389.448	初始点	3.623357	5.488691	3.540463
3558.529	5687.294	3441.583	落点	4647	1836	0
3589.482	5585.556	3488.578	第四阶段	x:	y:	z:
3623.357	5488.691	3540.463	速度	1.104577	-4.48474	1.027605
3649.466	5389.222	3576.786	初始点	3.756339	4.967126	3.69018
3680.293	5290.273	3612.915	落点	4504	1927	0
3693.656	5231.29	3626.238				
3711.944	5145.726	3652.775				
3740.741	5070.253	3673.556				
3756.339	4967.126	3690.18				
3796.567	4871.805	3709.659				
3816.064	4785.047	3723.572				
3824.888	4739.451	3731.534				
3848.706	4664.206	3735.238				
3842.607	4616.865	3738.858				

表 2 第二组数据

轨迹x:	y:	z:	第一阶段	x：	y:	z:
2859.358	6976.061	2559.896	速度	1.989916	-7.96317	6.475697
2862.997	6886.494	2634.194	初始点	2.859358	6.976061	2.559896
2903.875	6737.176	2751.225	落点	4133	1877	0
2924.959	6670.368	2818.932	第二阶段	x：	y:	z:
2949.087	6529.935	2905.317	速度	1.529447	-6.53617	4.500721
3000.014	6413.188	2990.139	初始点	3.006855	6.274015	3.104552
3006.855	6274.015	3104.552	落点	4049	1820	0
3042.601	6151.45	3189.426	第三阶段	x：	y:	z:
3072.6	6024.784	3261.928	速度	1.238211	-5.39457	2.847488
3073.706	5950.788	3293.705	初始点	3.144538	5.620482	3.478749
3110.078	5843.395	3360.845	落点	4012	1839	0
3122.34	5780.485	3413.69	第四阶段	x：	y:	z:
3144.538	5620.482	3478.749	速度	1.201489	-4.90674	1.464216
3183.348	5516.395	3530.018	初始点	3.259303	5.078242	3.711383
3208.741	5412.469	3584.718	落点	4078	1734	0
3205.338	5358.235	3610.581				
3239.284	5262.499	3644.181				
3238.939	5209.202	3665.235				
3259.303	5078.242	3711.383				
3291.005	4980.017	3737.262				
3313.272	4909.626	3750.838				
3317.035	4842.786	3769.539				
3337.677	4752.518	3787.882				
3349.979	4707.932	3792.258				
3365.428	4591.609	3803.495				
3375.086	4501.1	3806.263				
3389.282	4461.225	3816.774				
3397.257	4379.634	3804.675				
3428.835	4315.812	3795.604				
3427.91	4257.623	3788.382				

表 3 第三组数据

轨迹x:	y:	z:	第一阶段	x:	y:	z:
2942.019	6766.225	3340.99	速度	1.596286	-6.34094	6.014815
2965.583	6644.217	3457.477	初始点	2.942019	6.766225	3.34099
2996.768	6551.732	3546.336	落点	4084	2228	0
3024.661	6468.138	3624.805	第二阶段	x:	y:	z:
3040.461	6381.736	3699.41	速度	1.308219	-5.33578	4.109328
3061.088	6293.245	3759.421	初始点	3.079791	6.1497	3.887882
3079.791	6149.7	3887.882	落点	4058	2157	0
3114.215	6053.823	3937.711	第三阶段	x:	y:	z:
3124.128	5990.06	3999.643	速度	1.036429	-4.71969	2.677088
3136.663	5892.172	4077.8	初始点	3.202556	5.620825	4.248268
3171.099	5820.629	4125.96	落点	3984	2060	0
3178.436	5747.358	4167.588	第四阶段	x:	y:	z:
3202.556	5620.825	4248.268	速度	0.851464	-4.0641	1.465331
3227.919	5538.928	4285.539	初始点	3.294214	5.147686	4.458208
3233.671	5478.04	4326.335	落点	3931	2104	0
3258.288	5395.237	4367.84				
3254.811	5338.108	4383.431				
3280.166	5267.406	4419.505				
3294.214	5147.686	4458.208				
3319.807	5076.083	4472.211				
3317.742	5028.883	4496.956				
3336.752	4956.803	4515.488				
3342.422	4897.067	4517.638				
3359.739	4834.929	4540.445				
3384.053	4730.527	4536.61				
3396.667	4660.514	4549.615				
3386.067	4595.494	4537.32				

