编 号

本科生毕业设计（论文）

题目：智能视觉 PTZ 自动跟踪监控研究

物联网工程 学 院 物联网工程 专 业

学 号 0306110216

学生姓名 余 拓

指导教师 陈 莹 副教授

二〇一五年六月

设计总说明

针对传统安防监控系统中，单个摄像机监控范围有限的弊端，提出了一种基于 PTZ（Pan-Tile-Zoom, 旋转-俯仰-缩放）摄像机的自动跟踪监控系统。相较于传统监控摄像机，PTZ 摄像机的优势在于，其云台能够使摄像机镜头能够在一定程度上自由旋转和缩放。这大大扩展了单个摄像机的视野。本文总结了 PTZ 摄像机应用现状和视觉跟踪算法的研究现状，提出了自动跟踪监控系统的总体结构，设计并实现了视觉跟踪程序和 PTZ 摄像机控制程序，并将两者结合，通过一个图形用户界面程序为用户提供调用跟踪功能的入口。

本文使用的 PTZ 摄像机具有网络通信功能，可以通过 HTTP 与同一局域网下的终端计算机进行通信。PTZ 摄像机还提供云台控制功能函数的接口，终端计算机可向其发送 HTTP 请求来调用这些函数。PTZ 摄像机控制程序以 C#语言编写。视觉跟踪程序以 C++语言编写，生成为动态链接库文件，并提供 C#调用接口，以供 PTZ 摄像机程序调用。图形用户界面程序利用 WPF 编写。在 Visual Studio 集成开发环境完成所有代码的编写。

在对目标进行视觉跟踪的开始，终端计算机向 PTZ 摄像机发出获取实时图像的请求，PTZ 摄像机得到该请求后，将其当前获取的实时图像以单幅 JPG 图像的方式返回给终端计算机，并输入视觉跟踪程序；跟踪目标的大小和初始位置需要由用户在图形交互界面利用鼠标输入；视觉跟踪程序得到目标初始坐标和实时图像，开始对目标进行跟踪。

在对目标进行视觉跟踪的过程中，以处理一帧图像为一个周期，在一个周期中：PTZ 摄像机先将其采集到的实时视频数据传入局域网，终端计算机通过向同一局域网获取实时视频数据；视觉跟踪算法得到初始目标坐标后，开始对目标进行跟踪，并随实时图像的更新而不断更新目标的跟踪结果；PTZ 摄像机控制程序单独工作在一个线程，它根据视觉跟踪程序的跟踪结果，向 PTZ 摄像机发送对应的指令，控制云台运动，以确保跟踪目标始终位于图像中央部分。

视觉跟踪程序选用基于内核化相关滤波器的高速跟踪算法作为跟踪算法。本文深入研究了该算法，分析了其相较于其他跟踪算法的优点，最终将其运用于自动跟踪监控系统中。算法以 HOG 特征作为目标的图像特征，使用内核化相关滤波器。

PTZ 摄像机控制程序工作于一个独立的线程，程序预先设定了一个阈值，当跟踪目标中心距离图像中心超过这个阈值时，程序便开始控制云台运动，使目标中心与图像中心之间的距离缩小，距离小于阈值时，云台停止运动。

最后，本文通过实验验证了，跟踪算法可以对单个目标进行实时跟踪，PTZ 摄像机控制程序可以根据跟踪结果控制云台运动，以保证目标位于摄像机视野中。整个系统实时性较高，准确性较好，并具有一定的鲁棒性。

关键词：PTZ 摄像机；智能监控；视觉跟踪；内核化相关滤波器

ABSTRACT

Aiming at the drawback of limited vision of a single camera in traditional security surveillance system, an automatic tracking surveillance system based on PTZ (Pan-Tile-Zoom) camera is proposed. Compared to conventional surveillance cameras, PTZ camera advantage is that it enables the lens to rotate and zoom free to a certain extent, it expands the horizons of a single camera greatly. The paper summarizes the current status of PTZ camera application and tracking algorithms research, proposes the overall structure of the automatic tracking surveillance system, designs and implements a visual tracking program and a PTZ camera control program, and combines them, provides users an entrance to call tracking functions through a graphical user interface program.

PTZ camera used herein has a network communication function, can communicate with a terminal computer through HTTP under the same local network. PTZ camera also offers some PTZ control function interface, a terminal computer may sends HTTP requests to call these functions. PTZ camera control program is coded in C # language. Visual tracking program is coded in C ++ language, release as a dynamic link library file, and provides C # interface for PTZ camera program to call. The graphical user interface program is coded with WPF. All the code is coded and finished in Visual Studio integrated development environment.

At the beginning of the visual tracking, the terminal computer sends request to ask the PTZ camera for real-time images, after the PTZ camera gets the request, it return the real-time image to the terminal computer in form of single JPG image, which is then inputted to visual tracking program; The target size and the initial position is inputted by a user in a graphical interface with a mouse; Visual tracking program obtains the initial information and the read-time image, and then begins to track.

In the process of visual tracking, the paper regard a process of a frame as a period. The PTZ camera sends the real-time video data it captures to a local area network, the terminal computer gets the video data from the same local area network; after the visual tracking program gets initial coordinate, it begins to track the target, and updates the result of tracking with read-time image updates; The PTZ camera control program works in another thread, it sends corresponding commands to the PTZ camera according to the visual tracking program's result, ensure the target is always at the center of the image by controlling the rotational station.

Visual tracking program uses the high-speed tracking with kernelized correlation filters as tracking algorithm. The paper has an in-depth study in the algorithm, analyzes its advantages compared to other tracking algorithms and applies it on automatic tracking surveillance system at last. The algorithm uses HOG features as characteristics of the target image, uses kernelized correlation filters.

PTZ camera control program works in a separate thread, the program set a threshold previously, when the distance between the target and the center of the image exceeds the threshold, the program begins to control the rotational station, so that the distance reduce. When the distance is less than the threshold, the rotational station stops moving.

Finally, verified by experiments, tracking algorithm allows real-time tracking of single target, PTZ camera control program can control the rotational station by tracking results, to ensure that the target is in the camera vision. The whole system is well real-time and accurate, and has a robustness.

Keywords: PTZ camera; intelligent monitoring; visual tracking; kernelized correlation filters

目 录

第1章 绪论

本章将介绍本文的研究背景与意义；分析 PTZ 摄像机应用现状；分析视觉跟踪算法的研究现状，并介绍了一些经典的算法和一些新兴的算法；在最后一节中，本章对后面内容的章节安排作了简要介绍。

1.1 研究背景与意义

现代社会人们对公共场所的安全需求不断提升，越来越多的视频监控系统运用到了各种公共场所，如学校、银行、地铁站、停车场等等。传统的视频监控并不提供视频的处理和分析。除了监控画面本身，监控系统几乎无法其他任何有价值的信息。为了节省人力物力，和更好地发挥视频监控的效能，视频的自动实时检测和分析就变得十分必要，智能视频监控系统的重要性也越来越引起人们的注意。

PTZ 摄像机较于传统监控摄像机，优势在于其自身的云台使摄像机镜头能够在一定程度上自由旋转和缩放。加装网络通信模块的 PTZ 摄像机还能够利用局域网与终端设备进行通信，这样不仅摆脱了数据连接线的束缚，使摄像机的布置更为自由，还增加了对包括智能手机在内的移动终端设备的支持。

将高性能视觉跟踪程序和 PTZ 摄像机控制程序相结合，便可以实时跟踪运动目标，同时根据目标的位置，控制 PTZ 摄像机云台的旋转和镜头的缩放，针对被锁定的运动目标进行视觉导向的自动跟踪，以确保跟踪目标持续出现在镜头中央。这样不仅方便用户获取除视频画面本身的信息，还能提供用户感兴趣的目标的信息，方便监控的事后取证。可以广泛运用于银行、机场、车站、海关、交通、电力、厂矿企业等场合的安防监控。另外，这种算法和硬件的结合符合当下物联网产品多领域合作开发的趋势，具有良好的开发及应用前景。

1.2 PTZ 摄像机应用现状

现在市面上的各类 PTZ 摄像机产品种类丰富，但参数各不相同。高价位产品大多在摄像机上加装了网络通信模块，甚至内置了自动跟踪的程序，摄像机本身也具备诸如高分辨率、高帧率和多倍光学变焦等的优秀性能。低价位产品则良莠不齐，性能往往差强人意。受限于成本，很多场所的安防监控系统中所采用的摄像机的性能并不高。

另一种情况是，虽然用户在监控系统中采用了高性能的 PTZ 摄像机，却没有完全发挥 PTZ 摄像机的特性和全部性能，仅仅将它们当作普通摄像机来用。这造成了一定程度上的资源浪费。除了用户忽略了其丰富的功能、没有在实际使用这些功能外，其本身的某些智能特性——如自动跟踪——的性能也可能达不到用户的需求，在运行速度和跟踪精度上存在问题，也是用户没有将高性能 PTZ 摄像机物尽其用的原因之一。

所以综上所述，PTZ 摄像机不仅在硬件上有提升的空间，其包括但不限于自动跟踪程序的智能特性，也有着巨大的优化空间。

1.3 视觉跟踪算法的研究现状

视觉跟踪问题近几十年来一直是计算机视觉领域的热点问题之一。二十世纪 80 年代以前，受计算机性能的限制，对图像的处理与分析还主要以静态图像为主。二十世纪初 Berthold K. Horn 和 Brian G. Schunck 提出了光流法[1]（Optical Flow），这使得动态图像序列分析进入了一个研究的高潮，此后众多视觉跟踪算法被提出。

1.3.1 经典算法

文献[2; 3]中回顾了视觉跟踪算法从二十世纪 80 年代到二十一世纪初的发展情况，并将这些算法分为四类：基于区域的跟踪（Region-based tracking）、基于特征的跟踪（Feature-based tracking）、基于变形模板的跟踪（Deformable-template-based tracking）、基于模型的跟踪（Model-based tracking）。

1) 基于区域的跟踪

首先通过图像分割，或是人为设定的方式，获取包含目标的图像块，然后运用相关匹配的方法在图像序列中对目标进行跟踪。相关匹配可采用纹理、特征、或是颜色。平方和准则（sum of square different, SSD）是最常用的相关准则。结合线性预测、二次曲线预测或 Kalman 预测可以对每帧图像中的目标位置进行估计。

