分类号 _____ 密级 _____

UDC _____ 编号 _____

中国科学院研究生院

硕士学位论文

三通道太阳磁场望远镜远程观测终端系统设计

申基

指导教师 胡柯良 高级工程师 博士 中科院国家天文台

申请学位级别 硕 士 学科专业名称 天文技术与方法

论文提交日期 2008.5 论文答辩日期 2008.5

培养单位 中国科学院国家天文台

学位授予单位 中国科学院研究生院

答辩委员会主席 杨志良

摘要

随着网络技术的发展，网络传输的可靠性和稳定性得到的大大的提高，又由于天文台选址的特殊性，一般都会建在离城市远的或者环境恶劣的地方，这就使得望远镜的远程观测成为将来的发展趋势。同时太阳是离我们最近的恒星，她的剧烈活动和周期变化直接或间接的影响着人类的生活，也是引起通讯和航天器故障的重要原因，因此对太阳的研究和对日地环境灾变的预报尤为重要，这就把对太阳的实时观测的需求提升到了一个新的高度。利用怀柔三通道太阳磁场望远镜对太阳进行多层次同步观测可以同时获得日面不同层次的活动图像，这对于更好的理解太阳物理有着重要意义。

本文基于怀柔三通道太阳磁场望远镜开发了在局域网内对三通道 CCD 进行同步观测的远程终端观测系统，主要工作和贡献有：（1）利用网络传输在局域网内实现了对三通道望远镜的远程控制，有效降低了观测成本；（2）对三通道太阳望远镜的三个 CCD 进行同步远程控制，同时获得日面不同层次的活动图像；（3）系统的框架设计采用模块化思想，将望远镜中的主要设备和资源模块化封装起来，当望远镜更新或者增加新的相机时，可以快速方便的更新终端系统，有效提高了开发望远镜终端系统的效率。同时利用其模块化的特点，可以将三通道望远镜远程终端的框架系统作为太阳望远镜终端系统的开发模板，快速开发出新的望远镜的终端系统。

系统设计采用 vc.net 集成开发环境，使用 TCP/IP 协议，通过套接字网络编程，实现对三通道望远镜的三个 CCD 进行同步远程控制。目前三通道望远镜的终端系统已经在局域网内实现了图像数据和相机控制命令的传输等远程观测功能，并取得了初步的观测结果。

关键字：三通道望远镜；同步观测；模块化；远程观测

Abstract

As the development of the networks, transferring information by internet has made a great progress on reliability and stability, in addition, the general astronomical observing station is usually built far away from the city, therefore it's inconvenient for observing because of long way, so remote control of the telescope is becoming more and more important. The sun is the nearest star to us, and her fierce and periodic activity influences our humanity's life directly or indirectly, and it's also the reason causes the breakdown of the ground communication and spacecraft, so it's becoming more and more important to make solar activity prediction in real time. Using Huairou three-channel solar magnetic field telescope to carry out multi-layer synchronous solar observation can obtain live images of different layers at the same time, which has important significance in solar physics.

Based on the Huairou three-channel solar magnetic field telescope, the article discusses the development of remote terminal Observing System to carry out the observation of three-channel telescope synchronously, the major work and contributions are : (1) carry out the remote control to the three channel telescope in LAN by internet , and reduce the cost of the observation ; (2) by using the terminal system to control the three CCD on the telescope, get the live images in different layers at the same solar active region at the same time; (3) the system's frame design uses the modular thought , encapsulates the major installations and resources as modules, and make it more effective in developing and promoting the system.

By using VC.net integrated development environment, TCP/IP protocol and socket programming, carry out the control of the three CCD of three-channel solar telescope synchronously and remotely, greatly reduce the cost of observation, and has been made preliminary observations.

Key word: three channel telescope; synchronous observation; module design; remote observation

目录

第一章 绪论

1.1 课题研究背景及意义

太阳是离我们最近的恒星，她的剧烈活动和周期变化直接或间接的影响着人类的生活，也是引起通讯和航天器故障的重要原因，因此对太阳的研究和对日地环境灾变的预报尤为重要，这就把对太阳的实时观测的需求提升到了一个新的高度。对太阳的观测可分为空间观测和地面观测，大多的观测基地都设在离城市较远，靠近水域的地方，观测成本很高，实现远程实时观测就显得十分必要。现在许多夜间观测的望远镜都具有远程观测的能力，如日本在夏威夷的 Subaru 望远镜实现了在日本本土国家天文台的远程观测[1]。太阳观测与夜间观测相比，最大特点就是数据量巨大，这就对实时远程观测提出了更高的要求。现怀柔观测基地主要有两个观测室，两个观测室之间大概相距 200 米，为了在主观测室实时监控另一个观测室观测的情况，同时可以实时比较两个观测室的数据，实现协同观测，开发了实时远程控制三通道望远镜的远程终端观测系统。

多层次太阳磁场测量是太阳物理领域的一个难点和热点，为了实现怀柔基地三通道望远镜对太阳的多层次观测，要求对望远镜的三个 CCD 在时间精度达到毫秒级别进行同步采集，本文将讨论利用软件方法实现三个相机同步采集的方法，保证数据采集的可靠性。

对于以往的观测终端，在望远镜上的不同的相机都要开发出对应的终端程序来对相机进行控制，如果一架望远镜因要升级或者其他原因更换为不同型号的相机或者增添新的相机，就要重新开发出一套观测程序，这就给望远镜的维护带来很大的不便，在三通道望远镜中有三个 CCD 相机，其中有两个为 LYNX 系列 IPX-1M48 相机，另一个为 Punix TM-1400 相机，通过对相机控制共性的提取，利用 C++语言的继承派生特性，抽象出一个相机的基类来，其他各种型号的相机通过对相机基类的继承，只需要添加各种型号相机的对应的命令便可以完成相机控制的开发，对于望远镜更换或者添加新的相机，只需修改少量代码便可完成终端升级，大大提高了维护和升级望远镜的灵活性。

1.2 各章节的主要内容

本论文在第二章中首先介绍三通道望远镜远程观测终端系统的界面、系统特点、系统服务器端和客户端的程序架构，从总体上概述了终端系统的大体框架。

第三章介绍了网络编程的相关知识，以及在三通道望远镜的远程观测中采用的网络协议和设计方案，主要从望远镜的图像传输和相机命令传输两个方面讨论了远程传输的解决方法。

第四章中首先介绍了三通道望远镜中使用的相机性能和特性，然后讨论了在三通道望远镜中使用的三个 CCD 相机的同步采集的方法。

第五章从软件编程方面介绍了控制 CCD 相机的方法，讨论了三通道望远镜终端系统中相机的模块化编程的解决方案，利用该方案可以快速的提高开发和更新望远镜终端系统的效率。

最后在第六章中展示了利用三通道望远镜远程终端系统采集得到的数据图像，然后对后续和未来的工作做了简单介绍。

第二章 三通道太阳磁场望远镜远程控制终端系统概述

三通道太阳望远镜的远程观测系统主要分为两个部分，服务器和客户端，服务器端直接控制望远镜的三个 CCD，客户端可以安装在局域网内的任何一台机子上，通过网络和服务器实现通讯，实现远程观测。结构如图 2.1 所示：

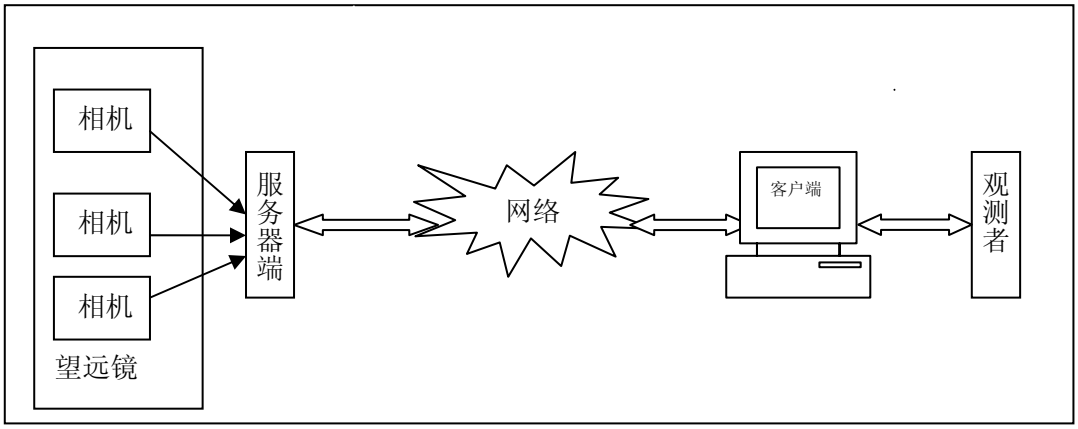


图 2.1 三通道太阳望远镜的远程观测系统结构图

本章将从系统界面，特点，和系统的程序架构上大体介绍三通道望远镜远程终端系统的框架和实现的功能。

2.1 系统界面

首先启动服务器端，各个相机做好准备，等待客户端连接，图 2.2 所示为服务器端和客户端的界面：

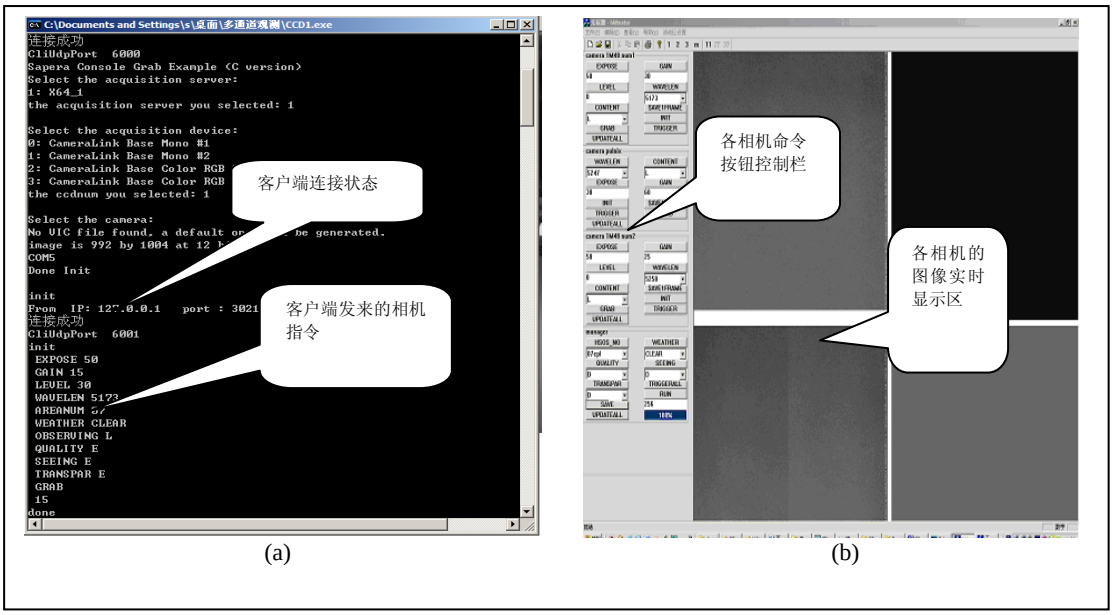


图 2. 2 远程控制系统界面 (a）服务器端 (b)客户端

客户端界面利用 vc.net 集成开发环境,MFC(Microsoft Foundation Class)窗口编程[2],开发出程序的界面。在开发客户端过程中,尽量保证客户端的功能简单,只是发送相机命令和接受相机图像数据,保证在安装有 Windows 操作系统的 PC 上就可以运行客户端程序。