附录 D 羽毛球系统所用部分程序

主机端：

头文件：

```
#ifndef GLOBAL_SERVER_H_
#define GLOBAL_SERVER_H_

//*************************头文件*********************************//
#include <Windows.h>
#include <stdio.h>
#include<cv.h>
#include <math.h>
#include <opencv.hpp>
#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2//opencv.hpp>

#include <CameraApi.h>
#include <CameraDefine.H>
#include "SerialPort.h"

#include <string>//用于输出寻找顶点的图片。
#include <iostream>

#pragma comment(lib,"ws2_32.lib")
#pragma comment(lib,"MVCAMSDK.lib")
#pragma comment(lib, "WS2_32")

    using namespace cv;
using namespace std;

//*************************宏定义*********************************//
#define Kf_x    0.0005
#define Kf_y    0.0001
#define Kf_z    0.0005
```

```
#define Mass      0.05
#define Gravity      9.8
#define K1        sqrt(Mass/(Kf_z*Gravity))
#define K2        sqrt(Kf_z/(Mass*Gravity))
#define K3        sqrt(Kf_z*Gravity/(Mass))
#define DstHeight      0.7//630//800//这里要把目标高度的单位改为m
#define Kf         0.0116//0.0122
//************************全局变量定义*******************************//
int                              g_hCamera = -1;         //设备句柄
unsigned char                   * g_pRgbBuffer=NULL;         //处理后数据缓存区

tSdkFrameHead                    g_tFrameHead;           //图像帧头信息
tSdkCameraCapbility         g_tCapability;          //设备描述信息
BYTE                               *g_readBuf=NULL;       //画板显示数据区


SOCKET slisten;
SOCKET sClient;
sockaddr_in remoteAddr;
int nAddrlen = sizeof(remoteAddr);

Mat frame; bool finish_processing=0;                    //一帧处理结束标志
bool getBall_r=0;                                         //当前帧是否有球
Point2i point2D_r;double pre_point_3D[3],point_3D[3];
bool predict_track=0;                                     //是否预测
bool get_nextFrame=0;                                     //是否获取下一帧
TickMeter tm_twoBall;                                     //两帧有球之间时间获取中间变量
int getBallAll_num=0;                                     //一次打球中有球的帧数
double time_getBallAll=0;                                 //获取两帧有球之间时间
int flag_firstPredict=0;                                  //是否为第一次预测
bool get_image=0;
int dstime=0;

double time_need,time_need_0,destime_pre=0;
double des_x_pre = 0.0,des_y_pre = 0.0;

double array_des_x[30],array_des_y[30],array_des_time[30];//用于对前十个数据做平均用。
int count_x = 0,count_y = 0,count_time = 0;
int flag_send_time_need = 0;//0表示无球，1表示找到球，2表示可发送挥拍时间。
```

```
int last_point_dst_x = 0,last_point_dst_y = 0,last_point_dst_time = 0;
double time_first_current[30];//用来统计从第一次找到球到当前找到球的时间。
double time_first_send = 0;//用来存储第一次找到球到开始给车发送挥拍的时间。
double array_dst_x_30[300],array_dst_y_30[300],array_dst_z_30[300];//用于储存30ms内的羽毛球的状态。
bool send_flag_200 = false;//用于控制是否给车发送数据？
double correct_k = 5;//初速度调节系数。
double time_send_200;//用于统计是否给车发送数据。

int name_main_count = 4;//这个是用来输出寻找顶点的图片的名字。
//************************函数声明*******************************//
extern void tripointer(double uvLx, double uvLy, double uvRx, double uvRy, double *pw);
double time_count(double f_v_z,double f_p_z,double f_dst_height);
int init_SDK();
void GetImage();
void CalPoint(double f_v_x,double f_v_y,double f_v_z,double f_p_x,double f_p_y,double f_p_z,double
f_dst_height,double &f_time_need,double &f_dst_x,double &f_dst_y);
void CalPoint30(double f_p_x,double f_p_y,double f_p_z,double f_predict_x,double f_predict_y,double
f_predict_z,double f_v_x,double f_v_y,double f_v_z,double *f_dst_p_x,double *f_dst_p_y,double *f_dst_p_z);

Point maxPoint(vector<Point> seriers_point,double center_x,double center_y,double pre_center_x,double
pre_center_y);
void CalPointFinal(double *f_x,double *f_y,double *f_z,double *f_t,int f_dst_height,int &f_dstx,int &f_dsty,int
&f_dsttime,bool &f_flag_bit);
//void calLastTime(int &cal_time,double *array_point,double dst_point,double *array_time,double
*array_time_all,double time_all);

#endif
```