2) 基于特征的跟踪

与基于区域的跟踪算法相同，基于特征的跟踪也使用相关算法。但它使用目标的某些局部特征而不是目标整体作为相关时的对象。特征一般比目标图像本身的信息量要小得多，这样才能在相关计算的时候减少计算量，以提升算法性能。图像大致有颜色、边缘、形状等视觉特征；直方图、灰度概率密度函数等统计特征；傅立叶描述子、小波变换系数等变换系数特征；矩阵的奇异值分解等代数特征。

3) 基于变形模板的跟踪

根据跟踪目标的轮廓在给定图像序列中找出感兴趣的部分。可变形模板是纹理或边缘可以按一定限制条件变形。它的不足支出在于它依赖有关被跟踪目标形状的先验知识，且有轻微的扰动时，物体的轮廓就会被扭曲。

4) 基于模型的跟踪

多用于特定形态的目标的跟踪，尤其对于刚体，如汽车的跟踪。它通过先验知识来获得目标的三维结构模型和运动模型，然后根据实际的图像序列，确定目标的三维模型参数，进而确定目标的实时运动参数。这种方法虽然能够在目标姿态改变时精准地跟踪目标，但同时也需要大量的计算量，而且几何模型的精度决定了运动分析的精度。

1.3.2 新兴算法

Y Wu 等人[4]为评价和分析视觉跟踪算法制定了一套标准（Benchmark），其中包含 50 个测试视频，能够测试算法在如目标被部分遮挡、光线昏暗、目标高速移动等各种情况下的性能。他们还对 2002 至 2012 年间的 29 个算法做出了测试，下面简要介绍其中几种近年来出现的高性能跟踪算法：

1) 基于自适应结构化局部稀疏外观模型的视觉跟踪算法[5]（Adaptive Structural Local Sparse Appearance model, ASLA）

ASLA 算法基于结构化局部稀疏外观模型。这种表示法通过一个异常队列池的方法同时利用了目标的局部信息和空间信息。从局部补丁中汇集获取的相似点，不仅能够使目标定位得更精准，还能解决目标遮挡问题。此外，作者还采用了结合了具备增量子空间学习和稀疏表示的模板更新策略。这种策略使得模板目标的外观变化与漂移的可能性更小，并降低了目标被遮挡模板所产生的影响

2) 跟踪学习检测[6; 7]（Tracking-Learning-Detection, TLD）

TLD 算法使用了 PN 学习（Postive-Negative Learning），这是一种利用带标记的样本和不带标记的样本之间存在的结构性特征来迭代地训练两类分类器并改善分类器性能的方法。PN 学习受正约束（Postive constraint）和负约束（Negative constraint）操控，它们被用来限制样本的标记过程。TLD 算法在长时间对目标进行跟踪上展现出了良好的性能。

3) Struck 算法[8]

Struck 算法主要提出一种基于结构输出预测的自适应视觉目标跟踪的框架，通过明确引入输出空间满足跟踪功能，能够避免中间分类环节，直接输出跟踪结果。同时，为了保证实时性，该算法还引入了阈值机制，防止跟踪过程中支持向量的过增长。

除了上面说到的几种算法，还有诸多优秀算法，这里不一一介绍。

视觉跟踪技术如今已经比较成熟，并且已经有部分成果进入实用化阶段，但随着计算机技术的不断发展，视觉跟踪作为计算机视觉领域中的核心技术之一，仍然具有强大的生命力和广阔的发展空间。

1.4 章节安排

本文设计并实现了一个 PTZ 摄像机自动跟踪监控系统。该系统将基于内核化相关滤波器的高速跟踪算法和 PTZ 摄像机的控制结合在一起，在跟踪目标的同时控制 PTZ 摄像机的云台运动，使运动目标能够始终保持在摄像机镜头的中央。该系统具有良好的实时性和精确性，并具有一定的鲁棒性。

本文采用如下的结构，来完成对方法的论述以及对实验结果的分析。

第一章绪论中，本文阐述了本课题的研究背景和研究意义，并介绍了 PTZ 摄像机的应用现状和视觉跟踪算法的研究现状。

第二章主要介绍了本文所提出的 PTZ 摄像机自动跟踪监控系统的总体架构，介绍了实现该系统所使用的一些软件工具。本章简要介绍了基于内核化相关滤波器[9]（Kernelized Correlation Filters，KCF）的高速跟踪算法，分析了其优点，并简单介绍了 PTZ 摄像机的控制程序。

第三章详细介绍了本文所使用的视觉跟踪程序，并在实验的基础上对基于内核化相关滤波器的高速跟踪算法进行分析。

第四章详细介绍了本文所使用的 PTZ 摄像机控制程序，分析了 PTZ 摄像机控制程序

的工作过程，并给出了实验结果，验证了 PTZ 摄像机控制算法的鲁棒性。

第五章是实验结果和分析，将第三章和第四章的结果结合成完整的自动跟踪监控系统，将实时视频输入系统，分析其输出结果。

第六章是总结与展望，客观总结了本文的工作，提出了几点不足之处，并对智能 PTZ 自动跟踪监控在未来的发展和应用做出展望。

1.5 本章小结

本章探讨了 PTZ 自动跟踪监控系统的研究背景及研究意义，分析了 PTZ 摄像机的应用现状，PTZ 摄像机要想充分发挥其作用，必须结合视觉跟踪算法。本章还介绍了几类经典的视觉跟踪算法，并简要介绍了 ASLA、TLD、Struck 这几种新兴算法。最后本章对下文中的内容做了章节安排。

第2章 系统总体分析与设计

本章主要分析 PTZ 摄像机自动跟踪监控系统的总体架构，介绍了本文在实施中所用到的工具，并简单介绍了了视觉跟踪程序中所用到的跟踪算法及 PTZ 摄像机控制程序。

2.1 监控系统设计的基本方案与原理

本文所使用的自动跟踪监控研究系统由具有网络通信功能的 PTZ 摄像机和终端 PC 组成，两者连入同一 AP（Access Point，访问接入点），位于同一局域网，摄像机和终端之间可以通过 HTTP 协议进行通信。由于终端 PC 和 AP、PTZ 摄像机和 AP 之间的通信并不是本文分析的重点，所以在分析整个系统时，认为终端 PC 于 PTZ 摄像机之间直接通信，AP 视作透明。系统各部分的连接如图 2-1 所示。

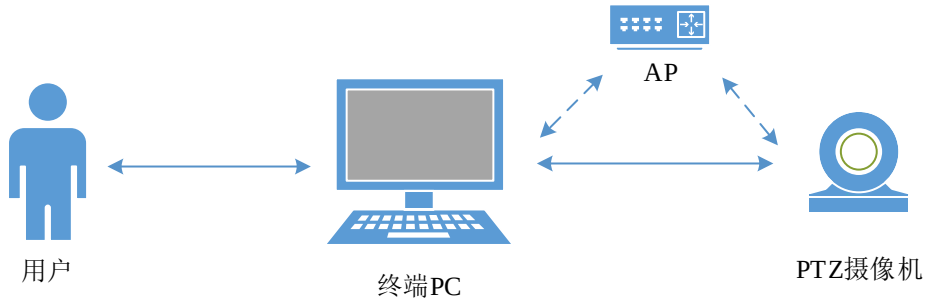


图 2-1 PTZ 自动跟踪监控系统结构

图 2-2 展示了 PTZ 自动跟踪监控系统的工作流程图。这里以处理一帧图像为一个周期，来分析系统在工作过程中的各个部分的工作原理。当终端 PC 向 PTZ 摄像机发出获取实时图像的请求，PTZ 摄像机得到该请求后，将其当前获取的实时图像以单幅图像的方式返回给终端 PC，图像格式为 JPG；终端 PC 得到当前实时图像，并将其输入视觉跟踪程序；视觉跟踪程序经过处理分析，得到跟踪目标当前的位置坐标 ，输入 PTZ 摄像机控制程序；PTZ 摄像机控制程序根据目标坐标，来决定 PTZ 摄像机云台的运动参数，并向 PTZ 摄像机发送对应的指令；最终云台根据指令运动，一个周期结束，PTZ 摄像机继续获取下一帧实时图像并开始一个新的周期。

2.2 工具介绍

本文使用 Microsoft 公司的 Visual Studio 来完成整个工程所有代码的编写和生成，包括视觉跟踪程序、PTZ 摄像机控制程序和界面程序。其中视觉跟踪程序用托管 C++[10]编写，使用了 OpenCV[11]（Open Source Computer Vision，开源计算机视觉库）库函数，最终程序生成为 DLL（Dynamic Link Library，动态链接库）文件，并提供 C#程序调用接口。PTZ 摄像机控制程序用 C#编写。界面程序利用 WPF（Windows Presentation Foundation，Windows 呈现基础）编写。

在视觉跟踪程序中还使用到了 Piotr's Computer Vision Matlab Toolbox[12]库函数中提取 HOG（Histogram of oriented gradient，方向梯度直方图）特征的相关函数和 FFTW[13]库函