对于服务器端,采用黑屏的控制台界面，主要实现三个 CCD 的同步采集,由于对同步时间要求在秒分级范围内,在不发生硬件故障和操作系统没有出现死锁的情况下,利用软件同步时间精度达到毫秒级范围内,能够实现三个相机的同步要求。

2. 2 系统特点

1) 采用动态按钮保证客户端安装的独立性

客户端采用 MFC 窗口编程，左端为控件栏，负责发送相机命令，右边的窗口客户区实时显示各个相机采集的图像。控件栏中的按钮根据服务器端的相机命令库动态的生成。这样就保证了客户端的独立性，使得客户端和相机之间没有关联，所有的命令库都来自服务器端，保证在任何一台安装有 Windows 系统的机器都可以运行该客户端,如下图 2.3 所示：

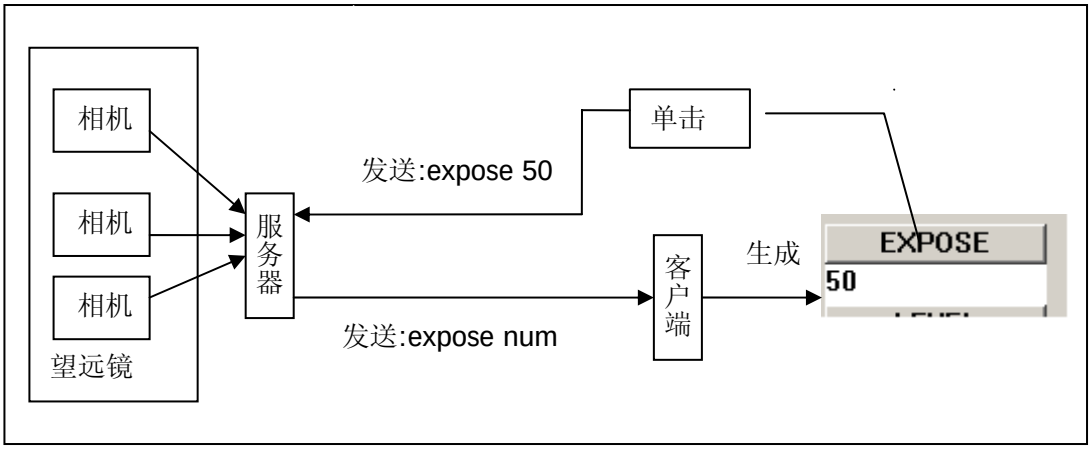


图 2.3 动态控制按钮的生成

2) 相机控制模块化编程，利用 C++语言的继承派生和多态的特性将相机的控制模块化，把相机控制的共性封装到一个相机的基类中，当望远镜更新或者增加新的相机时，可以快速方便的更新终端系统。大大提高了开发望远镜终端系统的效率和灵活性。

3) 服务器端采用了控制台式的黑屏界面，采用标准 C 的编程语言环境，避免采用和窗口相关的类，为了将来跨平台以及嵌入式系统的开发升级提供方便。

4) 多相机的控制，同时对三个相机进行控制,并利用线程和内核对象的同步[3]编程来实现三个相机的同步采集。

5) 对命令和图像数据采用不同协议传输，对于相机命令数据，采用 TCP 协议传输，保证命令准确无误的可靠传输，对于图像数据考虑到其数据量大，并且是实时显示图像，对图像质量要求不高，故采用 UDP 协议传输，允许一定程度的丢包。

2.3 服务器端

服务器端程序的类架构，如图 2.4 所示：

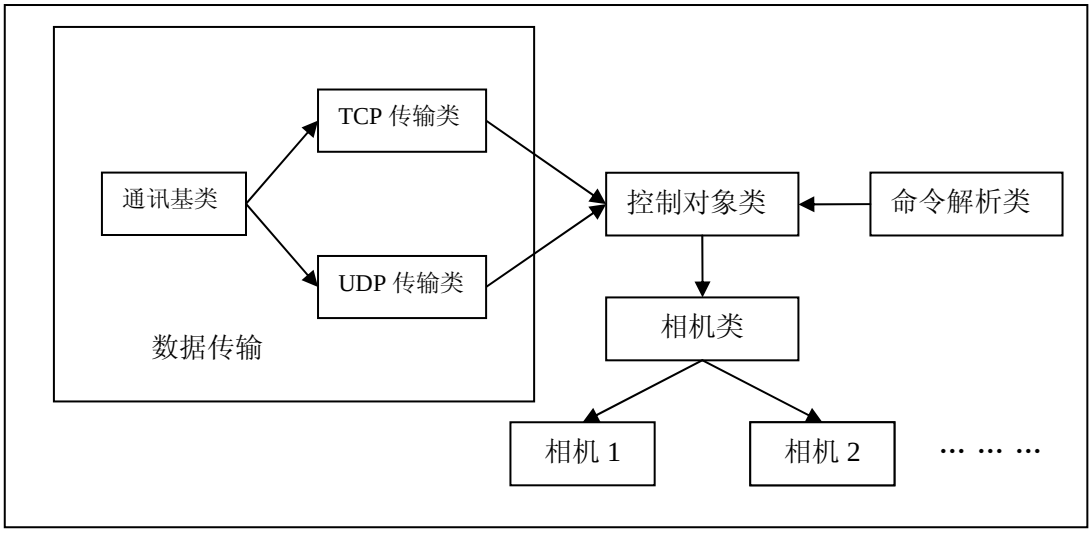


图 2.4 服务器端类架构

服务器端采用了控制台式的黑屏界面，避免采用和窗口相关的类，为了将来跨平台升级提供方便。同时相机类的设计提取了相机的的公有特性,除了控制相机的基本功能外,还有传输相机图像数据和相机命令的功能,针对不同型号的相机从相机类派生出子类来,来添加对应型号相机的命令参数，为将来更换或升级相机提供了方便，同时大大提高了代码的可移植性。

服务器端基本承担了所有的工作，客户端只负责发命令，服务器端来执行，并且通过 Camera Link 的 API 函数直接与相机通讯，服务器端主要包括网络部分，命令解析部分和相机控制部分，除此之外，服务器端还承担了以下工作：

1） 图像采集及各相机的同步

图像采集需求：每个相机采集 n 张图片，n 的值由观测者输入，并且要求三个相机都采完一幅图像时，有一个高压反转信号加载给滤光器，然后再采集下一张图像，这就要求三个相机采集图像的同步;

解决方案:创建管理类来对三个相机进行控制,另开启一个线程来采集图像,采用内核事件对三个相机的采图进行同步;

2） 对带有 FITS(Flexible Image Transport System) [4]头文件的图像的保存

需求分析：要求每个相机的图像存为两组,一组为高压图像的累加,另一组为反高压图像的累加,FITS 头文件中包括保存的图像位数,维度,相机采相时的参数,以及观测内容的参数等;

解决方案：利用高压反转信号将采集的图像分两组累加,按照 FITS 标准

写出类 CFits 对 FITS 头文件进行格式化,并添加到文件首部做为文件的头文件;

3) 计算磁场

在实时监测图像数据的同时,为了能够实时查看到太阳活动区各个层次的磁场,在每次采集完一组数据后实时计算出磁场图像并发送到客户端,通过客户端界面显示出来，观测者通过对磁场图像的查看来判断是否保存数据。

在采集一组图像的过程中,一组数据可采集多张图像,图像张数由观测者决定,CCD 相机每采集一次,有一个高压反转加载到滤光器上,CCD 相机再采集下一幅图像,最后将采集的图像按照高压和反高压分成两组,把两组图像累加得到两幅图像, 其中高压像素值和反高压像素值是通过叠加得到的两幅图像的对应像素值。

磁场计算的公式为:

磁场像素值=(高压像素值-反高压像素值)/(高压图像素值+反高压像素值)

利用每个相机采出的两组图像叠加成的两帧图像,高压图像和反高压图像并通过上面的公式来计算磁场图像。

2.4 客户端

1) 网络部分，实现和服务器的网络连接，以及通过网络传输命令数据和图像数据。
2) 控制栏部分，动态生成命令传输按钮，网络连接建立后，服务器端传来相机的命令库，通过解析命令，在控制栏中动态生成按钮控件。
3) 显示部分，包括相机图像实时显示和磁场显示，实时显示是在主窗口客户区以位图的形式显示各个相机的实时图像，磁场显示当采集完一组图像后，在服务器端计算出磁场，并传送到客户端，在另一个新的副窗口中显示出来。
4) 客户端的类架构（如图 2.5 所示）：