主体程序：

```
#include "global_server.h"
double tm_all_work = 0.0;
Point point2D_l;FILE *bb_file;CSerialPort port;

double time_save[5];
double time_get_ball_gap = 0.0;//统计两球之间的时间。
int flag_get_noball    = 0;
TickMeter tm_cal_point;
bool flag_track = false;
double point_3D_x[6],point_3D_y[6],point_3D_z[6];
```

```
double c = (50.0/180.0)*CV_PI;
int dstHeight = (320 + 580 * sin(c))*0.001;
//int dstHeight = 0.764;
//*****************************线程函数*****************************//
DWORD WINAPI Fun(LPVOID lpParamter)
{
    char recData[200];
    while(1)
    {
        while(!recv(sClient,recData,200,0));
        while(!finish_processing);
        if(getBall_r && recData[0]=='9')//全得到球
        {
            point2D_l.x=(recData[1]-48)*100+(recData[2]-48)*10+(recData[3]-48);
            point2D_l.y=(recData[4]-48)*100+(recData[5]-48)*10+(recData[6]-48);
            ++getBallAll_num;
            tm_twoBall.stop();
            time_getBallAll=tm_twoBall.getTimeSec();
            tm_twoBall.reset();
            if(time_getBallAll>0.12 && getBallAll_num>=1)
            {
                getBallAll_num=0;//一次结束
                predict_track=0;
                tm_all_work =0.0;
                des_x_pre = 0.0;
                des_y_pre = 0.0;
                destime_pre = 0.0;
                count_x = 0;
                count_y = 0;
                time_get_ball_gap = 0.0;//找到第一个球，那么将这个清零。
                cout<<"************************************"<<endl;
//*************************************写文件*********************************************//
                //if((bb_file=fopen("data.txt","a"))==NULL)
                //    printf("没打开data.txt");
                //char *buffer="*********************************************\n";
                //fprintf(bb_file,"%s\n",buffer);
                //fclose(bb_file);
            }
```

```
        else if(getBallAll_num>=1)
        {
            tm_twoBall.start();
            if(getBallAll_num==1)
            {
                tripointer(point2D_l.x, point2D_l.y,point2D_r.x, point2D_r.y,point_3D);
                point_3D_x[0] = point_3D[0];
                point_3D_y[0] = point_3D[1];
                point_3D_z[0] = point_3D[2];
                time_get_ball_gap=0.0;
                flag_get_noball=0;
            }
            else if(getBallAll_num>=2)
            {
                tm_all_work = ((recData[7]-48)*100+(recData[8]-48)*10+(recData[9]-48))*0.1;
                //cout<<"tm_all_work:"<<tm_all_work<<endl;
                time_get_ball_gap = time_get_ball_gap + tm_all_work;
                time_save[getBallAll_num-2] = time_get_ball_gap;
                time_get_ball_gap=0.0;
                tripointer(point2D_l.x, point2D_l.y,point2D_r.x, point2D_r.y,point_3D);
                point_3D_x[getBallAll_num-1] = point_3D[0];
                point_3D_y[getBallAll_num-1] = point_3D[1];
                point_3D_z[getBallAll_num-1] = point_3D[2];
                flag_get_noball=0;
                if(getBallAll_num == 6)
                    predict_track=1;
            }
        }
    }
    else
    {
        if(getBall_r || recData[0]=='9' )//两边有一个有球
        {
            if(getBallAll_num>=1)//只要找到第一个球，那么就证明这个过程开始了。
            {
                tm_all_work = ((recData[7]-48)*100+(recData[8]-48)*10+(recData[9]-48))*0.1;
                time_get_ball_gap = time_get_ball_gap + tm_all_work;
            }
            flag_get_noball=0;
```

```
            }
            else//都没球
            {
                if(getBallAll_num>=1)
                    ++flag_get_noball;
                if(flag_get_noball>3)
                {
                    time_get_ball_gap=0.0;
                }
                else
                {
                    tm_all_work = ((recData[7]-48)*100+(recData[8]-48)*10+(recData[9]-48))*0.1;
                    time_get_ball_gap = time_get_ball_gap + tm_all_work;
                }
            }

        }
        get_nextFrame=1;
        flag_track = true;
        while(flag_track);
        char * sendData = "1";
        send(sClient, sendData, strlen(sendData), 0);
        GetImage();
        finish_processing=0;
    }
}


int main(void)
{
    //**************************初始化*****************************//

    init_SDK();
    Mat shape=imread("1.png");
    cvtColor(shape,shape,CV_BGR2GRAY);
    vector<vector<Point>>contours1;
    findContours(shape,contours1,CV_RETR_LIST,CV_CHAIN_APPROX_NONE);
    contours1.erase(contours1.end()-1);
```

```
if(port.InitPort(5,115200))
	cout<<"init port success"<<endl;
else
	return 0;
port.OpenListenThread();

//************************变量定义*****************************//
Mat pre_gray,gray,img_contour,frame_temp;
Mat element(5,5,CV_8U,Scalar(1));
Point2f center;float radius;double area,arc,contour_scale;Rect contour_rect ;
int count_contour=0;double min_distance=1.5;int min_index;double distance[20];int index_temp=0;
vector<vector<Point>>contours;
vector<vector<Point>>::const_iterator itc;
double    v_x,v_y,v_z,p_x,p_y,p_z;double predictTime_all=0;double dst_x,dst_y,dst_x_0,dst_y_0;
int dstx,dsty;

//**********************通信***********************************//
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
if(WSAStartup(sockVersion, &wsaData)!=0)
{
	return 0;
}
//创建套接字
if(slisten == INVALID_SOCKET)
{
	printf("socket error !");
	return 0;
}
slisten= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
//绑定IP和端口
sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(8888);
sin.sin_addr.S_un.S_addr = INADDR_ANY;
if(bind(slisten, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET_ERROR)
{
	printf("bind error !");
}
```

```
//开始监听