数中的快速傅立叶变换函数。另外本文使用了 Matlab（Matrix Laboratory，矩阵实验室）对基于内核化相关滤波器的高速跟踪算法进行仿真。

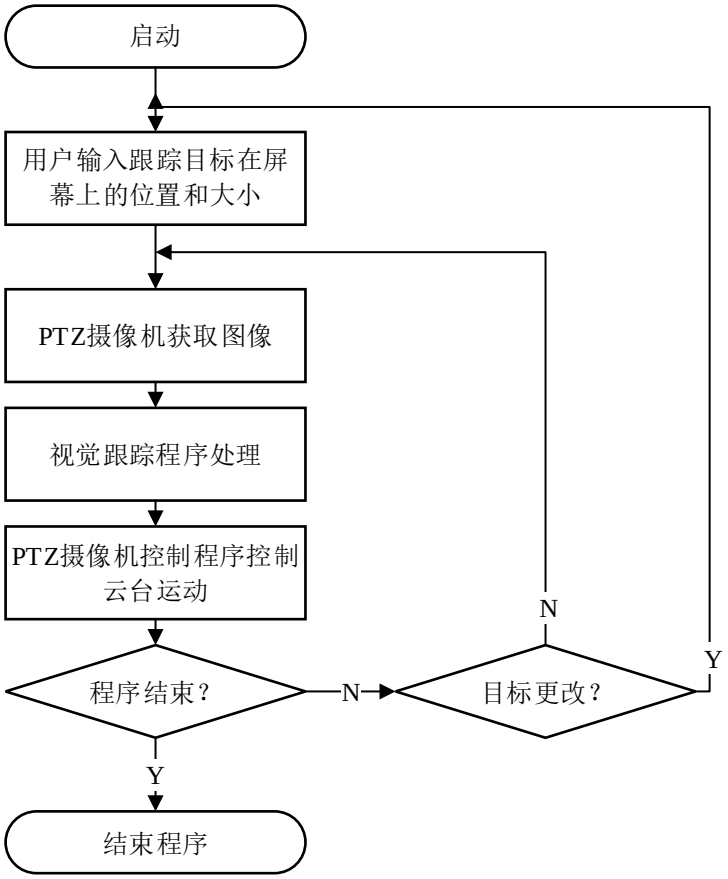


图 2-2 PTZ 自动跟踪监控系统工作流程图

2.3 视觉跟踪算法

本文使用J F Henriques 等人[9]提出的基于内核化相关滤波器的高速跟踪算法作为视觉跟踪程序的算法。由于结果数据的矩阵是循环矩阵，可以用离散傅里叶变换将其对角化，从而减少几个数量级的存储空间和计算量。算法作者还提出了基于 KCF（Kernelized Correlation Filters，内核化相关滤波器）和基于 DCF（Dual Correlation Filter，双重相关滤波器）的两种跟踪算法，并在 50 个标准视频[4]上进行了测试，并与其他优秀算法做出对比，发现其性能甚至超过 TLD、Struck 等一流跟踪算法，有着高帧率和高精度的特点，而且实施所需求的代码少。图 2-3 展示了测试结果。

作者利用 Matlab 实现了算法并公开了 Matlab 源码。算法在特征提取上提供 HOG 特征和灰度特征，在滤波器选择上提供 KCF 和 DCF 以及线性滤波器。本文在进行图像处理时，首先将图像灰度化，然后以 HOG 特征作为目标图像的特征，使用 KCF 滤波器，利用托管 C++语言及 OpenCV 库函数实现了算法，并将其使用在自动跟踪监控系统中，作为视觉跟踪程序。

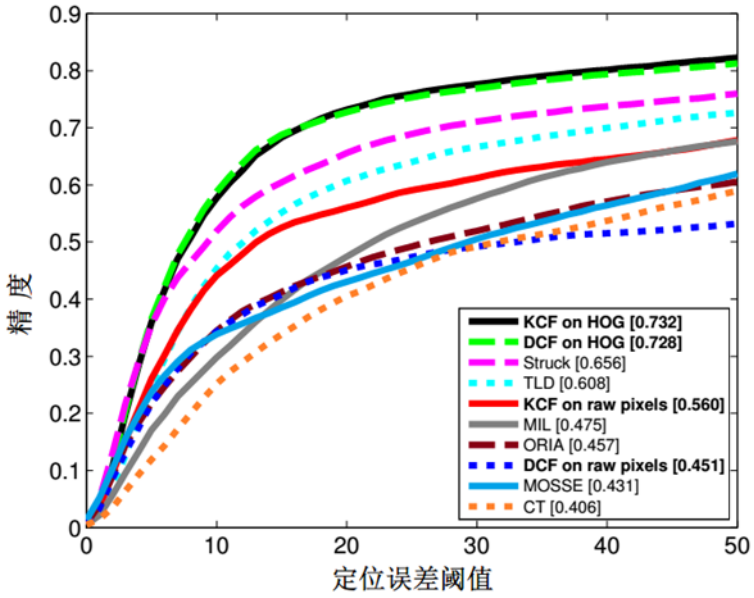


图 2-3 KCF 算法及 DCF 算法测试结果

2.4 PTZ 摄像机控制

本文所使用的 PTZ 摄像机安装有网络通信模块，其云台和摄像机的参数由内置的 CGI（Common Gateway Interface，通用网关接口）程序控制。通过向 PTZ 摄像机发送特定的 HTTP 请求，可以调用这些 CGI 程序，来达到控制 PTZ 摄像机云台参数的目的。

访问 PTZ 摄像机需要其在局域网中的 IP 地址、端口及登录名、密码。这里假设其 IP 地址为 ip，端口为 port，登录名为 john，密码为 smith。表 2-1 列举出了部分常用 PTZ 摄像机控制的 HTTP 请求语句。

表 2-1 部分 PTZ 摄像机控制的 HTTP 请求语句

实现功能	请求语句
登陆	http://ip:port/login.cgi?user=john&pwd=smith&pri=299
获取实时截图	http://ip:port/snapshot.cgi?user=john&pwd=smith
获取 JPG 图片流	http://ip:port/videostream.cgi?user=john&pwd=smith&.mjpg
云台向上旋转	http://ip:port/decoder_control.cgi?command=0&onestep=0
停止云台向上旋转	http://ip:port/decoder_control.cgi?command=1&onestep=0
云台向下旋转	http://ip:port/decoder_control.cgi?command=2&onestep=0
停止云台向下旋转	http://ip:port/decoder_control.cgi?command=3&onestep=0
云台向左旋转	http://ip:port/decoder_control.cgi?command=4&onestep=0
停止云台向左旋转	http://ip:port/decoder_control.cgi?command=5&onestep=0
云台向右旋转	http://ip:port/decoder_control.cgi?command=6&onestep=0
停止云台向右旋转	http://ip:port/decoder_control.cgi?command=7&onestep=0

2.5 本章小结

本章分析了 PTZ 摄像机自动跟踪监控系统的总体架构，介绍了各个部分之间的数据传

递关系。接着本章对完成整个系统所需要的各种工具也进行了简要的介绍，包括软件开发环境和使用的计算机语言、各种库函数。然后本章简单介绍了本文视觉跟踪程序所使用的KCF 算法，分析了其相较于其他算法所具备的优点。最后本文分析了 PTZ 摄像机控制程序的原理，列举出了控制 PTZ 摄像机所需要的部分 CGI 请求语句。

第3章 视觉跟踪程序设计

本章详细介绍了本文所使用的视觉跟踪程序，对其在一个图像处理周期中所做的工作进行了分析，并利用实时视频进行实验，客观分析了实验结果。

3.1 图像处理

3.1.1 图像读取

在 C++编写的视觉跟踪程序中，本文使用 OpenCV 中的 VideoCapture 类来实现 PTZ 摄像机实时画面的读取。向 VideoCapture 类中的 open 方法传入表 2-1 中的“获取 JPG 图片流”命令，再用 read 方法将实时图像以 JPG 图片的方式输出至一个 Mat 类型的变量中。为这个 Mat 变量取名为 frame，frame 中存储的即是当前 PTZ 摄像机所获取到的实时图像。随着程序的运行，frame 将在一个循环中被不断更新，以获取最新的实时图像。通过 OpenCV 的 imshow 函数，可以将 frame 在屏幕上显示出来。Mat 变量以行优先（row-major order）的方式来存储图像。

3.1.2 目标参数获取

用户通过图形界面来输入跟踪目标的参数。在程序得到目标参数前，图形界面仅仅显示摄像机的实时画面。用户使用鼠标框选住跟踪目标，此过程中，程序首先会记录鼠标左键按下时，鼠标所在位置的坐标(x_{m2}, y_{m2})，并在鼠标左键松开时记录下坐标(x_{m1}, y_{m1})。根据这两对坐标，可得到跟踪目标的初始参数：目标中心坐标(x_0, y_0)和目标尺寸$l_w \times l_h$。

$$(x_0, y_0) = \frac{(x_{m1}, y_{m1}) + (x_{m2}, y_{m2})}{2} \tag{3-1}$$

$$(l_w, l_h) = \left|(x_{m1}, y_{m1}) - (x_{m2}, y_{m2})\right| \tag{3-2}$$

3.1.3 目标图像提取

利用 3.1.2 得到的目标初始参数，继续提取目标的图像。在这之前需要得到灰度化的图像，而 frame 中存储的是 RGB 彩色图像。在 Matlab 中可以通过 rbg2gray 函数来完成图像灰度化。OpenCV 中则可以用 cvtColor 函数，具体调用语句：

cvtColor(frame, frame_gray, COLOR_BGR2GRAY);

其中 frame 为三通道矩阵，三个彩色通道按 B（蓝）、G（绿）、R（红）的顺序储存，frame_gray 为灰度化后的原始图像，为单通道矩阵，矩阵中元素的值为 0 至 255 的整数。COLOR_BGR2GRAY 为函数标记。

完成图像的灰度化之后，开始提取目标图像。根据 3.1.2 的结果，目标图像的四个角在 frame_gray 中的坐标应当是$(x_0 \pm \frac{l_w}{2}, y_0 \pm \frac{l_h}{2})$，但为了跟踪的准确性，本文实际提取的图像是用户提供目标图像尺寸大小的 2.5 倍，所以左上角坐标应为$(x_0 \pm \frac{5l_w}{4}, y_0 \pm \frac{5l_h}{4})$。根据

这些坐标，从 frame_gray 中复制对应的像素点到一个新的 Mat 型矩阵 window 中，这个尺寸为$\frac{5l_w}{4} \times \frac{5l_h}{4}$的单通道矩阵即为目标图像。将其宽度记作$l_{ww}$，高度记作$l_{wh}$。

在后面的操作中，目标的中心坐标会发生一定的变化，在短时间内可能会出现目标边界超出图像边界的情况，如图 3-1 所示。此时要利用目标边界的像素对缺失的像素部分进行填补。图 3-2 中目标图像的上方和下方出现的部分重复像素行，即是边界像素填补的结果。