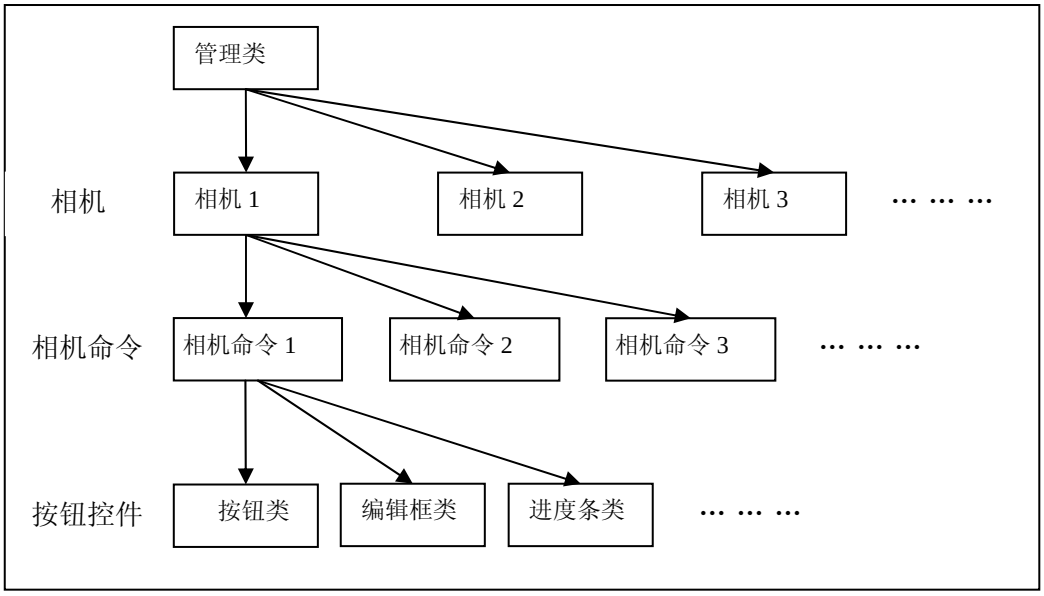


图 2.5 客户端类架构

2.5 小结

本章介绍了三通道望远镜远程控制终端的系统界面，特点以及服务器端和客户端的程序架构和各自实现的功能和任务，从大体上介绍了该系统的架构，下面将主要从数据的网络传输，相机同步采集以及相机的模块化编程三个方面具体展开讨论。

第三章　三通道望远镜网络传输及编程

本章主要介绍了三通道望远镜中数据远程传输方案的设定，以及数据传输的实现过程。三通道望远镜的数据分为两大类，一个相机采集的图像数据，另一个是控制相机的命令数据，对于数据的传输采用 Winsock 套接字的编程接口，采用 TCP/IP 协议作为数据传输的网络协议。

3.1 三通道望远镜远程观测网络协议采用方案

三通道望远镜中数据传输编程采用 Winsock 套接字编程，Winsock 是一种标准 API（application programming interface）应用程序编程接口，主要用于网络中的数据通讯，它允许两个或者多个应用程序（或进程）在同一台机器上或通过网络相互通信。使用 Winsock 接口，应用程序可通过普通网络协议如 TCP/IP（Transmission Control Protocol /Internet Protocol ）协议或 IPX（Internet Packet Exchange ）协议建立通信[5]。

采用最通用的成熟稳定的 TCP/IP 协议作为数据传输的网络协议，在望远镜观测时，有两类数据需要传输，一类是图像数据，即相机采得的实时图像，它的特点是数据量大，对数据可靠性要求不高，只需要作为实时监控，以此来判断是否真正采集一组科学数据，另一类是控制相机的命令数据，特点是数据量小，只是简单的发送几个字节的数据来控制和设定相机的属性，如相机的曝光时间、增益等，但是对数据的可靠性要求较高，不允许在传输过程中有丢失的情况发生。

根据图像数据和命令数据两类数据的特点，分别采用传输层中的 TCP（Transmission Control Protocol）和 UDP（User Datagram Protocol）两个协议来分别作为传输两类数据的传输协议，TCP 是一种面向连接的通信协议，通过序列确认以及包重发机制，提供可靠的数据流发送和到应用程序的虚拟连接服务，可以保证数据在发送过程中可靠的传送到接受端；UDP 是一种无连接的传输层协议，提供无连接的简单不可靠信息传送服务，只是将数据发出，而对数据是否到

达接收端不做验证。

根据两类数据的特点，为了提高传输效率，相机命令数据采用面向连接的TCP 传输协议来传输，图像数据采用无连接 UDP 协议来传输。下面分别介绍相机命令数据和图像数据的传输过程。

3.2 相机控制命令的传输

相机控制命令包括相机的曝光时间，增益，底值等相机的参数，还包括使相机初始化，控制相机采集图像的启示命令，采集图像张数命令，以及保存图像等命令，对于这些命令的传输采用面向连接的 TCP 协议来传输，下面首先介绍一下面向连接的通信的 Winsock 编程。

3.2.1 面向连接的通信

本节将介绍接受连接和建立连接所需要的 Winsock 函数。首先讨论的是如何通过监听客户机连接来开发服务器，并探讨接受或拒绝一个连接的过程。随后讨论的是怎样通过初始化与服务器的连接来开发客户机。最后讨论数据在面向连接会话中是如何传输的。

在 IP 中，面向连接的通信是通过 TCP/IP 协议完成的。TCP 提供两个计算机间可靠无误的数据传输。应用程序使用 TCP 通信时，在源计算机和目标计算机之间，会建立一个虚拟连接。建立连接之后，计算机之间便能以双向字节流的方式进行数据交换。

3.2.1.1 服务器 API 函数

这里所说的服务器其实是一个进程，它需要等待任意数量的客户机与之建立连接，以便为它们的请求提供服务。服务器必须在一个已知的名称上监听连接。在 TCP/IP 中，这个名称就是本地接口 IP 地址，再加上一个端口编号。每种协议都有一套不同的寻址方案，所以各自的命名方法也不同。在 Winsock 中，第一步是用 Socket 或 WSASocket 将给定协议的套接字绑定到它已知的名称上，这个过程是通过 bind API 调用来完成的。下一步是将套接字置为监听模式，这一步用 API 函数 listen 来完成。最后，若一台客户机试图建立连接，服务器必须通过 accept

或 WSAAccept 调用来接受连接[5]。连接过程如图 3.1 所示：

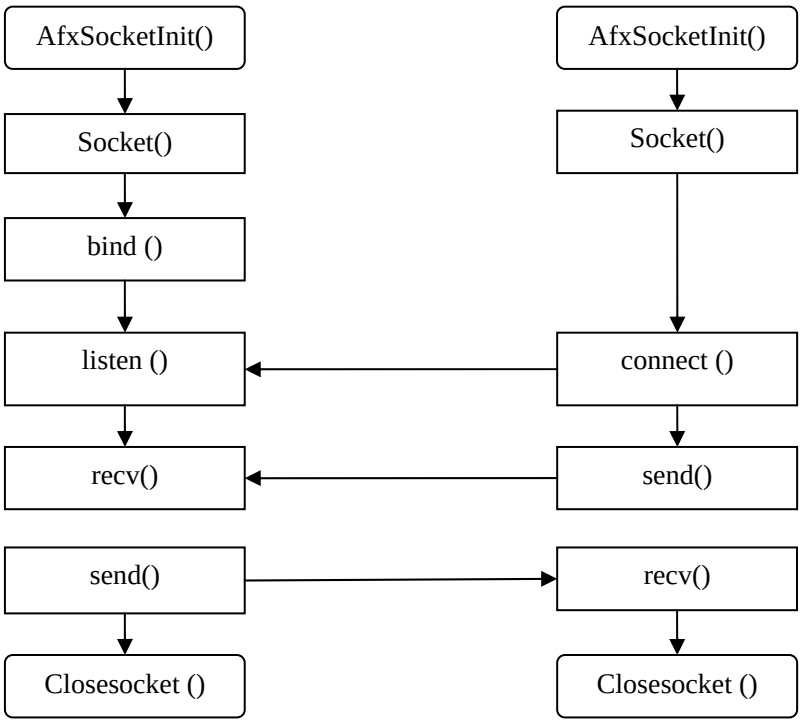


图 3.1 面向连接建立的流程图

3.2.1.1.1 绑定

一旦为某种协议创建了套接字，就必须将套接字绑定到一个已知的地址上。Bind 函数可将指定的套接字同一个已知地址绑定到一起。该函数的声明如下：

```
int bind(
    SOCKET                  s,
    const struct sockaddr FAR *  name,
    int                     namelen
);
```

其中第一个参数 s 代表用来等待客户机连接的那个套接字。第二个参数的类型是 struct sockaddr, 它的作用很简单，就是一个普通的缓冲区。根据所使用的那个协议，必须实际的填充一个地址缓冲区，并在调用 bind 时将其转换为一个 struct sockaddr。Winsock 头文件将 SOCKADDR 类型定义为 struct sockaddr。第三个参数代表要传递的，由协议决定的地址结构的长度。

一旦出错，bind 函数就会返回 SOCKET_ERROR。对 bind 而言，最常见的错

误是 WSAEADDRINUSE。如果使用的是 TCP/IP，那么 WSAEADDRINUSE 就表示另一个进程已经同本地 IP 接口及端口号绑定到了一起，或者那个 IP 接口和端口好处于 TIME_WAIT 状态。假如对一个已经绑定的套接字调用 bind，返回的将是 WSAEFAULT 错误。

3.2.1.1.2 监听

接下来要做的就是将套接字置入监听模式。Bind 函数的作用只是将套接字和指定的地址关联在一起。指示套接字等候连接传入的 API 函数则是 listen,其定义如下：

```
int listen(
    SOCKET       S,
    int    backlog
);
```

第一个参数同样是一个被绑定的套接字。Backlog 参数指定了被搁置的连接的最大队列长度。因为完全可能同时出现几个服务器的连接请求，所以这个参数非常重要。假定 backlog 参数为 2 。如果 3 个客户机同时发送请求，那么头两个会被放在一个“挂起”的队列中，以便应用程序依次为它提供服务。而第 3 个连接请求会失败，返回一个 WSAECONNERFUSED 的错误。一旦服务器接受了一个连接，那个连接请求就会从队列中删去，以便别人可继续发出请求。Backlog 参数其实本身就存在着限制，这个限制是由下层的协议提供程序决定的。如果这个参数出现非法值，那么系统会用与之最接近的一个合法值来取代。

与 lesten 相关的错误是非常直观的。到目前为止，最常见的错误是 WSAEINVAL。该错误通常意味着，在调用 listen 之前没有调用 bind。

3.2.1.1.3 接受连接

现在已经做好了接受客户机连接的准备。这是通过 accept 或 AcceptEx 函数完成的，Accept 的原形如下：

```
SOCKET accept(
    SOCKET s,
    struct sockaddr FAR * addr,
    int FAR * addrlen
```

);

其中参数 s 是一个被绑定的套接字，他处于监听模式。第二个参数应该是一个有效的 SOCKADDR_IN 结构的地址，而 addrlen 应该是 SOCKADDR_IN 结构的长度。对于属于另一种协议的套接字，应当用与那种协议对应的 SOCKADDR 结构来替换 SOCKADDR_IN。通过对 accept 函数的调用，可以为被搁置的连接队列中的第一个连接请求提供服务。Accept 函数返回之后，addr 结构会包含发送连接请求的那个客户机的 IPv4 地址信息，而 addrlen 参数则指出 addr 结构的长度。此外，accept 会返回一个新的套接字描述符，它对应于已经被接受的那个客户机连接。该客户机后续的所有操作都应该使用这个新的套接字，至于原来那个监听套接字，它仍然用于接受其他客户机连接，而且仍处于监听模式。