if(listen(slisten, 5) == SOCKET_ERROR)
{
    printf("listen error !");
    return 12;
}
printf("等待连接...\n");
sClient = accept(slisten, (SOCKADDR *)&remoteAddr, &nAddrlen);
if(sClient == INVALID_SOCKET)
{
    printf("accept error !");
}
printf("对方ip：%s \r\n", inet_ntoa(remoteAddr.sin_addr));
//******************多线程*********************************//
HANDLE hThread = CreateThread(NULL, 0, Fun, NULL, 0, NULL);
CloseHandle(hThread);
//////////////////////////////////////////////////////////////
char * sendData = "1";
send(sClient, sendData, strlen(sendData), 0);
GetImage();
cvtColor(frame, pre_gray, CV_BGR2GRAY);
tm_twoBall.start();

memset(array_des_x,0,30);//WWWWWWWWWWWWWWWWWWWWWWWWWWWWW
memset(array_des_y,0,30);

while(1)
{
    while(!get_image) {;}
    ////////////////////////////////////图片处理-轮廓获取//////////////////////////////////////
    frame_temp=frame.clone();
    get_image=0;
    cvtColor(frame_temp, gray, CV_BGR2GRAY);
    subtract(gray,pre_gray,pre_gray);
    morphologyEx(pre_gray,img_contour,MORPH_OPEN,element);
    threshold(img_contour,img_contour,7,255,THRESH_BINARY);
    findContours(img_contour,contours,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
    pre_gray=gray.clone();
    itc = contours.begin();
```

```
count_contour = 0;
min_distance = 2.0;
while(itc != contours.end())
{
    minEnclosingCircle(Mat(*itc),center,radius);
    area = contourArea(Mat(*itc));
    //arc=arcLength(Mat(*itc),1);
    contour_rect = boundingRect(Mat(*itc));
    contour_scale = (double)contour_rect.width/(double)contour_rect.width;
    if(contour_scale>1.5||contour_scale<1 || area>400 || area<40 || radius>30 )
        itc = contours.erase(itc);
    else
    {
        distance[count_contour] = matchShapes(contours1[0],*itc,CV_CONTOURS_MATCH_I2,0);
        if(distance[count_contour]>2){itc = contours.erase(itc);}
        else
        {
            itc++;
            count_contour++;
        }
    }
}
if(contours.size()>1)
    for(int i=0;i<count_contour;i++)
    {
        if(distance[i]<min_distance)
        {
            min_distance=distance[i];
            min_index=i;
        }
    }
if(contours.size()>1)
    {
        itc = contours.begin();
        while(itc != contours.end())
        {
            if(index_temp!=min_index || min_distance>0.7 )
                itc = contours.erase(itc);
            else {itc++;index_temp++;}
```

```
                }
            }
        if(contours.size()>0)
        {
            minEnclosingCircle(Mat(contours[0]),center,radius);
            drawContours(frame_temp, contours, -1, Scalar(0,0,255), 1);//CV_FILLED'
            point2D_r=center;
            if(center.x>30 && center.x<610 && center.y>30 && center.y<450)
                getBall_r=1;
            else
                getBall_r =0;
        }
        else
        {
            point2D_r.x=0;
            point2D_r.y=0;
            getBall_r=0;
        }
        finish_processing=1;
        imshow("frame",frame_temp);
        cvWaitKey(1);
    ////////////////////////////////////////空间三维预测/////////////////////////////////
        while(!get_nextFrame);//等着把三维标定做完才可以预测落点
        if(predict_track)
        {
             //开始计算落点
            getBallAll_num = 0;
            tm_cal_point.start();
            bool flag_bit = false;//是否击打的标志。根据y的速度方向。
            CalPointFinal(point_3D_x,point_3D_y,point_3D_z,time_save,dstHeight,dstx,dsty,dstime,flag_bit);
              tm_cal_point.stop();
              cout<<"tm_cal_point:"<<tm_cal_point<<endl;
              if(flag_bit)
              {
                  dstime = dstime - (int)(tm_cal_point.getTimeSec()*1000);//最后时间，等于总的预测时间 -
计算所需要的时间。
                  cout<<"dstx:"<<dstx<<"     "<<"dsty:"<<dsty<<"     "<<"time:"<<dstime<<endl;
                  dstime = dstime -200;
                  dstx = dstx-100;
```

```
                dsty = dsty +100;
                unsigned short send_temp = (unsigned short)(dstx>>8) + (unsigned short)dstx + (unsigned
short)(dsty>>8) + (unsigned short)dsty + (unsigned short)(dstime>>8) + (unsigned short)dstime;
                if(dsty>0    && dsty<5000    && dstx>0      && dstx<6100)
                {
                    for(int i=0;i<1;i++)
                    {
                        unsigned char d = 0x66;                    port.WriteData(&d,1);
                        d = (unsigned short)(dstx>>8);        port.WriteData(&d,1);
                        d = (unsigned short)(dstx);           port.WriteData(&d,1);
                        d = (unsigned short)(dsty>>8);        port.WriteData(&d,1);
                        d = (unsigned short)(dsty);           port.WriteData(&d,1);
                        d = (unsigned short)(dstime>>8);      port.WriteData(&d,1);
                        d = (unsigned short)(dstime);         port.WriteData(&d,1);
                        d = send_temp;                                port.WriteData(&d,1);
                        d = 0x88;                                  port.WriteData(&d,1);
                        cout<<"发送成功："<<endl;
                    }
                    //
                    //*************************************写文件
*****************************************//

                    //if((bb_file=fopen("data.txt","a"))==NULL)
                    //   printf("没打开data.txt");
                    //fprintf(bb_file,"%d     %f     %f     %f\n",getBallAll_num,dsty,dstx,dstime);
                    //fclose(bb_file);


                }
            }
            tm_cal_point.reset();
            predict_track=0;
        }
        flag_track = false;

        get_nextFrame=0;

    }
```

```
return 0;
}
```