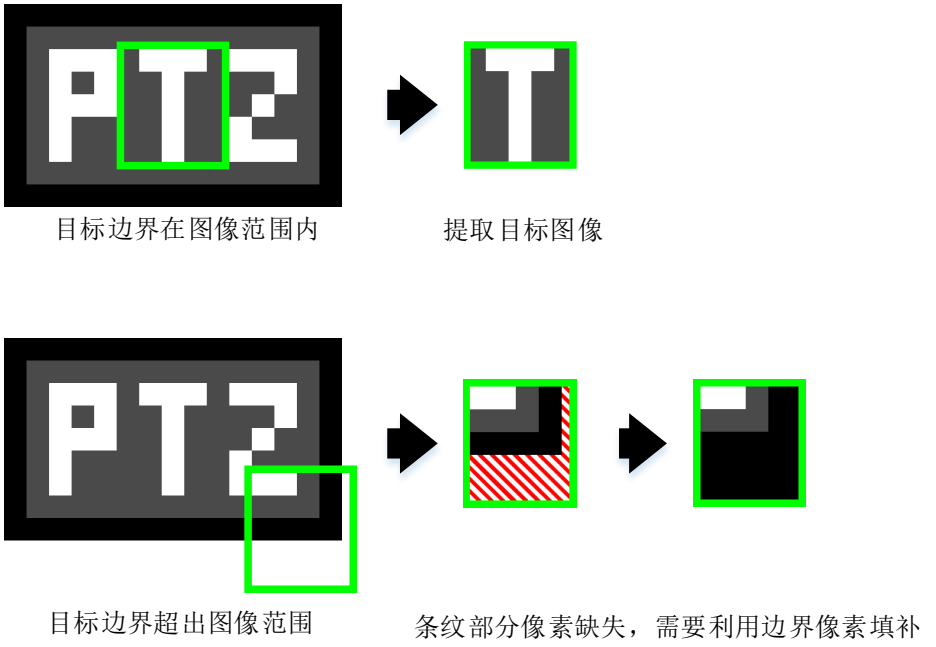


图 3-1 目标图像提取示意图

3.2 特征提取

在文献中，作者分别采用了 HOG 特征和灰度特征进行了仿真测试，采用 HOG 特征虽然增加计算量，使算法的实时性下降，但对于算法跟踪精度的提升很高。本文采用 HOG 特征作为图像特征。

本文使用 Piotr's Computer Vision Matlab Toolbox 库函数中的 HOG 函数来提取 HOG 特征，该函数虽然适用于 Matlab 平台，但其源码公开且核心部分是由 C/C++语言写成的，对传入参数和输出结果稍作处理即可用于 C++中。

取胞元（cell）大小为 4，梯度方向数为 9，对矩阵 window 提取 HOG 特征。得到其特征矩阵 feature，是一个三维矩阵，其大小为$l_{fw} \times l_{fh} \times 36$，其中$l_{fw} = 1/4\, l_{ww}$，$l_{fh} = 1/4\, l_{wh}$，且均取整。图 3-2 中展示了 HOG 特征提取的可视化结果。

为了进一步增加算法的跟踪精度，还要对 feature 进行一定的处理。将 feature 视作 36 个二维矩阵的集合，即$\{\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \cdots, \mathbf{A}_i, \cdots, \mathbf{A}_{36}\}$，同时记$\{\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \cdots, \mathbf{B}_i, \cdots, \mathbf{B}_{36}\}$为处理后的矩阵，符号“∘”代表逐点乘积（阿达马乘积）运算符。

$$\mathbf{B}_i = \mathbf{A}_i \circ \mathbf{M} \tag{3-3}$$

其中$\mathbf{M}$为：

$$\mathbf{M} = \mathrm{Hann}\left(l_{wh}\right) \cdot \mathrm{Hann}\left(l_{ww}\right)' \tag{3-4}$$

$\mathrm{Hann}(n)$为汉宁窗函数[14]，函数产生一个长度为 n 的列向量，是一个升余弦窗口。$\mathbf{M}$可以看作是一个扩展到二维平面的升余弦窗口，其尺寸为$l_{fw} \times l_{fh}$，与$\mathbf{A}_i$一致，这保证了它们能完成逐点乘法。$\mathrm{Hann}(n)$的表达式如下：

$$\mathrm{Hann}\left(n\right) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \tag{3-5}$$

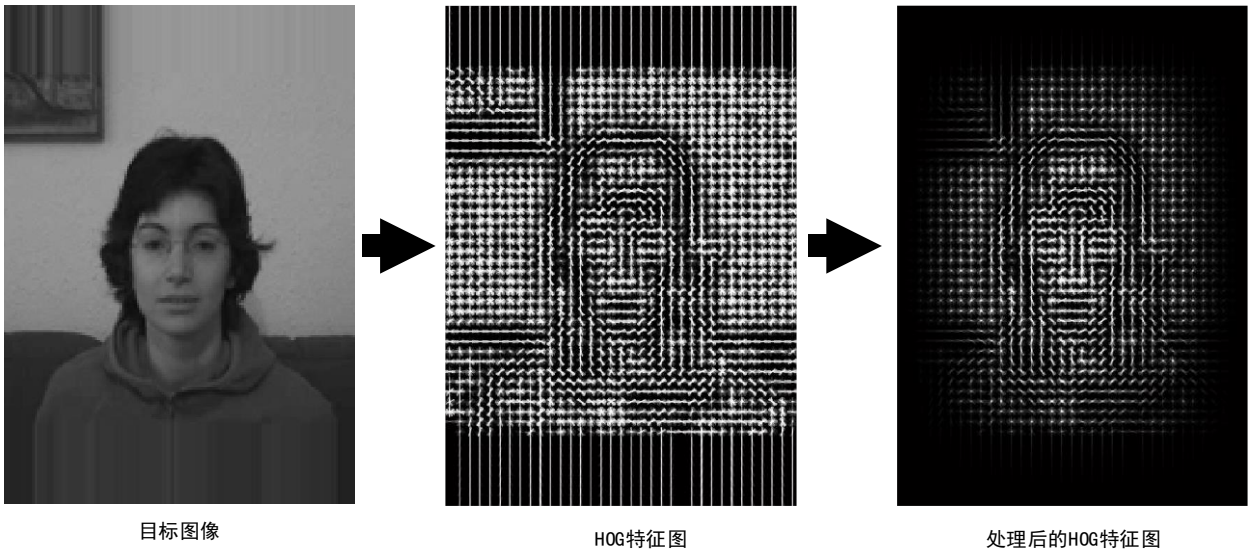


图 3-2 HOG 特征提取（测试图像来自文献[4]）

图 3-2 中展示了 HOG 特征矩阵在与一个二维升余弦窗口相乘之后的可视化结果，可以发现其边缘的特征被淡化了，相对的中心部分的特征得到强化。

经过处理后的 HOG 特征仍存储在 feature 矩阵中。3.3 中将在频域操作 HOG 特征，所以预先对 feature 作傅立叶变换。需要注意的是，feature 虽然是一个三维矩阵，但这只是它的存储方式，并不意味着需要对它作三维傅立叶变换，实际上它的第三维是独立的，可以看作 36 个二维矩阵。在作傅立叶变换的时候，仅需要对这 36 个二维矩阵分别作二维傅立叶变换。

在实施过程中，Matlab 中使用 fft2 函数，直接传入 feature 即可完成这一操作，得到一个尺寸与原矩阵相同，但数据类型为 complex double 的新矩阵，这是因为结果元素为复数。OpenCV 中可以使用自带的 dft 函数，也可以使用 FFTW 库函数替代，使用时需要对 feature 作一些结构上调整。得到的结果是一个与原矩阵相同，但为双通道的矩阵，结果元素同样

为复数。记 feature 的傅立叶变换结果为 xf（字母“ f ”表示其位于傅立叶域）。

3.3 核心化相关滤波器

本小节分析算法处理初始帧的情况，在 3.4 中分析后续帧。

3.3.1 回归标记

在作相关计算前，首先需要构造回归标记（labels）。为这些标记创建一个$l_{fw} \times l_{fh}$的单通道矩阵。矩阵的每一行都代表目标的一种偏移情况，整个 labels 矩阵就包含了目标（或者说目标特征，因为 labels 的尺寸与 feature 的尺寸一致）所有的偏移情况。同时 labels 还应是高斯型（Gaussian-shaped）的，这有利于减轻傅立叶域中的振铃效应。图 3-3 中以 2D 和 3D 图像的方式展示了 labels，测试图像与图 3-2 相同。构造 labels 的具体代码见附录。

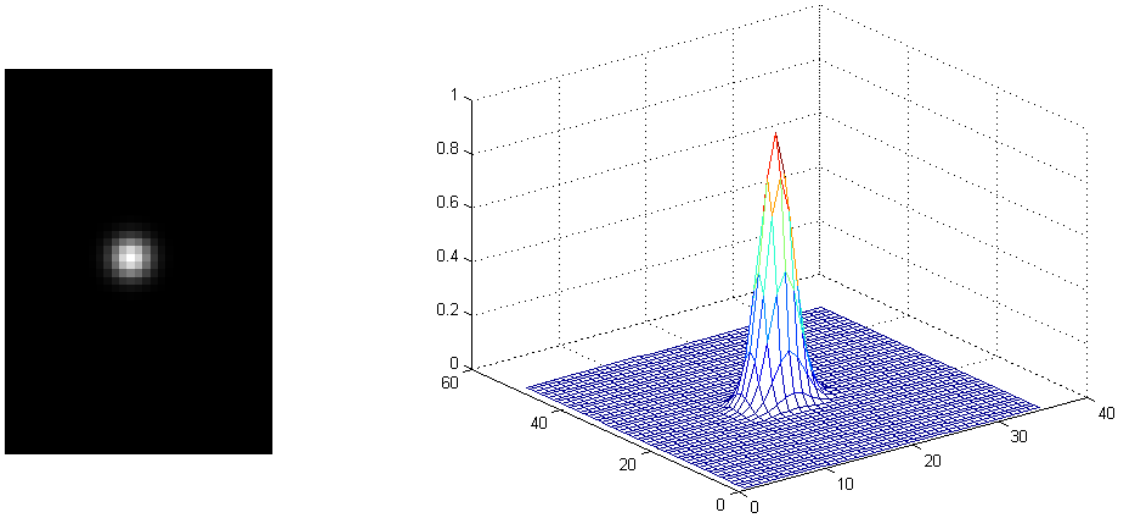


图 3-3 回归标记

这样构造的回归标记，其峰值位于中心。对 labels 矩阵作简单的循环移位，使中心平移并分布于四个角，左上角为原峰值。见图 3-4。这样做是为了避免相关计算结果中出现的不必要的移位。