如果出错，INVALID_SOCKET 将被返回。在监听套接字为异步或非阻塞模式，摒弃没有连接被接受时，最常见的错误是 WSAEWOULDBLOCK。

3.2.1.2 客户端 API 函数

客户机的创建要简单的多，建立成功连接所须的步骤也要少得多。创建客户机只需 3 步操作：

1．创建一个套接字。

2．建立一个 SOCKADDR 地址结构，结构名称为准备连接到的服务器名，对于 TCP/IP ，就是客户机应用程序所监听的服务器的 IP 地址和端口号。

3．用 connect 或 WSAConnect 初始化客户机与服务器的连接。

由于已经知道如何建立套接字和建立 SOCKADDR 结构，下面就是建立一个连接。连接套接字是通过调用 connect 函数来完成的，其定义如下：

```
int connect(
    SOCKET      s,
    const struct sockaddr FAR * name,
    int namelen
);
```

这个函数的参数定义为：参数 s 是即将在其上面建立连接的那个有效 TCP

套接字；Name 参数是 TCP 的套接字地址结构，表示要连接到的服务器；namelen 则是 name 参数的长度。

如果想连接的计算机没有进程用于监听给定端口，connect 调用就会失败，并产生 WSAECONNERFUSED。另一个错误可能是 WSAETIMEDOUT, 这种情况一般发生在试图了解的计算机不可用时。

3.2.1.3 数据传输

要在已建立连接的套接字上发送数据，可用这两个 API 函数：send 和 WSASend。第二个函数是 Winsock2 中特有的。同样的，在已建立了连接的套接字上接收数据也有两个函数：recv 和 WSARecv。后者也是 Winsock2 函数。所有关系到收发数据的缓冲区都属于简单的 char 类型，即面向字节的数据。事实上，它可能是一个包含任何原始数据的缓冲区，至于这个原始数据是二进制数据还是字符型数据，则无关紧要。

另外，所有的收发函数返回的错误代码都是 SOCKET_ERROR。最常见的错误是 WSAECONNABORTED 和 WSAECONNRESET。

3.2.1.3.1 send 函数

要在已建立连接的套接字上发送数据，可用 API 的 send 函数，其原型为：

```
int send(
    SOCKET          s,
    const char FAR *    buf,
    int                 len,
    int                 flags
);
```

SOCKET 参数是已建立了连接，将用于发送数据的套接字。第二个参数 buf 则是指向字符缓冲区的指针，该缓冲区中包含即将发送的数据。第三个参数 len，指定即将发送的缓冲区内的字符数。最后，flags 可为 MSG_DONTROUTE 或 MSG_OOB。另外，flags 还可以是对这些标志进行按位或运算的一个结果。MSG_DONTROUTE 标志要求传输层不要将它发出的数据包路由出去。是否实现

这一请求由下层的传输来决定.

在顺利的情况下，send 将返回发送的字节数：若发生错误，就返回 SOCKET_ERROR。常见的错误是 WSAECONNABORTED, 这一错误一般在虚拟回路由于超时或协议有错而中断的时候发生。发生这种情况时，因为套接字不能再用了，所以应该将它关闭。当远程主机上的应用程序通过执行强行关闭或意外中断操作重新设置虚拟回路时，或远程主机重新启动时，发生的则是 WSAECONNERET 错误。发生这一错误时，应该关闭这个套接字。最后一个常见的错误是 WSAETIMEOUT，它在连接由于网络故障或远程连接系统异常死机而导致连接中断时发生。

3.2.1.3.2 recv 函数

对于在已建立连接的套接字上接受数据的传入来说，recv 是最基本的方式，其定义如下：

```
int recv(
    SOCKET      s,
    char    FAR * buf,
    int       len,
    int       flags
);
```

第一个参数 s，是准备用来接收数据的那个套接字。第二个参数 buf，是用于接收数据的字符缓冲区，而 len 则是准备接收的字节数或 buf 缓冲区的长度。最后的 flags 参数可以是下面的值：0, MSG_PEEK, MSG_OOB。另外，还可对这些标志中的每一个进行按位“或”运算。0 表示没有特殊的行为。MSG_PEEK 表示要将可用的数据复制到所提供的接收端缓冲区内，但是并不从系统缓冲区中将这些数据删除。待发字节数也将被返回。

消息查看功能有一些缺点。它不仅仅导致性能下降，因为需要进行两次系统调用，一次是查看，另一次是无 MSG_PEEK 标志的真正删除数据的调用，在某些情况下可能不可靠。返回的数据可能没有反映出真正可用的总量。与此同时，把数据留在系统缓冲区内，可容纳传入数据的系统空间就会越来越少。结果使得系统减少各发送端的 TCP 窗口容量。这样，应用程序就不能获得最大的数据吞

吐率。因此，最好尽量把所有数据都复制到自己的缓冲区中，并在那里操作数据。

在基于消息或基于数据报的套接字上使用 recv 时，如 UDP 当挂起数据大于所提供的缓冲区时，缓冲区会尽量的让数据填满。这时，recv 调用会产生 WSAEMSGSIZE 错误。消息大小的错误是在使用面向消息的协议时发生的。而流协议，如 TCP 则把传入的数据缓存下来，并尽量的返回应用程序所要求的数据，即使被挂起的数据量比缓冲大。因此，对流传输协议来说，就不会碰到 WSAEMSGSIZE 这个错误。

3.2.1.4 流协议

大多面向连接的协议，如 TCP 同时也是流传输协议，在流传输协议中，发送者和接收者可以将数据分解成小块数据。或将数据合并为大块数据。对于流套接字上收发数据所用的函数，它们不能保证要求进行读取或写入的数据量。如用 send 函数来发送一个有 2048 字节的字符缓冲区，采用下面的代码：

```
//用 2048 字节的数据填充 sendbuff
char sendbuff[2048];
int    nBytes = 2048;

//假定 s 是一个有效的、已连接的流套接字
ret = send (s,sendbuff,nBytes,0);
```

对于 send 函数而言，可能会返回已发出的少于 2048 的字节，因为对每个收发数据的套接字来说，系统都为他们分配了相当充足的缓冲区空间，所以 ret 变量将被设定为已发送的字节。在发送数据时，内部缓冲区会将数据一直保留到可以将它发到线上为止。几种常见的情况可导致这一情形的发生，如传输大量的数据可以令缓冲区快速填满。同时，对 TCP/IP 来说，还有一个窗口大小的问题。接收端会将窗口大小设定为 0，为挂起的数据做好准备。对发送端来说，这样会强制它在收到一个新的大于 0 的窗口大小之前，不得发送数据。在使用 send 调用时，缓冲区可能只能容纳 1024 个字节，这时便有必要重新提交剩下的 1024 个字节。要保证将所有的字节发出去，可采用下面的代码：

```
char sendbuff[2048];
int    nBytes = 2048,nLeft,idx;
//用 2048 字节的数据填充 sendbuff
//假定 s 是一个有效的、已连接的流套接字
nLeft = nBytes;
idx = 0;

while(nLeft>0)
{
      ret = send(s,&sendbuff[idx],nLeft,0);
      if(ret == SOCKET_ERROR)
      {
            //出错
      }
      nLeft -= ret;
      idx +=ret;
}
```

在流套接字上接收数据时，这一原则同样适用，但意义不大。因为流套接字是一个不间断的数据流，在读取它时，应用程序通常不会关心应该读多少数据。如果所有消息长度都一样，则处理起来比较简单，如需要读取的 512 个字节的消息代码如下：

```
char recvbuff[1024];
int    ret,nLeft,idx;

nLeft = 512;
idx = 0;

while(nLeft>0)
{
      ret = recv(s,&recvbuff[idx],nLeft,0);
      if(ret = SOCKET_ERROR)
      {
            //出错
      }
      idx +=ret;
```

```
        nLeft -=ret;
    }
```

如果消息长度不同，就有必要利用自己的协议来通知接收端，让它知道即将到来的消息长度是多少。如写入接收端的前 4 个字节总是整数，大小为即将到来的消息的字节数，接收端每次开始读取时，会先查看前 4 个字节，把它们转换成一个整数，并查看构成消息的字节数是多少。

3.2.1.5 中断连接

在关闭一个连接时，套接字的状态很重要，对于每一个套接字来说，它的初始状态都是 CLOSED。若客户机初始化了一个连接，就会向服务器发送一个 SYN 包，同时将客户机套接字置为 SYN_SENT。服务器收到 SYN 包后，会发出一个 SYN-ACK 包，可客户机需要用一个 ACK 包对它做出响应。此时，客户机的套接字将处于 ESTABLISHED 状态。如果服务器一直不发送 SYN-ACK 包，客户机就会超时，并返回到 CLOSED 状态[6]。

若服务器的套接字同本地接口及端口绑定起来，并在它上面进行监听，那么套接字的状态便是 LISTEN。客户机试图与服务器连接时，服务器就会收到一个 SYN 包，并用一个 SYN-ACK 包做出回应。服务器套接字的状态变成 SYN_RCVD。最后，客户机发出一个 ACK 包，它将使服务器套接字的状态变成 ESTABLISHED。一旦应用程序处于 ESTABLISHED 状态，就可以通过两种方法来关闭它。如果由应用程序来关闭，便叫做“主动套接字关闭”；否则，套接字的关闭便是被动的。如果是主动关闭，应用程序便会发出一个 FIN 包。应用程序调用 closesocket 或 shutdown 时，会向通信对方发送一个 FIN 包，而且套接字的状态将变成 FIN_WAIN_1。正常情况下，通信对方会用一个 ACK 包作为回应，套接字的状态随之变成 FIN_WAIT_2。如通信对方也关闭了连接，它会发出一个 FIN 包，我们的机器则会以一个 ACK 包作为回应，并将套接字的状态置为 TIME_WAIT。TIME_WAIT 状态也叫做 2MSL 等待状态。其中，MSL 代表“分段最长生存时间”（Maximum Segment Lifetime），表示一个数据包在被丢弃之前，可在网络上存在多长时间。每个 IP 包都含有一个 TTL（time-to-live，生存时间）字段。若它递减到 0，包便会丢弃。一个包经过网络上的每个路由器时，TTL 值都会减 1，然后