主体程序所用函数：

```
void GetImage()
{
    CameraGetImageBuffer(g_hCamera,&g_tFrameHead,&g_readBuf,200);
    CameraImageProcess(g_hCamera, g_readBuf, g_pRgbBuffer,&g_tFrameHead);
    frame=Mat(480, 640, CV_8UC3, g_pRgbBuffer, g_tFrameHead.iWidth*(g_tFrameHead.uiMediaType ==
CAMERA_MEDIA_TYPE_MONO8?1:3));
    CameraReleaseImageBuffer(g_hCamera,g_readBuf);
    get_image=1;
}

int init_SDK()
{

    int                     iCameraCounts = 4;
    int                     iStatus=-1;
    tSdkCameraDevInfo       tCameraEnumList[4];
    CameraSdkInit(1);
    CameraEnumerateDevice(tCameraEnumList,&iCameraCounts);
    if(iCameraCounts==0){
        return -1;
    }
    iStatus = CameraInit(&tCameraEnumList[0],-1,-1,&g_hCamera);
    if(iStatus!=CAMERA_STATUS_SUCCESS){
        return -2;
    }
    CameraGetCapability(g_hCamera,&g_tCapability);
    g_pRgbBuffer = (unsigned
char*)malloc(g_tCapability.sResolutionRange.iHeightMax*g_tCapability.sResolutionRange.iWidthMax*3);
    CameraPlay(g_hCamera);
    return 0;
}


void CalPointFinal(double *f_x,double *f_y,double *f_z,double *f_t,int f_dst_height,int &f_dstx,int &f_dsty,int
&f_dsttime,bool &f_flag_bit)
```

```
{

    //这里假设用另外ＲＡＮＳＡＣ已经去除了杂点。


    double f_new_p[2][3];
    f_new_p[0][0]=f_x[0]*0.001;f_new_p[0][1]=f_y[0]*0.001;f_new_p[0][2]=f_z[0]*0.001;
    f_new_p[1][0]=f_x[5]*0.001;f_new_p[1][1]=f_y[5]*0.001;f_new_p[1][2]=f_z[5]*0.001;
    //cout<<"point:"<<f_new_p[0][0]<<"    "<<f_new_p[0][1]<<"    "<<f_new_p[0][2]<<endl;
    //cout<<"point:"<<f_new_p[1][0]<<"    "<<f_new_p[1][1]<<"    "<<f_new_p[1][2]<<endl;

    double f_t_all = f_t[0] + f_t[1] + f_t[2] +f_t[3] + f_t[4];
    //cout<<"time_1_6:"<<f_t_all<<endl;

    double f_v_x=0.0,f_v_y=0.0,f_v_z=0.0;
    f_v_x = (f_new_p[1][0] - f_new_p[0][0])/(f_t_all*0.001);
    f_v_y = (f_new_p[1][1] - f_new_p[0][1])/(f_t_all*0.001);
    f_v_z = (f_new_p[1][2] - f_new_p[0][2])/(f_t_all*0.001);

    if(f_v_y<0)
    {
        f_flag_bit = true;

        double distance_diff = 0.0;
        double distance_delt = 0.0001;
        double Km = Kf/Mass;

        double f_delt_x = 0.0,f_delt_y = 0.0,f_delt_z = 0.0;
        double f_width = 0.0,f_height=0.0,f_delt_w = 0.0,f_delt_h = 0.0;

        double f_new_v_xy = 0.0;
        double f_new_v_xy2=0.0;
        double f_new_v_z = 0.0;
        double f_new_v = 0.0;
        double f_new_v2 = 0.0;
        double f_angle_x = 0.0,f_angle_y = 0.0,f_angle_z = 0.0,f_angle_xy = 0.0;
        double f_a_all = 0.0,f_a_xy = 0.0,f_a_z = 0.0;

        double time_span = 0.0005;//0.5ms;
```

```
double time_span2 = pow(time_span,2)*0.5;
double time_span_cal = 0.002;//2ms    计算目标位置的时间间隔
double time_span_cal2 = pow(time_span_cal,2)*0.5;
int f_time_count = 0;

double f_predict_x=0.0,f_predict_y=0.0,f_predict_z=0.0;

int flag_stop =0;
int count_iterate    = (int)(f_t_all/(time_span*1000));

while(1)
{
    ++flag_stop;//停止标志
    f_new_v_xy2 = pow(f_v_x,2) + pow(f_v_y,2);
    f_new_v2 = pow(f_v_z,2) + f_new_v_xy2;
    f_new_v_xy = sqrt(f_new_v_xy2);
    f_new_v = sqrt(f_new_v2);
    f_new_v_z = f_v_z;
    f_angle_x = f_v_x / f_new_v_xy;//计算角度
    f_angle_y = f_v_y / f_new_v_xy;
    f_angle_xy = f_new_v_xy / f_new_v;
    f_angle_z = f_new_v_z / f_new_v;

    f_width = 0.0;f_height = 0.0;//初始化
    //cout<<f_new_v<<"    "<<f_new_v_xy<<"      "<<f_new_v_z<<endl;

    for(int i=0;i<count_iterate;i++)
    {
        f_a_all = -Km * f_new_v2;
        f_a_xy = f_a_all * f_angle_xy;
        f_a_z = f_a_all * f_angle_z - 9.8;
        f_delt_h = f_new_v_z * time_span + f_a_z * time_span2;
        f_delt_w = f_new_v_xy * time_span + f_a_xy * time_span2;

        f_width = f_width + f_delt_w;//计算运动距离。
        f_height = f_height + f_delt_h;

        f_new_v_xy = f_new_v_xy + f_a_xy*time_span;//更新速度
        f_new_v_z = f_new_v_z + f_a_z*time_span;
```

```
        f_new_v_xy2 = pow(f_new_v_xy,2);
        f_new_v2 = pow(f_new_v_z,2) + f_new_v_xy2;
        f_new_v_xy = sqrt(f_new_v_xy2);
        f_new_v = sqrt(f_new_v2);

        f_angle_xy = f_new_v_xy / f_new_v;
        f_angle_z = f_new_v_z / f_new_v;

    }

    f_predict_x = f_new_p[0][0] + f_width * f_angle_x;
    f_predict_y = f_new_p[0][1] + f_width * f_angle_y;
    f_predict_z = f_new_p[0][2] + f_height;


    f_delt_x = f_new_p[1][0] - f_predict_x;
    f_delt_y = f_new_p[1][1] - f_predict_y;
    f_delt_z = f_new_p[1][2] - f_predict_z;

    //distance_diff = pow(f_delt_x,2) + pow(f_delt_y,2) + pow(f_delt_z,2);
    //cout<<"delt:"<<f_delt_x<<"    "<<f_delt_y<<"    "<<f_delt_z<<endl;
    //cout<<"1:"<<pow(f_delt_x,2)<<"    "<<pow(f_delt_y,2)<<"    "<<pow(f_delt_z,2)<<endl;


    if( (f_delt_x<distance_delt && f_delt_y <distance_delt && f_delt_z <distance_delt)    ||flag_stop ==
51 )//如果满足条件，那么就开始预测。
    {
        //cout<<"distance:"<<distance_diff<<endl;

        f_new_v_xy2 = pow(f_v_x,2) + pow(f_v_y,2);
        f_new_v2 = pow(f_v_z,2) + f_new_v_xy2;
        f_new_v = sqrt(f_new_v2);
        f_new_v_xy = sqrt(f_new_v_xy2);
        f_new_v_z = f_v_z;
        f_angle_x = f_v_x / f_new_v_xy;//计算角度
        f_angle_y = f_v_y / f_new_v_xy;