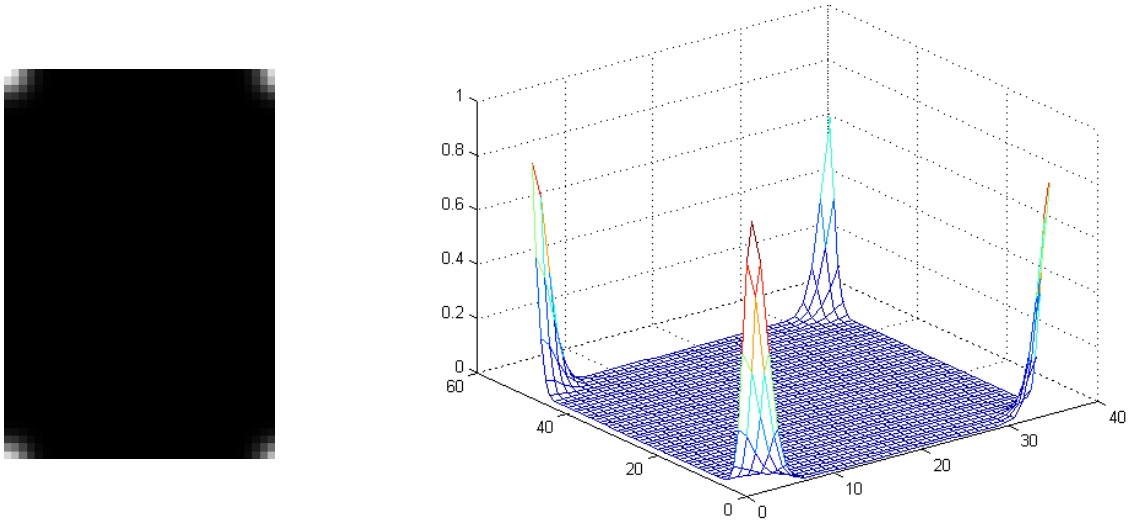


图 3-4 中心移位后的回归标记

对循环移位后的 labels 作傅立叶变换，得到矩阵 yf。

3.3.2 相关计算

对 xf 作高斯内核相关计算。使用到径向基函数（Radial Basis Function，RBF）：

$$\kappa\left(\mathbf{X},\mathbf{X}'\right)=k_i^{\mathbf{XX}'}=h\left(\left\|\mathbf{X}-\mathbf{X}'\right\|^2\right) \tag{3-6}$$

其中$\mathbf{X},\mathbf{X}'$是两个输入变量，$k_i^{\mathbf{XX}'}$是输出结果的元素，h 是某种函数。文献[9]中在式(3-6)的基础上，使用了高斯内核，并使函数能够处理三维（原文中为“multiple channels”，即“多通道”。这里称作“三维”是为了避免与 OpenCV 中矩阵通道的概念混淆。xf 是三维矩阵，若将其看作是多个二维矩阵，则也可认为是一个多通道二维矩阵，但本文仅在某一矩阵元素为复数时，使用“双通道”这一词修饰，并不提及“多通道”）输入变量。其中$\mathbf{X}=[\mathbf{X_1},\cdots,\mathbf{X_c}]$。上标“^”代表变量位于傅立叶域，$\sigma=0.5$。

$$\mathbf{k}^{\mathbf{XX}'}=\exp\left(-\frac{1}{\sigma^2}\left(\left\|\mathbf{X}\right\|^2+\left\|\mathbf{X}'\right\|^2-2\mathcal{F}^{-1}\left(\sum_c\hat{\mathbf{X}}_c^*\circ\hat{\mathbf{X}}_c'\right)\right)\right) \tag{3-7}$$

处理初始帧时，两个输入变量$\mathbf{X},\mathbf{X}'$均为 xf。计算结果为 kf，是一个尺寸为$l_{fw}\times l_{fh}$的二维双通道矩阵。

3.3.3 快速训练

利用如下模型进行快速训练：

$$\hat{\boldsymbol{\alpha}}=\frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{XX}}+\lambda} \tag{3-8}$$

$\hat{\mathbf{k}}^{\mathbf{XX}}$是 kf 作傅立叶变换后的结果，$\hat{\mathbf{y}}$即为 yf，$\lambda$是常数，这里取$\lambda=1\times10^{-4}$。除法为逐元素除法。结果$\hat{\boldsymbol{\alpha}}$存放于矩阵 alphaf 中。因为处理的是初始帧，所以用 alphaf 和 xf 来作检测时的模型。新建两个矩阵 model_alphaf 和 model_xf，它们的值分别为 alphaf 和 xf。

至此，算法对初始帧的处理完成。3.4 中将继续分析算法后续的循环过程。

3.4 目标参数更新

同初始帧的处理一样，算法在循环的开始也要提取特征。重复 3.2 的工作，将特征矩阵的傅立叶变换结果保存在 zf 矩阵中。接着以 zf 和 model_xf 为输入，按式(3-7)计算相关，得到$\hat{\mathbf{k}}^{\mathbf{XZ}}$，储存于矩阵 kzf。

3.4.1 快速检测

通过快速检测来对各个循环移位后的目标特征矩阵与原特征矩阵间的相关程度做出评价，其结果矩阵$\mathbf{f(z)}$储存于 response 中。为方便计算，在傅立叶域求解$\mathbf{f(z)}$：

$$\hat{f}(z) = \hat{k}^{xz} \circ \hat{\alpha} \tag{3-9}$$

对$\hat{f}(z)$求傅立叶反变换即可得到$f(z)$。另外 response 中仅保留$\hat{f}(z)$的实部，它的尺寸为$l_{fw} \times l_{fh}$，单通道。response 中的峰值点位置就是目标在这一帧的预测位置，但由于 3.3.1 中所做的移位操作，response 的峰值点坐标还需要做一定的转换才能得到目标预测坐标。

3.4.2 参数更新

得到了新的目标坐标后，提取特征，得到 xf；利用 xf 进行相关计算，得到 kf；对 kf 进行快速训练，得到 alphaf……进入循环，重复之前工作。但后续帧中 model_alphaf 和 model_xf 的值受前一帧影响：

model_alphaf = (1 - interp_factor) * model_alphaf + interp_factor * alphaf;

model_xf = (1 - interp_factor) * model_xf + interp_factor * xf;

其中 interp_factor 是常数，这里取 interp_factor = 0.02。

3.5 实验结果分析

2.3 小节中，展示了 KCF 算法的仿真结果。本节将分析 C++实现的跟踪程序在实时视频下的测试结果。

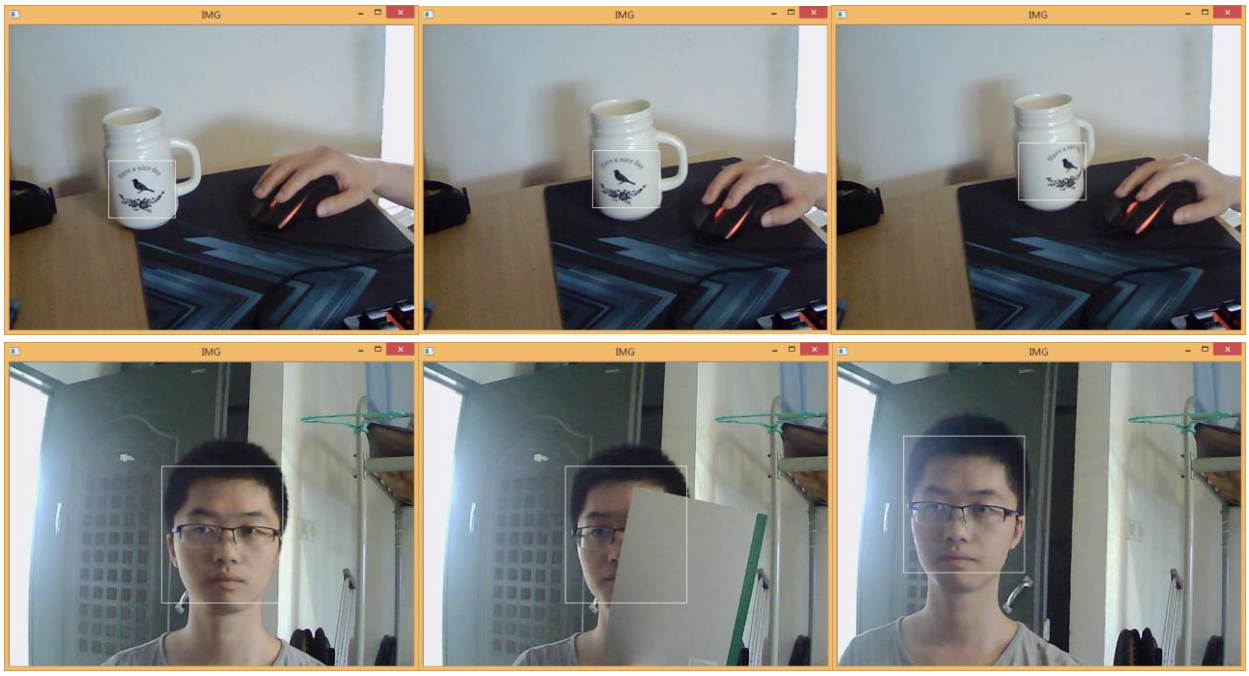


图 3-5 视觉跟踪程序实时测试结果

图 3-5 展示了视觉跟踪程序在测试中对一个水杯上的图案（见左上图）进行实时跟踪的结果。跟踪过程中水杯稍微转动（见右上图），图案发生了轻微的变形，但从图中可以看到，白色矩形方框仍然准确地标记出了目标图案的所在位置。

本文还针对人脸（见图 3-5 中下方三幅图）、物体高速运动、目标遮挡等几种情况进行了跟踪测试，跟踪程序在测试中表现出了较好的准确性，目标被遮挡或目标变形时，跟

踪程序也能准确地进行跟踪。另外程序实时性较好，在普通性能的电脑上（Inter(R) Core(TM) i5-2410M CPU @2.3GHz， 8GB DDR3 RAM），能以每秒 15 帧以上的速度处理实时视频。