继续传递，一旦应用程序进入 TIME_WAIT 状态，那么就会一直持续两倍 MSL 的时间。这样，如果最后一个 ACK 包丢失，TCP 就可以重新发送它，也就是说，FIN 会被重新传送出去。两倍于 MSL 时间的等待状态结束之后，套接字便进入 CLOSED 状态。如图 3.2 所示：

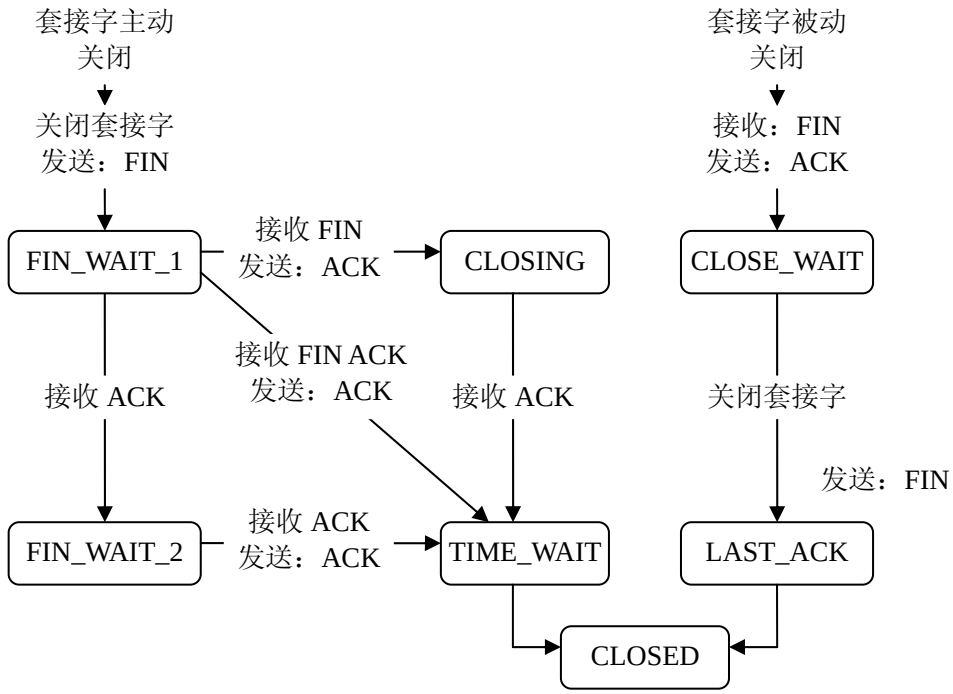


图 3.2 中断连接流程图

采取主动关闭措施时，沿两条路径可以进入 TIME_WAIT 状态。当只有一方发出一个 FIN，并接受一个 ACK 回应，另一方仍然可以自由地发送数据，直到它也被关闭为止。因此，需要另外两条路线发挥作用。在另一条路径中，一台计算机与之连接的通信对方会同时要求关闭；计算机向对方送出一个 FIN 数据包，并从它那里接受一个 FIN 数据包。随后计算机会发出一个 ACK 数据包，对对方的 FIN 包做出回应，并将自己的套接字置为 CLOSING 状态。计算机从对方那里接收到最后一个 ACK 包之后，它的套接字就变成 TIME_WAIT。

主动关闭时，另一条路径其实就是同步的变体：套接字从 FIN_WAIT_1 状态直接变成 TIME_WAIT 状态。若应用程序发出一个 FIN 数据包，但马上从对方那里接收到一个 FIN-ACK 包时，这种情况就会发生。在这种情况下，对方会确认是否收到了应用程序的 FIN 包，并送出自己的 FIN 包。对于这个包，应用程序会以一个 ACK 包做出回应。

TIME_WAIT 状态的主要作用是，在 TCP 连接处于 2MSL 等待状态的时候，定义那个连接的一对套接字不可以被重新使用。这对套接字由本地 IP 端口以及远程 IP 端口组成。某些 TCP 实施方案不允许重新使用处于 TIME_WAIT 状态下的套接字对中的任何端口号。在微软的实现的 TCP 连接中，不会存在这个问题。然而若试图通过一对已处于 TIME_WAIT 状态的套接字建立连接，连接的建立就会失败，并返回 WSAEADDRINUSE 错误。

在被动关闭的情况下，应用程序会从对方那里收到一个 FIN 包，并用一个 ACK 包做出回应。此时，应用程序的套接字就会变成 CLOSE_WAIT 状态。由于对方已关闭自己的终端，所以它不能再发送数据了。但应用程序却不同，它能一直发送数据，直到它关闭了自己的连接终端为止。要想连接终端，应用程序会发送出自己的 FIN 包，令应用程序的套接字状态变成 LAST_AKC。应用程序从对方接受到一个 ACK 包后，它的套接字就会到 CLOSED 状态。

在编程时，利用函数 closesocket 就能够释放于一个打开的套接字句柄关联的资源，但是 closesocket 可能会带来负面影响，即可能会导致数据的丢失，所以在调用 closesocket 之前，应该先利用 shutdown 函数从容的终止连接。

3.2.1.5.1 shutdown

为了保证通信方能够收到应用程序发出的所有数据，对一个好的应用程序来说，应该通知接受端“不再发送数据”。同样通信对方也应该如此。这就是所谓的正常关闭方法，由 shutdown 函数来执行。Shutdown 的定义如下：

```
int shutdown(
    SOCKET      s,
    int         how,
);
```

how 参数可以是下面的任意一个值：SD_RECEIVE, SD_SEND 或 SD_BOTH。如果是 SD_RECEIVE，就表示不允许再调用接收函数。这对底部的协议层没有影响。对于 TCP 套接字来说，不管数据是等候接收，还是将随后抵达，都要重新设置连接。在 UDP 套接字上，仍然接受并排列传入的数据，因为对于无连接协议而言，shutdown 毫无意义。如果选择 SE_SEND，表示不允许再调用发送数据。

对于 TCP 套接字来说，这样会在所有数据发出，并得到接收端确认之后，生成一个 FIN 包。最后如果指定 SD_BOTH，则表示取消连接两端的收发操作。

3.2.1.5.2 closesocket

closesocket 函数用于关闭套接字，它的定义如下：

```
int colsesocket(SOCKET s);
```

对 closesocket 的调用会释放套接字描述符，然后再利用套接字执行的调用就会失败，并出现 WSAENOTSOCK 的错误。如果没有对该套接字的其他引用，那么所有与套接字描述符关联的资源都会被释放。其中包括丢弃所有队列中的数据。

对这个进程中任何一个线程来说，它们发出的被挂起的同步调用都会在未投递任何通知消息的情况下被删除。被挂起的重叠操作也会被删除。与该重叠操作关联的任何事件、完成例程或完成端口能被执行，但最后都会失败，并出现 WSA_OPRATION_ABORTED 错误。

3.2.2 三通道望远镜相机命令传输

三通道望远镜相机命令的传输采用面向连接的 TCP 协议传输，在服务器端，TCP 通讯的实现是通过基类 CComm 的继承派生来生成的 CTcp 类来完成的，其中 CComm 类中封装了 TCP 和 UDP 共同的部分，如远程通讯地址，端口号等，在 CTcp 中添加接收和发送的功能函数，做为一个实例对象内嵌在相机的类中，使得相机类具有远程通信的能力，接收和发送数据。

在客户端，通过和服务器端建立连接来发送和接收数据，当客户端和服务器端建立好连接后，服务器端会将该相机的命令集合发送给客户端，客户端将命令库中的每条命令解析出来，同时根据这些命令动态生成按钮，捕捉这些按钮的消息，通过点击这些按钮就能够向服务器端发送相应的命令，客户端和服务器端的通信流程如图 3.3 所示：

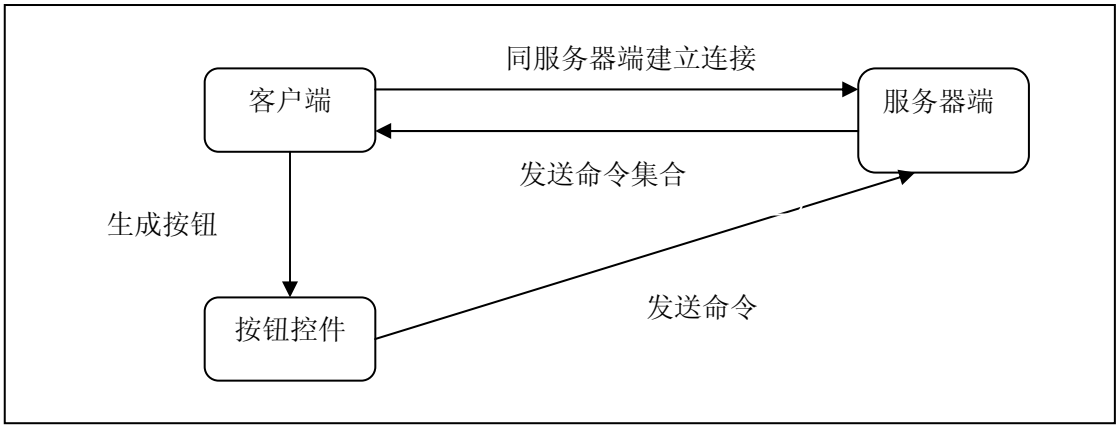


图 3.3 建立连接流程图

3.3 相机图像数据的传输

对于相机的图像数据，拿 IPX-1M48 相机来说，一幅图的大小大约是 1M 左右，如果曝光时间为 20 毫秒，每秒钟的数据量为 50M，数据量是很大的，为了缓解图像数据的压力，采取了以下两个措施，一是将采集的图像抽取一部分，数据量变为原来数据量的四分之一，二是将图像数据的传输选择无连接 UDP 来传输，有效减少了带宽的占有，同时也节省了系统资源。下面介绍下无连接通讯的连接步骤。

3.3.1 无连接通信

和面向连接的协议比较起来，无连接协议的行为有很大不同，因此，收发数据的方法也会有所差别。和面向连接的服务器比较起来，无连接接受端改动不大，所以先讨论接收端，也可以称之为服务器，然后再讨论发送端。

在 IP 中，无连接通信是通过 UDP/IP 协议完成的。UDP 不能确保可靠的数据传输，但能将数据发送到多个目标，或者接受的多个源数据。例如，如果一个客户机向一个服务器发送数据，则数据将立刻被传输，不管服务器是否准备接收这些数据。如果服务器接受来自客户机的数据，该服务器也不会发消息确认数据已收到。数据的传输使用数据报，即离散信息包。