        f_angle_xy = f_new_v_xy / f_new_v;
        f_angle_z = f_new_v_z / f_new_v;
```

```
f_width = 0.0;f_height = f_new_p[0][2];

while(1)
{
    ++f_time_count;

    f_a_all = -Km * f_new_v2;
    f_a_xy = f_a_all * f_angle_xy;
    f_a_z = f_a_all * f_angle_z - 9.8;

    f_delt_h = f_new_v_z * time_span_cal + f_a_z * time_span_cal2;
    f_delt_w = f_new_v_xy * time_span_cal + f_a_xy * time_span_cal2;

    f_width = f_width + f_delt_w;//计算运动距离。
    f_height = f_height + f_delt_h;

    if( f_new_v_z<0 && f_height < f_dst_height)
    {
        //cout<<"f_time_count:"<<f_time_count<<endl;
        f_dsttime = (int)(f_time_count * time_span_cal*1000);
        f_dsttime = f_dsttime - f_t_all;//计算第6个点到击打的时间。
        f_dstx = (int)((f_new_p[0][0] + f_width*f_angle_x)*1000);
        f_dsty = (int)((f_new_p[0][1] + f_width*f_angle_y)*1000);
        break;
    }

    f_new_v_xy = f_new_v_xy + f_a_xy*time_span_cal;//更新速度
    f_new_v_z = f_new_v_z + f_a_z*time_span_cal;
    f_new_v_xy2 = pow(f_new_v_xy,2);
    f_new_v2 = pow(f_new_v_z,2) + f_new_v_xy2;
    f_new_v_xy = sqrt(f_new_v_xy2);
    f_new_v = sqrt(f_new_v2);

    f_angle_xy = f_new_v_xy / f_new_v;
    f_angle_z = f_new_v_z / f_new_v;
}

break;
```

```
            }
            else
            {
                if(flag_stop<50)
                {
                    f_v_x = f_v_x + correct_k * f_delt_x;//修正初速度
                    f_v_y = f_v_y + correct_k * f_delt_y;
                    f_v_z = f_v_z + correct_k * f_delt_z;
                    //cout<<"v:"<<f_v_x<<"   "<<f_v_y<<"   "<<f_v_z<<endl;

                }
                else
                {
                    cout<<"sha bi!!!"<<endl;
                    //break;//假设迭代没有求出相近结果，但是，我想结果也不会差到哪里把。
                }
            }
        }
    }
    else
    {
        f_flag_bit = false;
    }
}
void pointClearNoise(double *f_x,double *f_y,double *f_z,double *f_t,double &f_t_all,double
&f_angle_x,double &f_angle_y)
{
    vector<Point2f> f_points;
    Vec4f f_lines;
    float f_k=0.0,f_b=0.0;
    float f_distance = 0.0;
    float f_distance_delt = 500.0;//可容许的误差范围
    bool f_flag_iterate = true;
    int f_point_count = 6;
    vector<Point2f>::const_iterator f_itc;


    for(int i=0;i<f_point_count;i++)//将所有的点放到直线拟合的容器里。
    {
```

```
        f_points.push_back(Point2f(f_x[i],f_y[i]));
    }

    while(f_flag_iterate)//当所有的点都满足容许的范围之内时。迭代结束。这些点可以作为使用点。
    {
        f_flag_iterate = false;
        //for(int i=0;i<f_point_count;i++)//将所有的点放到直线拟合的容器里。
        //{
        //  f_points.push_back(Point2f(f_x[i],f_y[i]));
        //}

        f_itc = f_points.begin();
        fitLine(Mat(f_points),f_lines,CV_DIST_L2,0,0.01,0.01);


        for(int i=0;i<f_point_count;i++)
        {
            if(f_lines[0] < 0.001)
            {
                f_distance = abs(f_lines[2] - f_x[i]);
            }
            else
            {
                f_k = f_lines[1]/f_lines[0];//0代表的是cos，1代表的是sin
                f_b = f_lines[3] - f_lines[2] * f_k ;
                f_distance = abs(f_k * f_x[i] - f_y[i] + f_b)/sqrt(pow(f_k,2) + 1);//点到直线距离公式。
            }
            if(f_distance > f_distance_delt)
            {
                f_flag_iterate = true;
                if(i==0)//这里如果学会用vector的话，那么就不用再一个一个赋值了。直接对vector操作。
                {
                    int j = i;
                    for(;j<f_point_count-2;j++)
                    {
                        f_x[j] = f_x[j+1];
```

```
                    f_y[j] = f_y[j+1];
                    f_z[j] = f_z[j+1];
                    f_t[j] = f_t[j+1];
                }
                f_x[j] = f_x[j+1];
                f_y[j] = f_y[j+1];
                f_z[j] = f_z[j+1];
                //f_t[j] = 0.0;
                //f_x[j+1]=0.0;
                //f_y[j+1]=0.0;
                //f_z[j+1]=0.0;
            }
            else
            {
                int j=i;
                for(;j<f_point_count-2;j++)//如果不等于零，那么，将前一次的时间与这一次的时间相加，赋给前一个时间数组。
                {
                    f_x[j] = f_x[j+1];
                    f_y[j] = f_y[j+1];
                    f_z[j] = f_z[j+1];
                    f_t[j-1] = f_t[j-1] + f_t[j];
                }
                f_x[j] = f_x[j+1];
                f_y[j] = f_y[j+1];
                f_z[j] = f_z[j+1];
                //f_t[j] = 0.0;
                //f_x[j+1]=0.0;
                //f_y[j+1]=0.0;
                //f_z[j+1]=0.0;
            }
            --f_point_count;
            f_itc = f_points.erase(f_itc);
        }
        else
        {
            ++f_itc;
        }
    }
```

```
    }
    f_angle_x = f_lines[0];//j将拟合的直线角度传出
    f_angle_y = f_lines[1];
    --f_point_count;
    if(f_angle_x<0.001)
    {
        f_x[1] = f_x[0];
        f_y[1] = f_y[f_point_count];
    }
    else
    {
        double f_k_vertical = -1/f_k;
        f_x[1] = (f_y[f_point_count] - f_k_vertical * f_x[f_point_count] - f_b) / (f_k -
f_k_vertical);
        f_y[1] = f_k * f_x[1] + f_b;
    }
    for(int i=0;i<f_point_count;i++)
    {
        f_t_all = f_t_all + f_t[i];
    }
}
```