但程序还存在一些不足，当跟踪高速运动的物体，或者运动物体突然加速、方向突变时，就会出现目标丢失或者错误跟踪的情况。

3.6 本章小结

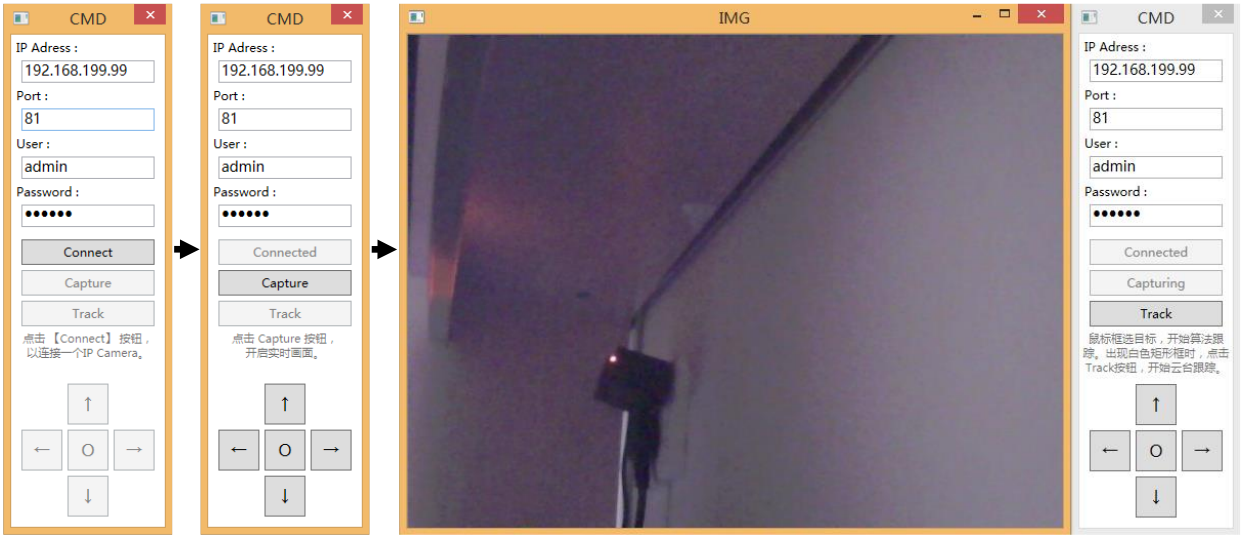
本章将视觉跟踪程序中所使用的 KCF 算法利用托管 C++及 OpenCV 库函数实现，并对其在图像处理中一个处理周期的工作过程（构造回归标记、相关计算、模板快速训练、快速检测、参数更新）做出了详尽的分析。利用实时视频对程序进行实验检测，并客观分析了实验的结果，得出其性能基本满足实时跟踪的需求，并提出了一些不足之处。

第4章 PTZ 摄像机控制程序设计

本章详细介绍了本文所使用的 PTZ 摄像机控制程序，首先展示了本文所提出系统的图形用户界面程序；接着介绍 PTZ 摄像机控制算法，算法以伪代码的形式给出；最后对 PTZ 摄像机控制程序进行了实验测试。

4.1 图形用户界面

利用 WPF 构建系统的 UI 程序：



初始界面　点击“Connect”按钮后　点击“Capture”按钮后

图 4-1 PTZ 自动跟踪监控系统 UI 程序

如图 4-1 所示，PTZ 自动跟踪监控系统 UI 程序为一个长条形窗口，窗口从上到下排布的分别是 4 个 PTZ 摄像机信息输入框、3 个功能按钮、1 个信息提示框和 1 组云台手动控制按钮。

4 个 IP 摄像机信息输入框分别记录 PTZ 摄像机在局域网中的 IP 地址、及端口号，PTZ 摄像机登陆用户名、密码。这些信息由用户输入。在用户单击“Connect”按钮后，程序收集 4 个信息输入框中的字符串，拼接成 http 请求语句，并向 PTZ 摄像机发起通信请求，即发送表 2-1 中的“登陆”请求。登陆成功后程序会得到一个特定返回值，登陆失败则会报错，提示重新登陆。

程序和 PTZ 摄像机之间的通信成功后，UI 程序中的其他功能按钮被激活，由浅色变为深色。用户可以通过界面下方的云台控制按钮手动控制云台转动，如，按住标有“↓”的按钮时，云台会向俯角方向旋转；按下标有“〇”的按钮时，云台会复位到初始位置。单击“Capture”按钮，程序则会向 PTZ 摄像机请求得到实时画面，即发送表 2-1 中的“请求 JPG 图片流”请求。请求成功后，程序会打开一个新窗口，用以显示 PTZ 摄像机所拍摄的实时画面。如图 4-1 右侧画面所示。

在得到 PTZ 摄像机的实时画面后，用户可以用鼠标在画面中框选跟踪目标，程序根据用户的操作来获取跟踪目标的初始参数，这一过程在 3.1.2 小节中已经做了详细的阐述。

值得注意的是，在单击“Track”按钮之前，程序仅对目标做算法跟踪，云台并不会自动跟踪目标。

单击“Track”按钮之后，程序将为 PTZ 摄像机控制程序开辟一个新的线程，根据视觉跟踪程序所得到的目标坐标，来控制云台的运动。

4.2 PTZ 摄像机控制

4.2.1 跟踪程序调用

在 2.2 节中已经提到，本文的视觉跟踪程序是由托管 C++编写，PTZ 摄像机控制程序由 C#编写。其中视觉跟踪程序分为两部分，第一部分源代码，这部分代码与普遍 C++编写的代码差别很小，仅仅区别在托管 C++中的一些变量和函数是托管类型。视觉跟踪程序的第二部分是一个托管类，第一部分中的托管类型变量和函数需要在这个类中被包装起来，以便供 C#程序调用。将视觉跟踪程序生成为 DLL 文件，并在 C#程序中添加对该 DLL 文件的引用，就能在 C#中像调用其他 C#类一样，调用托管类中的变量和函数了。

4.2.2 摄像机控制

本小节以 C#风格的伪代码的形式来描述和分析摄像机云台的控制算法：

```
Point pos = new Point(); //声明 Point 类型变量，用以存储目标当前坐标。
bool isMoving = false; //布尔型变量，云台运动时，该变量取真值。
while(true)  //摄像机控制程序与主程序处于不同线程，所以这里的循环不会影响主程序。
{
    pos = tracker.getPos();//从跟踪程序处得到当前目标坐标。

    //center 为屏幕中心坐标。threshold 为一个阈值，分 x 方向和 y 方向。
    //下面分别是目标坐标在上、下、左、右四个方向上超过一定阈值时：
    if ((center.Y - pos.Y)>threshold.Y) and isMoving is false)
    {
        PTZ.Up(); //云台向上旋转。
        isMoving = true;
    }
    if ((pos.Y – center.Y)>threshold.Y) and isMoving is false)
    {
        PTZ.Down(); //云台向下旋转。
        isMoving = true;
    }
    if ((center.X - pos.X)>threshold.X) and isMoving is false)
    {
```

```
            PTZ.Left(); //云台向左旋转。
            isMoving = true;
        }
        if ((pos.X – center.X) > threshold.X) and isMoving is false)
        {
            PTZ.Right(); //云台向右旋转。
            isMoving = true;
        }

        //当云台转动，使目标坐标在阈值范围内时：
        if (isMoving is true and abs(pos – center)<=threshold)
        {
            PTZ.Stop(); //云台停止旋转。
            isMoving = false;
        }
        Sleep(T); //线程暂停 T 毫秒。
    }
```

经过多次测试后发现，threshold.X 取 60，threshold.Y 取 80，T 取 10 的时候，程序取得较好的控制结果。

4.3 实验结果分析

为了单独测试 PTZ 摄像机控制程序，让其不受视觉跟踪程序结果的影响，本节用鼠标指针在实时画面窗口中的坐标来代替视觉跟踪程序的跟踪结果，以在实时画面窗口中移动鼠标的方式来模拟跟踪目标的运动，观察云台运动的情况，从而达到评估 PTZ 摄像机控制程序性能的目的。如图 4-2 所示，画面中的红色圆圈标记出了鼠标所在的位置，也就是测试中的目标坐标的位置。可以看到因鼠标坐标的变化，云台作出了旋转，引起了画面的变动。

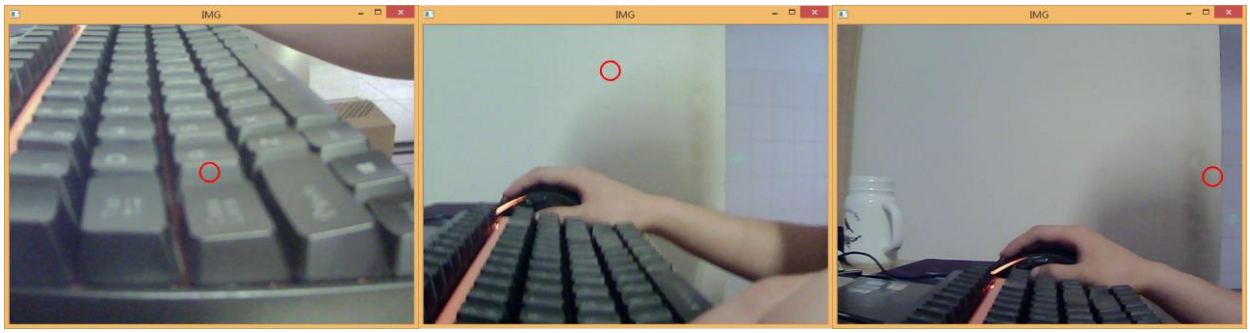


图 4-2 PTZ 摄像机控制程序根据鼠标坐标控制云台运动

实验中 PTZ 摄像机表现出了良好的准确性与实时性。但实验所用的 PTZ 摄像机的云台存在一个问题：在进行左、右方向旋转时，云台先向欲旋转方向相反的方向旋转一个很小的角度，引起图像突然抖动。在 PTZ 摄像机控制程序和视觉跟踪程序相结合时，这可能会影响到视觉跟踪的结果，从而影响整个系统的准确性。

4.4 本章小结

本章展示了利用 WPF 编写的 PTZ 摄像机控制程序的图形用户界面程序。该 UI 程序提供自动跟踪监控系统中各种功能的入口。本章还列出了实现的 PTZ 摄像机控制的算法的伪代码，将 PTZ 摄像机控制程序利用 C#语言实现，并将鼠标的屏幕坐标作为目标坐标，实验测试程序控制云台跟踪目标的结果。最后对结果进行了客观分析，提出了几点不足。