3.3.1.1 接收端

对于在一个无连接套接字上接收数据的进程来说，先用 socket 或 WSASocket 创建套接字。再把这个套接字和准备接收数据的接口绑定在一起，这是通过 bind 函数来完成的。和面向连接不同的是，不必调用 listen 和 accept。相反，只需等待接收数据。由于没有连接，始发于网络上任何一台机器的数据报都可被接收端的套接字接收。最简单的接收函数是 recvfrom。它的定义为：

```
int recvfrom(
        SOCKET        s,
        char FAR *    buf,
        int           len,
        int           flags,
        struct sockaddr FAR * from,
        int FAR *     fromlen
);
```

前面四个参数和 recv 的参数是一样的，其中包括标志的可能值：MSG_PEEK 和 MSG_OOB。对监听套接字的给定协议来说，from 参数是一个 SOCKADDR 结构，带有指向地址结构长度的 fromlen。这个 API 调用的返回数据时，SOCKADDR 结构内便填入了发送数据的那个工作站的地址。

在无连接套接字上接收（发送）数据的另一种方法是建立连接。无连接的套接字一旦建立，便可利用 SOCKADDR 参数（它被设为准备与之通信的远程接收端地址）调用 connect 或 WSAConnect。但事实上并没有建立真正的连接。传递到连接函数的套接字地址是与套接字关联在一起的，如此一来，才能够用 recv 和 WSARecv 来代替 recvfrom 和 WSARecvFrom。其原因是数据的始发处是已知的。如果在应用程序中，一次只和一个端点进行通信，便能很容易的与数据报套接字建立连接。

3.3.1.2 发送端

要在一个无连接的套接字上发送数据，直接建立一个套接字，调用 sendto 就可以发送数据了，它的定义是这样的：

```
int sendto(
    SOCKET      s,
    const char FAR * buf,
    int         len,
    int         flags,
    const struct sockaddr FAR * to,
    int         tolen
);
```

除了 buf 是发送数据的缓冲区，len 指明发送多少字节外，其余参数和 recvfrom 的参数一样。To 参数是一个指向 SOCKADDR 结构的指针，该结构带有接收数据的那个工作站的目标地址。

通过接收数据的方式，就可以把一个无连接的套接字连接到一个端点地址，并可以用 send 发送数据。这种关联一旦建立，就不能再用有地址的 sendto，除非这个地址是传递到其中一个连接函数的地址，否则调用就会失败，出现 SEAEISCONN 错误。要取消套接字句柄与目标地址的关联，唯一的办法就是在这个套接字句柄上以 INADDR_ANY 为目标地址调用 connect.

3.3.1.3 释放套接字资源

因为无连接协议没有连接，所以也不会有对连接的正式关闭和从容关闭。在接收端完成收发数据时，它只需要在套接字句柄上调用 closesocket 函数，便可释放为套接字分配的所有相关资源。

3.3.2 三通道望远镜的图像数据传输

同发送相机命令相比，图像的传输要简单一些，客户端和服务器端之间不需要建立连接，直接发送数据到目的地址就可以了，对于服务器端来说，同发送相机命令一样，从通信的基类 CComm 继承派生出类 CUdp，再将 CUdp 的一个实例对象内嵌到相机的类中，使得相机类具有传输图像数据的功能。在每传输一幅图像前，根据图像的大小，先将图像分割成大小一样的数据包，每个数据包按照顺序排列序号，然后将每个数据包发送到客户端。

在客户端，通过开启一个线程来接受数据，根据每个数据包的序号来重新将一幅图像拼接出来，然后再显示到客户端界面窗口的客户区。

第四章 三通道望远镜 CCD 相机的应用及同步控制

在三通道望远镜中，使用了三台 CCD 相机，其中两台为 LYNX 系列 IPX-1M48 相机，另一台为 Punix TM-1400 相机，三台相机在望远镜工作的过程中必须同步采集以保证太阳同一区域不同层次数据的采集，本章首先介绍三台 CCD 相机的性能和参数，然后讨论了在三通道望远镜中三台 CCD 同步观测的实现方案。

4.1 LYNX 系列 IPX-1M48 相机

LYNX 相机是一种 CCD 相机,它具有高级的,高分辨率的,累积扫描,完全可编程和局部模块升级的特性.这些相机的图像处理引擎是建立在一百万 FPGA 门和 32 位 RISC 处理器上,具有可编程的图像分辨率、帧速率、增益、偏移, 可编程曝光时间完成的异步外部触发,快速触发,双重曝光和电子快门,长时间积分,开关输出,,转换函数校正,温度监控,和用户可编程可上载的 LUT(LookUpTable),电子快门和长时间曝光的综合运用可以使相机曝光时间达到从 1/200,000 秒到十秒钟.同时这种相机还包括标准的 Camera Link 接口,兼容 8 ,10 ,12 位的数据输出,可用一个通道或者两个通道来传输。同时还包含 RS232 异步串口通信接口来控制相机,所有的线路都包含在一根电缆中。这些相机可以通过 Camera Link 接口使用基于 GUI 的界面来设置,也可以通过任意的模拟终端使用简单的 ASCII 命令来控制相机。相机的适应性和可延展性使得其有广泛的应用空间,如机器视觉、高清晰图像的衡量、监控、医药、和科学图像、智能运输系统、特征辨认、文档处理等领域[6]。

CCD 相机是将光信号转化为电信号的一种电子器件[7],相机包括光感应部件 CCD(Charge Coupled Device) ,通过对光的感应来产生电信号,CCD 包含两维阵列的感应元素，硅光电管，就是我们俗称的像素。当光子落入像素时就会产生光电子，同时产生的光电子数是和光强成线性比例的[8]，虽然每个像素所收集的电子数目是和光强和曝光时间成线性关系，但是电子的数目还会随着光的波长的改变

而改变。当曝光完成后，每个像素的电子被转移到一个垂直的寄存器中（VCCD），然后按照行为单位垂直往下移位，转移到水平寄存器（HCCD）。接着从水平寄存器中按照像素为单位进行水平移位，一次转移一个像素，转移到输出节点中，同时在这里将电子数转化为电压值。然后将获得的电压信号转存在一个视频放大器中，然后将其发送到对应的视频输出。在每个水平寄存器的两端各有一个视频放大器，根据操作模式的 不同，电子可以被发送到任意一个输出。从获得整个图像到从水平寄存器中将所有像素都输出所用的时间间隔称为一帧。为了获得彩色图像，一组颜色滤波器（红，绿，蓝）按照层模式安放在像素之上。开始的颜色是绿色。(如图 4.1 所示)

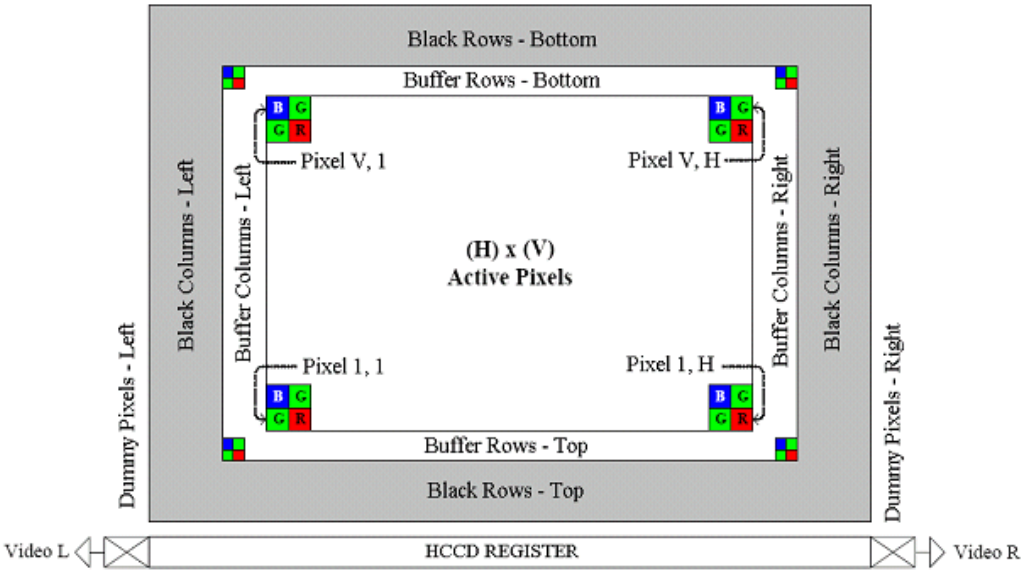


图 4. 1 彩色像素结构图

4. 1. 1 相机特性

1．单通道输出：当设置在单通道输出模式，所有的像素移位到水平寄存器（HCCD）从左视频放大器输出。如图 4.2 所示：

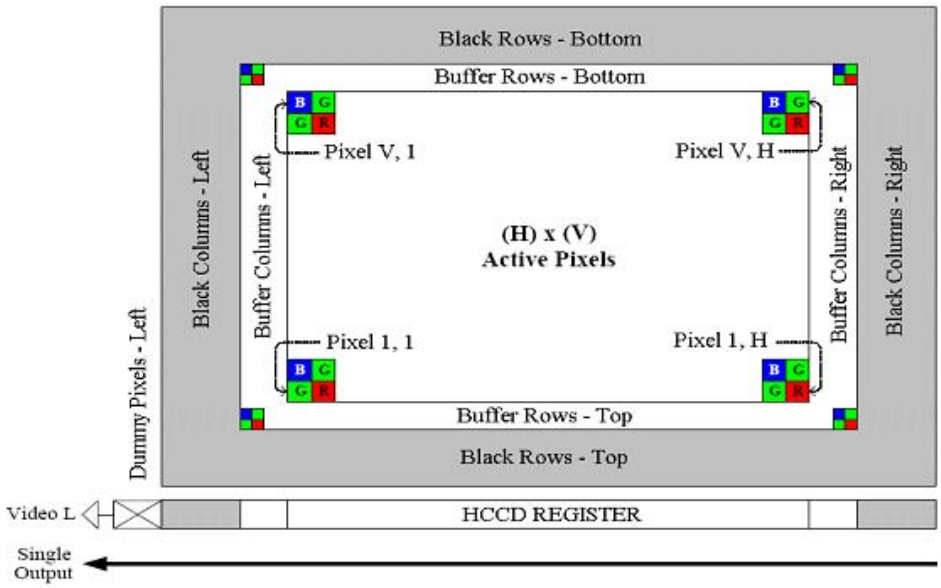


图 4. 2 单通道输出结构图

2．双通道输出：当设置为双通道模式，图像被分裂为两个相等的部分，每部分包括一半水平的像素和全部的垂直的像素部分，左边的像素值移位到水平寄存器后，输出到左边的视频放大器，而右边的输出到右视频放大器(如图所示)。在水平方向上，第一部分的图像是正常的，而第二部分的图像是左或者右镜像的。相机通过翻转和重组镜像的部分来重新构建图像。如图 4.3 所示：