客户端部分程序：

```
//**********************头文件*************************//
#include <WINSOCK2.H>
#include <STDIO.H>
#include <CameraApi.h>
#include <cv.h>
#include <opencv2/opencv.hpp>
#include "highgui.h"
using namespace cv;
using namespace std;

#pragma comment(lib, "MVCAMSDK.lib")
#pragma  comment(lib,"ws2_32.lib")
//********************全局变量*************************//
int                     g_hCamera = -1;     //设备句柄
unsigned char           * g_pRgbBuffer=NULL;     //处理后数据缓存区
tSdkCameraCapbility     g_tCapability;      //设备描述信息
```

```
BYTE                    *g_readBuf=NULL;     //画板显示数据区
tSdkFrameHead           g_tFrameHead;        //图像帧头信息

SOCKET sclient;

Mat frame;
bool get_image=0;
bool getBall_1=0;
bool finish_processing=0;
Point2i point2D_1;
//********************宏定义*****************************//
#define ServerAdress  "169.254.104.82"

//********************函数声明*************************//
int init_SDK();

void GetImage();

////////////////////////////////////////////////////////////////////////////////////////
TickMeter tm_img,tm_all,tm_send;
int tm_count = 0;

DWORD WINAPI Fun(LPVOID lpParamter)
{
    char sendData[30],recData[20];
    while(1)
    {
        memset(recData,0,20);
        tm_img.start();
        while(!finish_processing);
        tm_img.stop();
        //cout<<"tm_img:"<<tm_img.getTimeSec()<<endl;
        tm_img.reset();
        tm_send.start();
        if(getBall_1)////? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
        {
            //sendData[0]='9';

    //sendData[1]=(char)((int)point2D_1.x/100);sendData[2]=(char)(((int)point2D_1.x%100)/10)
```

```
;sendData[3]=(char)(((int)point2D_1.x%10));

    //sendData[4]=(point2D_1.x/100);sendData[5]=((point2D_1.x%100)/10);sendData[6]=(char)((p
oint2D_1.x%10));
            //sendData[7]=0x00;
            //cout<<"tm_count:"<<tm_count<<endl;
            if(point2D_1.x>=100 && point2D_1.y>=100)
            sprintf(sendData,"%d%d%d%d",9,point2D_1.x,point2D_1.y,tm_count);
            else if(point2D_1.x>=100 )
            sprintf(sendData,"%d%d0%d%d",9,point2D_1.x,point2D_1.y,tm_count);
            else if(point2D_1.y>=100 )
            sprintf(sendData,"%d0%d%d%d",9,point2D_1.x,point2D_1.y,tm_count);
            send(sclient, sendData, strlen(sendData), 0);
            //cout<<"     "<<point2D_1.x<<"   "<<point2D_1.y<<endl;
        }
        else//没球发送0
        {
            /*sendData[0]='0';
            sendData[1]=0x00;*/
            sprintf(sendData,"%c0%d0%d%d",'0',0,0,tm_count);//没有球，也要发送时间。
            send(sclient, sendData, strlen(sendData), 0);
        }
        tm_send.stop();
        //cout<<"tm_send:"<<tm_send.getTimeSec()<<endl;
        tm_send.reset();
        finish_processing=0;

        while(!recv(sclient,recData,1,0));
        tm_all.stop();
        //if(tm_all.getTimeSec()>0.015)
        //cout<<"tm_all:"<<tm_all.getTimeSec()<<endl;
        tm_count = (int)(tm_all.getTimeSec()*10000);
        tm_all.reset();
        tm_all.start();
        if(recData[0]=='1')
        {
            GetImage();
        }
    }
```

```
}
int main(void)
{
    //**********************变量定义**********************************//
    vector<vector<Point>>contours_moban,contours;
    Mat pre_gray,gray,img_contour,frame_temp;
    Mat element(5,5,CV_8U,Scalar(1));
    Point2f center;float radius;double area,arc,contour_scale;Rect contour_rect ;
    int count_contour=0;double min_distance=1.5;int min_index;double distance[100];int
index_temp=0;
    vector<vector<Point>>::const_iterator itc;

    //*****************************初始化*****************************//
    init_SDK();
    Mat moban=imread("1.png");
    cvtColor(moban,moban,CV_BGR2GRAY);
    findContours(moban,contours_moban,CV_RETR_LIST,CV_CHAIN_APPROX_NONE);
    contours_moban.erase(contours_moban.end()-1);
    //*****************************通信初始化*****************************//
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA data;
    if(WSAStartup(sockVersion, &data) != 0)
    {
        return 0;
    }
    if(sclient == INVALID_SOCKET)
    {
        printf("invalid socket !");
        return 0;
    }
    sclient= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in serAddr;
    serAddr.sin_family = AF_INET;
    serAddr.sin_port = htons(8888);
    serAddr.sin_addr.S_un.S_addr = inet_addr(ServerAdress);
    cout<<"44"<<endl;
    if (connect(sclient, (sockaddr *)&serAddr, sizeof(serAddr)) == SOCKET_ERROR)
    {
        printf("connect error !");
```

```
    closesocket(sclient);
    while(1);
    return 0;
}

char recData[3],*sendData="2";
int ret=0;
while(!recv(sclient,recData,1,0));
if(recData[0]='1')
{
    GetImage();
}
cvtColor(frame, pre_gray, CV_BGR2GRAY);
//GetImage();//??????????????????????
////////////////////////////////////////////////////////////////////////////
HANDLE hThread = CreateThread(NULL, 0, Fun, NULL, 0, NULL);
CloseHandle(hThread);
finish_processing=0;
////////////////////////////////////////////////////////////////////////////
TickMeter tm;
while(1)
{
    while(!get_image) {;}
    tm.start();
    frame_temp=frame.clone();
    get_image=0;
    cvtColor(frame_temp, gray, CV_BGR2GRAY);
    subtract(gray,pre_gray,pre_gray);
    morphologyEx(pre_gray,img_contour,MORPH_OPEN,element);
    threshold(img_contour,img_contour,7,255,THRESH_BINARY);
    findContours(img_contour,contours,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
    pre_gray=gray.clone();
    itc = contours.begin();
    count_contour = 0;//??????????????????????
    min_distance = 2.0;
    while(itc != contours.end())
    {
        minEnclosingCircle(Mat(*itc),center,radius);
        area = contourArea(Mat(*itc));
```

```
            //arc=arcLength(Mat(*itc),1);
            contour_rect = boundingRect(Mat(*itc));
            contour_scale = (double)contour_rect.width/contour_rect.width;
            if(contour_scale>1.5||contour_scale<1 || area>400 || area<40 || radius>30 )
                itc = contours.erase(itc);
            else
            {
                distance[count_contour] =
matchShapes(contours_moban[0],*itc,CV_CONTOURS_MATCH_I2,0);
                if(distance[count_contour]>2){itc = contours.erase(itc);}
                else
                {
                    itc++;
                    count_contour++;
                }
            }
        }
        if(contours.size()>1)
            for(int i=0;i<count_contour;i++)
            {
                if(distance[i]<min_distance)
                {
                    min_distance=distance[i];
                    min_index=i;
                }
            }
            if(contours.size()>1)
            {
                itc = contours.begin();
                while(itc != contours.end())
                {
                    if(index_temp!=min_index || min_distance>0.7 )
                        itc = contours.erase(itc);
                    else {itc++;index_temp++;}
                }
            }
            if(contours.size()>0)
            {
                minEnclosingCircle(Mat(contours[0]),center,radius);
```

```
                drawContours(frame_temp, contours, -1, Scalar(0,0,255), 1);//CV_FILLED
                //circle(frame_temp,Point(center),1,Scalar(255,0,0),-1);
                point2D_l=center;
                if(center.x>30 && center.x<610 && center.y>30 && center.y<450)
                    getBall_l=1;
                else
                    getBall_l =0;
            }
            else
            {
                point2D_l.x=0;
                point2D_l.y=0;
                getBall_l=0;
            }
            finish_processing=1;
        //memset( distance, 0, sizeof(distance));
            tm.stop();
            //cout<<"time_img:"<<tm.getTimeSec()<<endl;
            tm.reset();
            imshow("frame",frame_temp);
            cvWaitKey(1);

    }


}
```

```
void GetImage()
{
    CameraGetImageBuffer(g_hCamera,&g_tFrameHead,&g_readBuf,200);
    CameraImageProcess(g_hCamera, g_readBuf, g_pRgbBuffer,&g_tFrameHead);
    frame=Mat(480, 640, CV_8UC3, g_pRgbBuffer, g_tFrameHead.iWidth*(g_tFrameHead.uiMediaType
== CAMERA_MEDIA_TYPE_MONO8?1:3));
    CameraReleaseImageBuffer(g_hCamera,g_readBuf);
```

```
    get_image=1;
}

int init_SDK()
{

    int                     iCameraCounts = 4;
    int                     iStatus=-1;
    tSdkCameraDevInfo       tCameraEnumList[4];
    CameraSdkInit(1);
    CameraEnumerateDevice(tCameraEnumList,&iCameraCounts);
    if(iCameraCounts==0){
        return -1;
    }
    iStatus = CameraInit(&tCameraEnumList[0],-1,-1,&g_hCamera);
    if(iStatus!=CAMERA_STATUS_SUCCESS){
        return -2;
    }
    CameraGetCapability(g_hCamera,&g_tCapability);
    g_pRgbBuffer = (unsigned
char*)malloc(g_tCapability.sResolutionRange.iHeightMax*g_tCapability.sResolutionRange.iWidth
Max*3);
    CameraPlay(g_hCamera);
    return 0;
}
```