第5章 实验结果与分析

本章结合第 3 章和第 4 章的工作，对整个 PTZ 自动跟踪监控系统进行了实验测试，给出了实验结果，并对结果进行了分析。

5.1 实验结果

本节是对 3.5 节和 4.3 节所做工作的完善与补充。

将视觉跟踪程序和 PTZ 摄像机控制程序，以及 UI 程序结合在一起，组成了完整的 PTZ 摄像机自动跟踪监控系统。图 5-1 展示了部分测试结果，从图中的背景变化和白色矩形方框位置变化，可以看出系统不仅对跟踪目标易拉罐（上部分三幅图）和人脸（下部分三幅图）做出了准确的视觉算法跟踪，其 PTZ 摄像机控制程序也通过控制云台运动，使目标始终保持在摄像机视野中央部分。图 5-2 展示了目标在发生变形、平面旋转和被部分遮挡情况下的目标跟踪情况。从图中可以看出，在目标发生一定的变形时，系统能对目标做出跟踪，但是在跟踪一段时间后，结果误差逐渐变大，准确性开始降低。在目标平面旋转时，同样会出现这种情况。当目标被部分遮挡时，跟踪结果并不会受到太大的影响。图 5-2 中左上角和右下角的测试图像的颜色与其他图像有一定差异，这是因为测试时光线较暗，测试所用的 PTZ 摄像机自动开启了红外照明功能，造成了一定程度的光线变化。系统的跟踪结果并没有明显受到光线变化的干扰，这说明系统的跟踪结果对一定程度的光线变化不敏感。

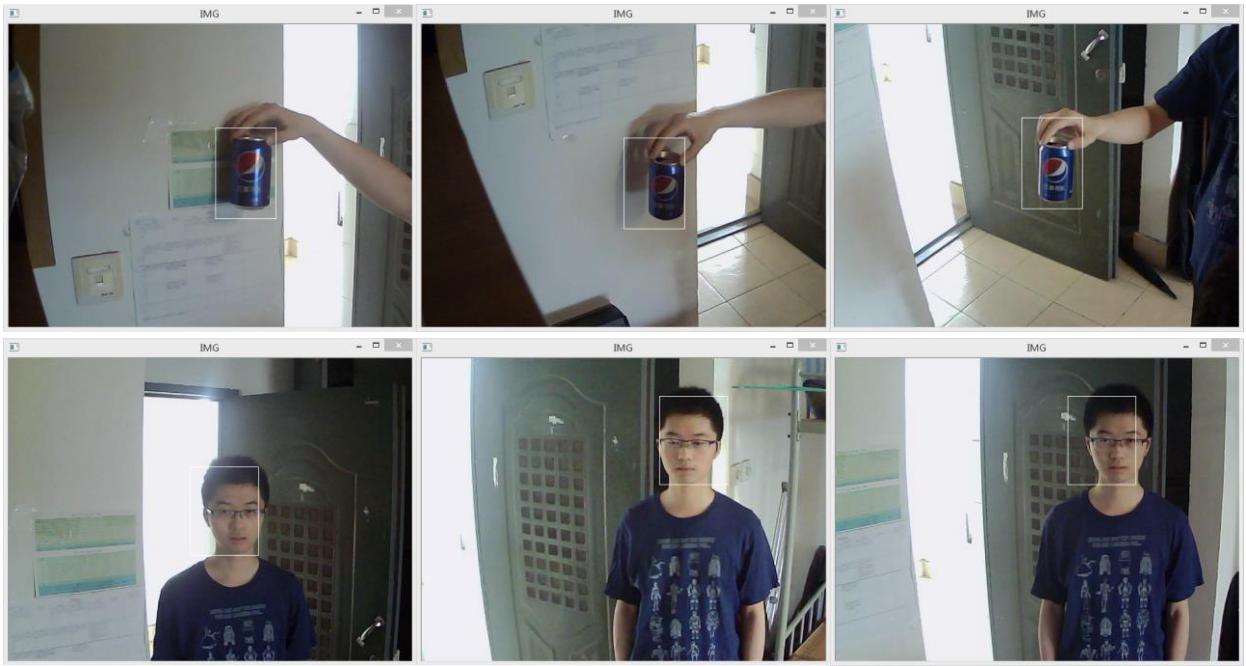


图 5-1 PTZ 摄像机自动跟踪监控系统测试

整个系统表现出了较好的准确性与实时性。

5.2 结果分析

在 3.5 节和 4.3 节中，本文已经通过实验发现了视觉跟踪程序和 PTZ 摄像机控制程序

中存在的部分缺点与不足，并给出了解释。本节从整个系统的角度来分析实验结果。

目标变形

目标平面旋转

目标被遮挡

图 5-2 目标变形、平面旋转、被遮挡情况下的跟踪情况

首先是系统的实时性，PTZ 摄像机所获取的视频画面为每秒 25 帧，在进行跟踪时，图像的处理速度会根据跟踪目标的尺寸大小，在每秒 15 帧至 25 帧之间浮动。虽然能够达到实时性的要求，但是其处理速度仍然有提升的空间。在用托管 C++及 OpenCV 库函数实现视觉跟踪算法的过程中，本文对三维矩阵的处理效率一直很低。算法原作者[9]使用 Matlab 处理这些三维矩阵，Matlab 中的很多函数都有针对高维矩阵的重载，基本的加、减、乘、除、逐元素乘、逐元素除运算符也有对应的重载。但是在 OpenCV 中，针对 Mat 类型的矩阵并不像 Matlab 中那样有着丰富重载。例如 3.2 节中提到的，对一个三维矩阵做二维傅立叶变换，实际上是将它当作 36 个二维矩阵的集合，对每一个二维矩阵做二维傅立叶变换。这在 Matlab 中仅需要调用 fft2 函数就能完成。但针对这个问题，本文在实施代码的过程中，首先尝试的是将所有三维矩阵用二维矩阵向量组的方式来代替，即 vector<Mat>类型。使用这种类型是考虑到算法中可能出现的动态大小的矩阵（实际上并没有）。在对其中某一个具体矩阵进行操作时，通过向量组索引来找到矩阵。但实验证明，这种方法效率低下，算法将花费大量时间在建立矩阵向量组变量和索引上。接着本文尝试将三维矩阵存放在一个 dims 值为 3 的 Mat 类型矩阵中，并用指针和内存操作来取得其中的某个二维矩阵。这种方法高效的多，本文最终也采用了这种方法。后期改进的工作中，可以尝试利用 Mat 数组来代替三维矩阵，然后针对 Mat 变量和 Mat 数组变量，为各个功能函数（如离散傅立叶

变换）添加重载。

本文为 PTZ 摄像机控制程序的云台控制部分开辟了一个独立的线程，实际上在视觉跟踪程序中也可以采用多线程操作来提高程序运行速度，以优化系统的实时性。但这需要对算法的结构进行较大的改动。

其次是系统的准确性。PTZ 摄像机采集到的视频画面尺寸为640×480，画面中含有较多的噪点。考虑到采用 HOG 作为特征，噪点的影响不大。HOG 特征取自灰度图像，针对这一点，可以做出两种优化：一是增强图像对比度，以增强 HOG 特征；二是将彩色图像的三个通道分离，分别提取 HOG 特征，然后取其中特征最明显的，或者按三者加权的方式来得到最终的特征。这会增加算法的计算量，需要牺牲一定的实时性。

4.3 节中提到的 PTZ 摄像机云台旋转中出现的问题也会影响跟踪准确性，可以通过更换更高性能的 PTZ 摄像机来解决，但这会增加系统的成本。

最后是系统的鲁棒性。系统的鲁棒性很大程度上由视觉跟踪程序的鲁棒性决定。本文对于跟踪结果做了一个简单的判断，来增强系统的鲁棒性：当当前目标坐标与上一帧的目标坐标距离在一定阈值内时，认为跟踪程序跟踪正常；超过阈值时，则认为跟踪出错，取上一帧的目标坐标作为跟踪结果。这减少了图像抖动及目标突然移动所带来的误差，但效果并不是非常明显。为了增强系统的鲁棒性，后期工作中还需要采取其他措施。

5.3 本章小结

本章对完整的 PTZ 自动跟踪监控系统做出了实验测试，并详尽分析了测试结果。在实时性、准确性、鲁棒性这三点上，分析并找出了系统的缺陷及造成这些缺陷的原因。这些缺陷主要出自于视觉跟踪程序在实施过程中出现的一些问题（如三维矩阵问题），以及算法本身存在的一些不足。本章还对改进系统的可行方法和策略进行了一定探讨。

第6章 总结与展望

6.1 总结

本文设计并实现了 PTZ 自动跟踪监控系统，系统能够对预先指定的目标进行实时跟踪，实时性和准确性较好。但系统的鲁棒性还有待提升，视觉跟踪程序的处理速度也存在提升的空间。

6.2 展望

PTZ 摄像机在一定程度上解决了监控系统中单一摄像机视野范围有限的问题，但要想继续扩展跟踪视野，避免不了使用多摄像机来进行跟踪。若使用多个 PTZ 摄像机对单一目标进行跟踪，跟踪的视野将得到极大的扩展，综合多个摄像机的跟踪结果，得到的目标位置的准确性也将得到提升。

另一方面，再考虑单摄像机跟踪能力的提升。可以通过改进视觉跟踪算法，实现单摄像机对多个运动目标实现跟踪。结合之前提到的，多摄像头跟踪单目标，可以实现多摄像头对多目标跟踪的监控系统。在此基础上能扩展出许多实用的功能，实现真正的智能监控。

而眼前的下一步工作，是继续优化视觉跟踪程序的执行效率，在三维矩阵处理上和内存操作上进行改进，尽量使用多线程处理来加速程序；继续完善系统，减少运行过程中的意外错误。

参考文献

[1] Horn B K, Schunck B G. Determining optical flow[C]. 1981 Technical Symposium East, 1981: 319-331.

[2] 侯志强, 韩崇昭. 视觉跟踪技术综述[J]. 自动化学报, 2006, 32(4): 603-617.

[3] 李谷全, 陈忠泽. 视觉跟踪技术研究现状及其展望[J]. 计算机应用研究, 2010, 27(8): 20-27.

[4] Wu Y, Lim J, Yang M-H. Online object tracking: A benchmark[C]. Computer vision and pattern recognition (CVPR), 2013 IEEE Conference on, 2013: 2411-2418.

[5] Jia X, Lu H, Yang M-H. Visual tracking via adaptive structural local sparse appearance model[C]. Computer vision and pattern recognition (CVPR), 2012 IEEE Conference on, 2012: 1822-1829.

[6] Kalal Z, Matas J, Mikolajczyk K. Pn learning: Bootstrapping binary classifiers by structural constraints[C]. Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, 2010: 49-56.

[7] Kalal Z, Mikolajczyk K, Matas J. Tracking-learning-detection[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2012, 34(7): 1409-1422.

[8] Hare S, Saffari A, Torr P H. Struck: Structured output tracking with kernels[C]. Computer Vision (ICCV), 2011 IEEE International Conference on, 2011: 263-270.