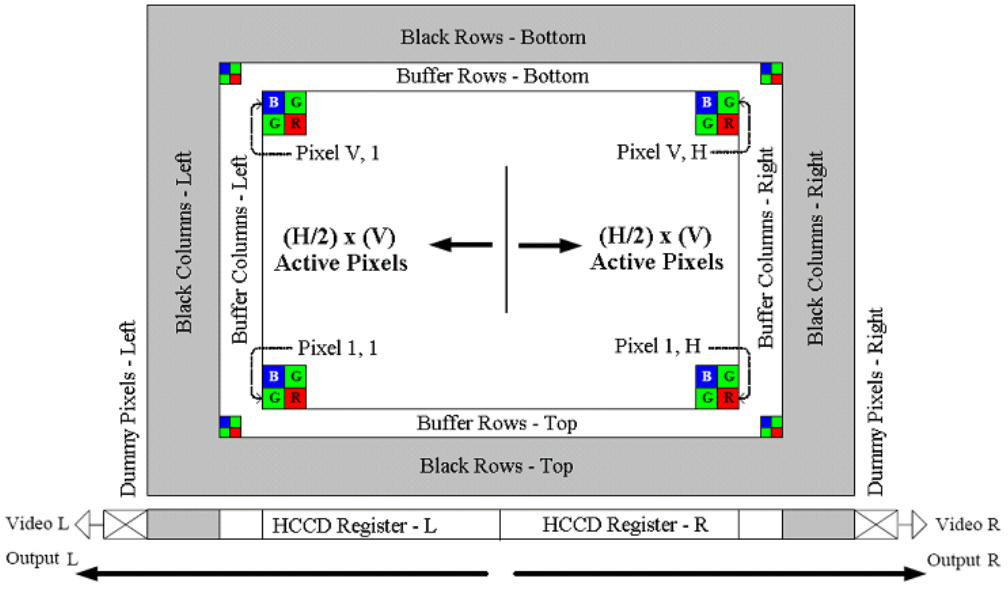


图 4. 3 双通道输出结构图

3．IPX-1M48 时序表

在单通道模式下，每行包含 8 个空的像素（E1-E8），后跟 12 个伪像素作为黑

色的参照（R1-R12）,后跟 2 个缓存像素（B1，B2），后跟 1000 个可用像素（D1-D1000），后跟 2 个缓存像素（B1，B2），后跟另外 12 个伪像素（R1-R12）。如图 4.4 所示：

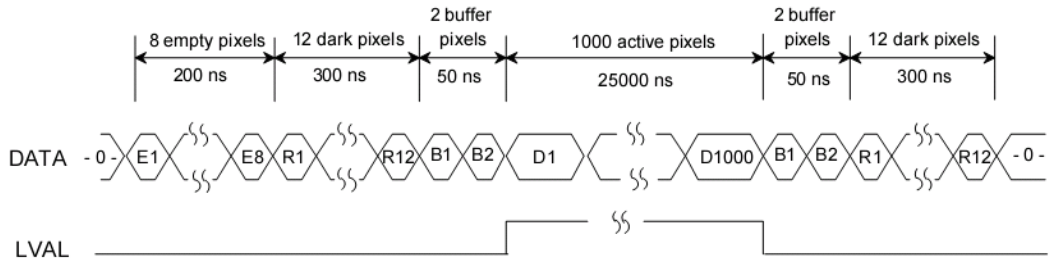


图 4. 4 单通道行输出时序图

在双通道模式下，每行包括 8 个空像素（E1-E8），后跟 12 个伪像素（R1-R12），后跟 2 个缓存像素（B1，B2），后跟 500 个可用的像素（D1,D500），如图 4.5 所示：

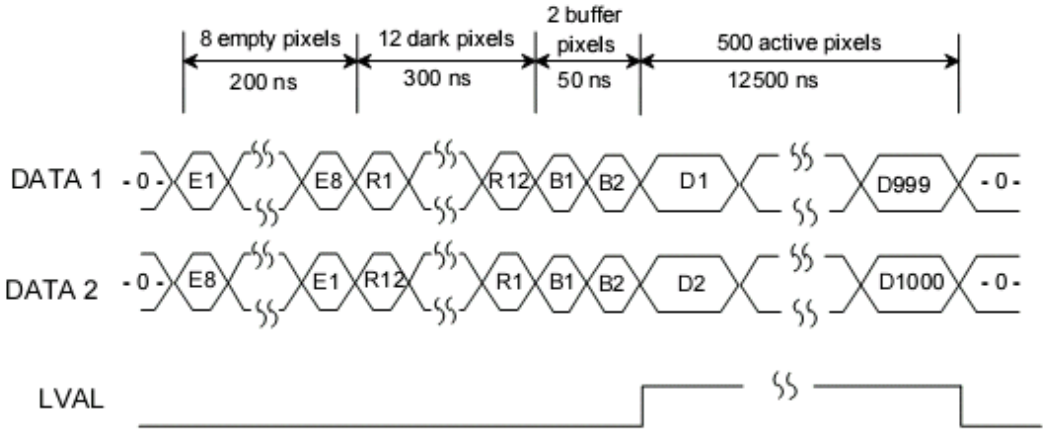


图 4. 5 双通道输出时序图

在时钟的上升沿开始对数据采样，LVAL（line valid）信号在可用像素段内激活为上升沿。每一帧（对于所有节点）包括 61 微秒的垂直帧时序，然后是 4 个伪行（RL1-RL4），后跟 2 个缓存行（BL1，BL2），后跟 1000 个可用行（DL1-DL1000）。在可用行段的范围内（DL1-DL1000），每帧的 FVAL（frame valid）信号被激发为上升沿。如图 4.6 所示：

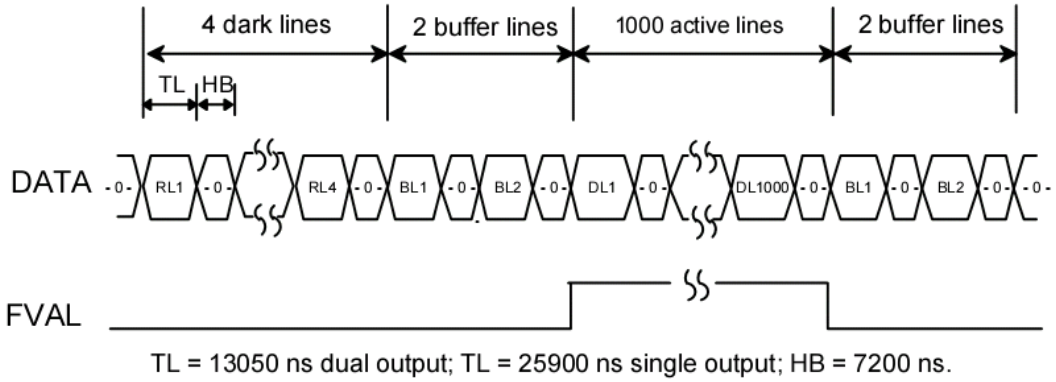


图 4.6 帧同步信号时序图

4.1.2 相机连接

LYNX 相机和外界仪器设备的连接是通过定位在相机后面板的两个接口和一个 LED 指示灯实现的，如图 4.7 所示：

1．相机输出——标准的 Camera Link 接口，提供数据传输，同步，控制和串口接口；

2．电源接口——提供电源和 I/O 接口；

3．状态ＬＥＤ——提示相机当前的状态；

4．序列号——显示相机的型号；

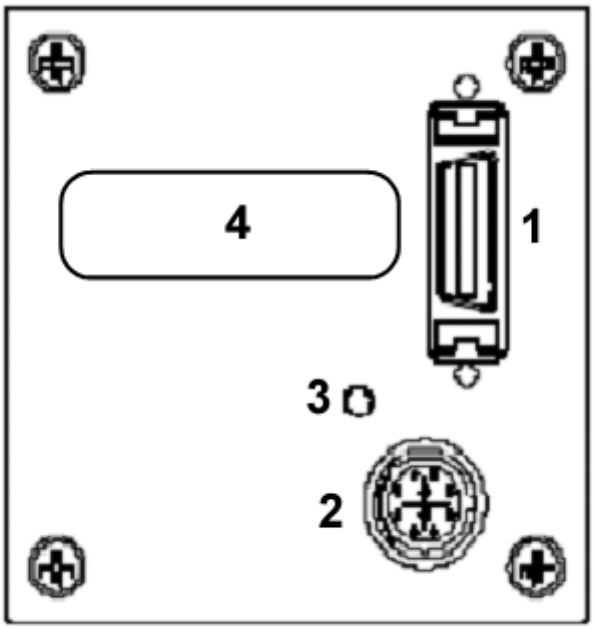


图 4. 7 IPX-1M48 相机背部面板结构图

相机的输出接口：

相机的数据输出是同标准 Camera Link 兼容的，包括 24 个数据位，3 个同步信号（LVAL，FVAL 和 DVAL），1 个参考时钟，1 个延展输入触发 CC1 和一个双向窗口接口。（图 4.8 和 4.9 为输出接口和各针对应的信号）