在 学 取 得 成 果

在学期间所获的奖励

2012.5	北京科技大学首届“单片机大赛”	一等奖
2012.10	北京科技大学自动化学院“晚缘”项目	优秀志愿者
2013.5	微软“创新杯”PPT 应用大赛	最佳创意奖
2014.5	全国大学生机器人大赛	一等奖

致谢

在 MEI 的这一年半当中，我收获了让我受益终生的经验。从技术上，我在这期间才开始编写出一些像样的 c++程序；我在这里接触到了机器视觉，学习了图像处理库 OpenCV，并让我深刻的感觉到了实践对于科研的重要性。然而，能让我一生都铭记于心的，是在这里收获的精神上的财富。在这里，我找到了理想，从初识机器视觉，到深入的了解，这个过程让我喜欢上机器视觉这个领域，并将它作为自己愿意为之终生奋斗的地方，另外，在这里，我收获到了深厚的友谊，他们不善言辞，但乐于奉献，敢于承担，能与他们在一起奋斗，让我感觉到是一件自豪的事情，他们对事的认真、对人的真诚深深地打动了我，并影响我也成为这样一个人。

首先，在这里，我非常感谢王粉花老师对我的监督，在王老师的鞭策下，我才能在规定的时间内做出成果，完成比赛。我还要感谢曾慧老师对我的技术上的支持，在曾老师耐心的指导下，使得我能突破瓶颈，攻克难关。

其次，感谢学校对这项比赛的大力支持，为本科生搭建了一个锻炼的平台，让我们在这里尽情的展现自己的才华，为以后的工作学习打下了夯实的基础。

此外，我还要感谢 MEI 的每一位队员，感谢他们在技术上对我的无私的帮助，在生活中对我的容忍与谦让，在工作上对我潜移默化的影响。是他们，让我这一年过的如此的充实。