[9] Henriques J F, Caseiro R, Martins P, et al. High-speed tracking with kernelized correlation filters[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2015, 37(3): 583-596.

[10] MSDN: Microsoft Developer Network. C++ 托管扩展编程 [EB/OL]. http://msdn.microsoft. com/zh-cn/library/ms384255(v=vs.71).aspx.

[11] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. "O' Reilly Media, Inc.", 2008.

[12] Dollár P. Piotr's image and video Matlab Toolbox (PMT)[J]. Software available at: http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html, 2013.

[13] Frigo M, Johnson S G. FFTW: An adaptive software architecture for the FFT[C]. Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, 1998: 1381-1384.

[14] Wikipedia, The Free Encyclopedia. Hann function[EB/OL]. http://en.wikipedia.org/w/index.php?title=Hann_function&oldid=661699183.

致 谢

首先感谢陈莹导师在毕业设计期间提供的敦促与帮助。对于我不懂的问题，陈老师都及时为我指出了解决思路。为期一个学期的毕业设计，陈老师的指点让我受益良多。

感谢知乎用户坂本土豆子、莫涛以及 Stack Overflow 用户 Joe Chakra 对我的提问做出的回答。

感谢众多开源代码库的作者，他们的分享精神令我感到钦佩。

感谢江南大学物联网工程学院顾德老师在论文写作格式问题中提供的帮助。

附 录

跟踪功能的主函数

```
void cvFun::Tracker(Point2i pos, Size2i targetSize, double padding, double lambda, double outputSigmaFactor, double interpFactor, int cellSize, int orientations)
{
    //预先声明变量，申请内存空间
    Mat img, bgrImg;
    bool firstFrame = true;
    Point2i draw1, draw2;
    Mat patch, response, model_alphaf, alphaf, kf, xf, zf, kzf, model_xf;
    Point maxLoc = { 0, 0 };
    Point minLoc = { 0, 0 };
    Point2i temp;
    //patch 是否过大？如果过大，则作尺寸减半处理
    bool resizeImage = (sqrt(targetSize.width*targetSize.height) >= 100);
    if (resizeImage)
    {
        pos = pos / 2;
        targetSize = targetSize / 2;
    }
    //用来处理的图像要比 patch 稍微大一点。padding 作为放大系数
    Size2i windowsz;
    windowsz.height = targetSize.height * (1 + padding);
    windowsz.width = targetSize.width * (1 + padding);
    double outSigma = sqrt(targetSize.width*targetSize.height)*outputSigmaFactor / cellSize;
    Mat yf = cvFun::FFT(cvFun::GaussianShapedLabels(outSigma, windowsz / cellSize));
    //构造 cosine window
    Mat cosWindow = cvFun::Hann(yf.rows).t() * cvFun::Hann(yf.cols);

    while (1)//进入循环
    {
        if (!m_TrackerActive)
        {
            return;
        }
```

```
// 读取视频信息
m_VideoCap.read(bgrImg); // read a new frame from video
if (bgrImg.empty())
{
	cout << "End of Sequence" << endl;
	break;
}
if (resizeImage)
{
	Mat bgrimgClone = bgrImg.clone();
	Size sz;
	pyrDown(bgrImg, bgrimgClone, sz, BORDER_DEFAULT);
	cvtColor(bgrimgClone, img, COLOR_BGR2GRAY);
}
else cvtColor(bgrImg, img, COLOR_BGR2GRAY);
if (!firstFrame)
{
	patch = cvFun::GetSubwindow(img, pos, windowsz);
	zf = cvFun::FFT(cvFun::GetFeatures(patch, cellSize, orientations, cosWindow));
	kzf = cvFun::GaussianCorrelation(zf, model_xf, 0.5);//kernel_sigma = 0.5
	response = cvFun::IDFT(cvFun::ComplexMul(model_alphaf, kzf));
	double tempmin, tempmax;
	minMaxLoc(response, &tempmin, &tempmax, &minLoc, &maxLoc);
	if ((maxLoc.x + 1) > (zf.size[2] / 2))
		maxLoc.x = maxLoc.x - zf.size[2];
	if ((maxLoc.y + 1) > (zf.size[1] / 2))
		maxLoc.y = maxLoc.y - zf.size[1];
	temp = pos + cellSize * maxLoc;
	if (abs((temp - pos).x*(temp - pos).y) <= 20) pos = temp;

	if (resizeImage) m_Pos = 2 * pos;
	else m_Pos = pos;
}
patch = cvFun::GetSubwindow(img, pos, windowsz);
xf = cvFun::FFT(cvFun::GetFeatures(patch, cellSize, orientations, cosWindow));
```

```
            kf = cvFun::GaussianCorrelation(xf, xf, 0.5);//kernel_sigma = 0.5
            alphaf = yf / (kf + lambda);
            if (firstFrame)
            {
                model_alphaf = alphaf.clone();
                model_xf = xf.clone();
            }
            else
            {
                model_alphaf = (1 - interpFactor)*model_alphaf + interpFactor*alphaf;
                model_xf = (1 - interpFactor)*model_xf + interpFactor *xf;
            }
            draw1.x = pos.x - targetSize.width / 2;
            draw1.y = pos.y - targetSize.height / 2;
            draw2.x = pos.x + targetSize.width / 2;
            draw2.y = pos.y + targetSize.height / 2;
            if (resizeImage)
            {
                draw1 = draw1 * 2;
                draw2 = draw2 * 2;
            }
            rectangle(bgrImg, draw1, draw2, Scalar(255, 255, 255), 1.5);//白色矩形框
            imshow("IMG", bgrImg);//显示结果
            if (waitKey(30) == 27) //等待“ESC”30 毫秒
            {
                m_TrackerActive = false;
                cout << "esc key is pressed by user" << endl;
                break;
            }
            firstFrame = false;
        }
        return;
}
```

高斯型标记构造函数

```
Mat cvFun::GaussianShapedLabels(double sigma, Size2i sz)
{
    Mat rs, cs;
    vector<int> y(sz.height);
    vector<int> x(sz.width);
    int index_y = 1 - (int)sz.height / 2;
    int index_x = 1 - (int)sz.width / 2;
    for (int i = 0; i < sz.height; i++)
    {
        y[i] = index_y++;
    }
    for (int i = 0; i < sz.width; i++)
    {
        x[i] = index_x++;
    }
    Ndgrid(rs, cs, y, x);
    Mat labels;
    exp(-0.5 / pow(sigma, 2)*(rs.mul(rs) + cs.mul(cs)), labels);
    labels = CircShift2D(labels, y[0], x[0]);
    return labels;
}
```

高斯相关计算函数

```
Mat cvFun::GaussianCorrelation(Mat xf, Mat yf, double sigma)
{
    double xx = cvFun::Norm(xf);//计算范数
    double yy = cvFun::Norm(yf);
    Mat yf_conj = cvFun::Conj(yf);
    Mat xyf = cvFun::ComplexMul(xf, yf_conj);
    Mat ixyf = cvFun::IDFT(xyf);
    Mat xy = cvFun::Sum(ixyf);
    double pow_sigma = sigma*sigma;
    Mat kf = (xx + yy - 2 * xy) / xf.size[0] * xf.size[1] * xf.size[2];
    int n = kf.rows*kf.cols*kf.channels();
```

```
    double* p_kf = (double*)kf.data;
    for (int i = 0; i < n; i++)
    {
        if (p_kf[i] <= 0) p_kf[i] = 0;
        p_kf[i] = exp(-1 / (p_kf[i] * pow_sigma));
    }
    kf = cvFun::FFT(kf);
    return kf;
}
```

目标图像提取函数

```
Mat cvFun::GetSubwindow(Mat img, Point2i pos, Size2i sz)
{
    int h_temp = pos.x - sz.width / 2;
    int v_temp = pos.y - sz.height / 2;
    vector<int> h(sz.width);
    vector<int> v(sz.height);
    int i, j;
    for (i = 0; i < sz.width; i++)
    {
        if (h_temp < 0)       h[i] = 0;
        else if (h_temp >= img.rows)       h[i] = img.rows - 1;
        else       h[i] = h_temp;
        h_temp++;
    }
    for (i = 0; i < sz.height; i++)
    {
        if (v_temp < 0)       v[i] = 0;
        else if (v_temp >= img.rows)       v[i] = img.rows - 1;
        else       v[i] = v_temp;

        v_temp++;
    }
    Mat subwindow = Mat::zeros(sz, CV_8UC1);
    uchar *p_subwindow = (uchar*)subwindow.data;
```

```
    uchar *p_img = (uchar*)img.data;
    int index;
    for (i = 0; i < subwindow.rows; i++)
    {
        for (j = 0; j < subwindow.cols; j++)
        {
            index = v[i] * img.cols + h[j];
            p_subwindow[i*subwindow.cols + j] = p_img[index];
        }
    }
    return subwindow;
}
```

特征提取函数

```
Mat cvFun::GetFeatures(Mat img, int cellSize, int orientations, Mat cosWindow)
{
    int height = img.rows;
    int width = img.cols;
    int page = height*width;
    int size[] = { 36, width / cellSize, height / cellSize };//t
    Mat I = Mat(img.size(), CV_32FC1);
    Mat H = Mat(3, size, CV_32FC1);
    Mat So = Mat(3, size, CV_64FC1);
    uchar* p_img = img.data;
    float* p_I = (float*)I.data;
    float* p_H = (float*)H.data;
    float* p_O = (float*)malloc(page*sizeof(float));
    float* p_M = (float*)malloc(page*sizeof(float));
    double* p_So = (double*)So.data;
    for (int i = 0; i < page; i++)
    {
        p_I[i] = (float)p_img[i];
    }
    I = I.t() / 255;
    p_I = (float*)I.data;
```

```
    gradMag(p_I, p_M, p_O, height, width, 1, false);
    hog(p_M, p_O, p_H, height, width, cellSize, orientations, 1, false, 0.2);
    for (int i = 0; i < size[0] * size[1] * size[2]; i++)
    {
        p_So[i] = (double)p_H[i];
    }
    vector<Mat> x_real = cvFun::M2V(So);
    vector<Mat> x(x_real.size());
    for (int i = 0; i < x_real.size(); i++)
    {

        x_real[i] = x_real[i].t();
        x_real[i] = x_real[i].mul(cosWindow);

    }
    Mat x_m = cvFun::V2M(x_real);
    free(p_O);
    return x_m;
}
```