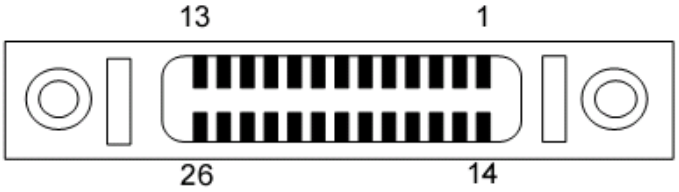


图 4. 8 相机输出接口图

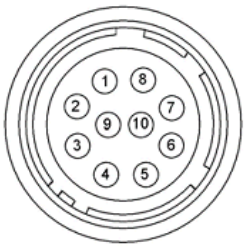
Pin Assignment

Cable Name	Pin	CL Signal	Type	Description
Inner Shield	1	Inner Shield	Ground	Cable Shield
Inner Shield	14	Inner Shield	Ground	Cable Shield
- PAIR 1	2	- X 0	LVDS - Out	Camera Link Channel Tx
+ PAIR 1	15	+ X 0	LVDS - Out	Camera Link Channel Tx
- PAIR 2	3	- X 1	LVDS - Out	Camera Link Channel Tx
+ PAIR 2	16	+ X 1	LVDS - Out	Camera Link Channel Tx
- PAIR 3	4	- X 2	LVDS - Out	Camera Link Channel Tx
+ PAIR 3	17	+ X 2	LVDS - Out	Camera Link Channel Tx
- PAIR 4	5	- X CLK	LVDS - Out	Camera Link Clock Tx
+ PAIR 4	18	+ X CLK	LVDS - Out	Camera Link Clock Tx
- PAIR 5	6	- X 3	LVDS - Out	Camera Link Channel Tx
+ PAIR 5	19	+ X 3	LVDS - Out	Camera Link Channel Tx
+ PAIR 6	7	+ SerTC	LVDS - In	Serial Data Receiver
- PAIR 6	20	- SerTC	LVDS - In	Serial Data Receiver
- PAIR 7	8	- SerTFG	LVDS - Out	Serial Data Transmitter
+ PAIR 7	21	+ SerTFG	LVDS - Out	Serial Data Transmitter
- PAIR 8	9	- CC 1	LVDS - In	Software External Trigger
+ PAIR 8	22	+ CC 1	LVDS - In	Software External Trigger
+ PAIR 9	10	N/C	N/C	N/C
- PAIR 9	23	N/C	N/C	N/C
- PAIR 10	11	N/C	N/C	N/C
+ PAIR 10	24	N/C	N/C	N/C
+ PAIR 11	12	N/C	N/C	N/C
- PAIR 11	25	N/C	N/C	N/C
Inner Shield	13	Inner Shield	Ground	Cable Shield
Inner Shield	26	Inner Shield	Ground	Cable Shield

图 4. 9 相机输出接口对应信号

相机电源接口：

相机的工作电压为+12VDC，+/-5%，1.5A，通过变压器可获得。输入电流为220VAC。电源都通过相机的电源接口连接：（图 4.10 为对应的电源接口图和对应针的信号）



Pin	Signal	Type	Description
1	Trigger In -	TTL - Input	External Trigger Input
2	Trigger In +	TTL - Input	External Trigger Input
3	GND	Power - Input	Power Ground Return
4	GND	Power - Input	Power Ground Return
5	+ 12 V	Power - Input	+ 12 V Power Supply
6	+ 12 V	Power - Input	+ 12 V Power Supply
7	Strobe Out -	TTL - Output	Strobe Light Sync Pulse
8	Strobe Out +	TTL - Output	Strobe Light Sync Pulse
9	Auto Iris +	Input	Auto Iris Feedback Input
10	Auto Iris -	Output	Auto Iris Control Output

图 4. 10 相机电源接口图

4.1.3 曝光控制

1）电子快门

相机在常态的设置模式下，曝光时间是由帧速率决定的。在强光条件下，电子快门能够准确的控制图像的曝光时间。电子快门不会影响帧速率，只是减少了电子的采集量。通过定位快门脉冲来控制曝光时间，它是和 VCCD 的脉冲相关的，电子快门脉冲的位置必须设置在整个帧的时间范围内，精度为 10 微秒，最小的遮挡时间为 50 微秒。如图 4.11 所示：

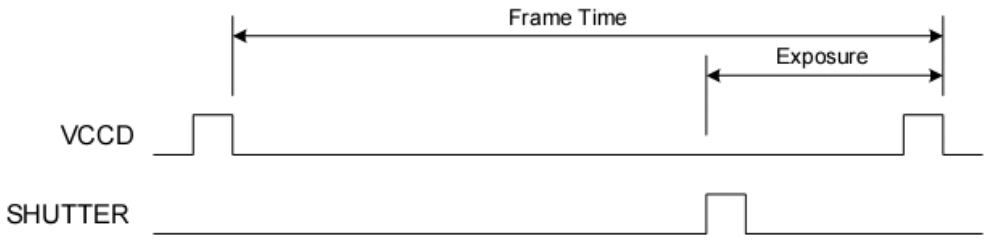


图 4.11 电子快门时序图

2）可变帧速率——可编程积分

可变帧速率模式可以提供帧速率慢于常态的速率，这样做有两个影响：一是降低了 Camera Link 接口对带宽的要求，二是增加了帧的曝光时间。在常态模式下，常规的帧速率决定了曝光时间，而对于可变帧速率的模式，可以将 VCCD 的脉冲移动至常规的 VCCD 脉冲范围之外，如图 4.12 所示：

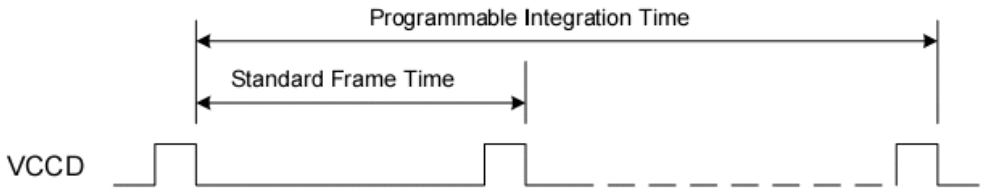


图 4.12 可编程帧速率时序图

所得的帧速率可由下面的公式计算得出：

帧速率[fps] = 1/积分时间[sec]

可以通过控制积分时间来控制帧速率，可控的范围从 2fps（0.5 秒积分时间）

至常态的帧速率，对于 IPX-1M48 相机来说，常态的帧速率为 48fps，设置的精度为 1 fps。通过输入需求的帧速率可以计算出相对应的积分时间。相机命令'sfr'（set frame rate）可以设置帧速率，命令'gce'用来获取曝光时间。还可以通过电子快门来减少曝光时间，利用相机命令'sst'(set shutter time)。

3）长积分

常积分用来扩展图像的曝光时间使其长于标准的帧时延。在常态的相机模式下，最小帧速率决定了最大曝光时间，可以通过调节 VCCD 脉冲窗口超出到常规的 VCCD 脉冲窗口来获得需求的曝光时间。如图 4.13 所示：

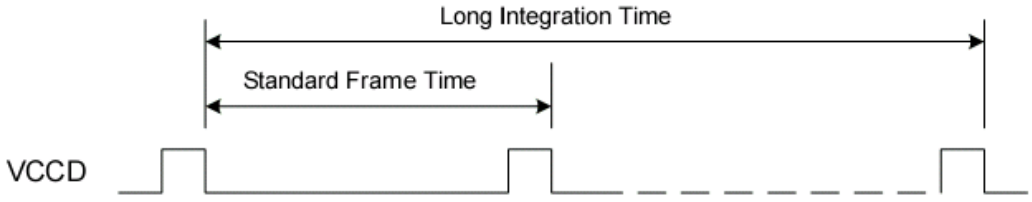


图 4. 13 长积分时序图

这种模式是和可变帧速率模式很相像，唯一的区别是在长积分模式下不能使用电子快门。积分时间可以以 10 微秒为单位增量，范围从 10 毫秒到 10 秒钟，可以用命令'sli'(set long integration) 设置长积分模式。使用长积分模式就减慢了帧速率，最终的帧速率可以通过下面的公式计算得出：

帧速率[fps]= 1/长积分时间[sec]

在长积分模式下 LED 指示灯每间隔 2 秒钟亮一下。

4. 1. 4 外部触发

在常态模式的设置下，相机是自由运行的，用外部触发模式可以使相机同外部时钟脉冲同步起来。对于外部触发来说，有两种模式：软件触发和硬件触发。在硬件触发模式下，相机通过背部的接口来接收触发信号。硬件触发的输入和其余相机硬件能明显区分开来, 如图 4.14 所示：

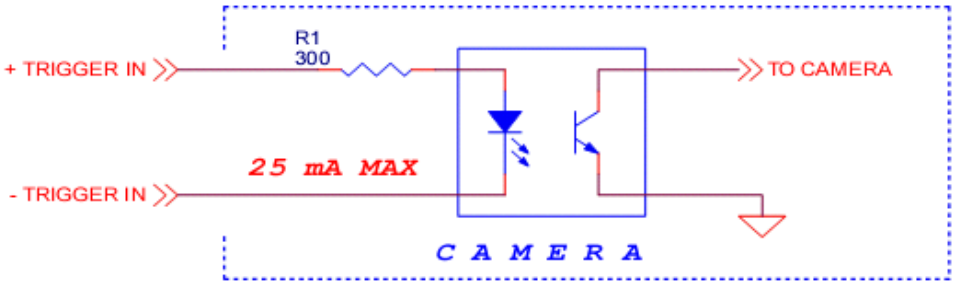


图 4. 14 硬件外部触发电路图

输入信号”+TRIGGER IN”和”-TRIGGER IN”同外部的触发源相连接。在外部脉冲的边缘产生一个在”+TRIGGER IN”和”-TRIGGER IN”之间的正的电压差，此时触发信号发送到相机。”+TRIGGER IN”和”-TRIGGER IN”之间电压的差值必须为正值，且在 3.3~5.0 伏之间。电路中总电流一定不能超过 25mA，为了限制电流，在电路中使用了一个 300 欧姆的电阻。实际的触发脉冲的持续时间不会影响积分时间，第一帧的积分时间是由预曝光寄存器中的值决定的，可以用命令’spe’(set pre-exposure)来设置其值。触发脉冲持续的最小值是 100 微秒，没有最大值的限制，但是触发脉冲的持续时间最好越短越好，尤其是一连串的脉冲被用到的情况下。

在软件触发的模式下，相机通过帧获取器来获得触发信号，帧获取器依靠相机控制信号 CC1 来实现。在这种模式下，第一帧的曝光时间可通过以下两种方式来通过编程设定：

1） 第一帧积分时间由在预曝光寄存器中的值来决定。

2） 第一帧的曝光时间由可用 CC1 触发脉冲的持续时间来决定。

不管是硬件触发还是软件触发模式都支持以下三种子模式触发。一是标准，二是快速获取，三是双曝光。当相机通过软件编程将其设置为任意一种（软件或硬件）触发模式时，相机就从自由运行模式切换到空闲模式等待外部的脉冲。下面对三个子模式作简单介绍。

4. 1. 4. 1 标准触发——可编程曝光

在标准触发模式下，相机处于空闲状态等待触发信号。一旦接收到外部触发信号，相机会清空水平和垂直寄存器，通过发送一个 5 微秒的遮蔽脉冲来清理像素并且开始积分。第一帧的曝光时间可通过编程设定为 10 微秒到 655 毫秒（以 10 微秒为单位增量）之间的值。在触发脉冲的上升沿和积分开始之间有一个 5

微秒的延迟，这是由于 5 微秒的遮蔽脉冲造成的。如图 4.15 所示：

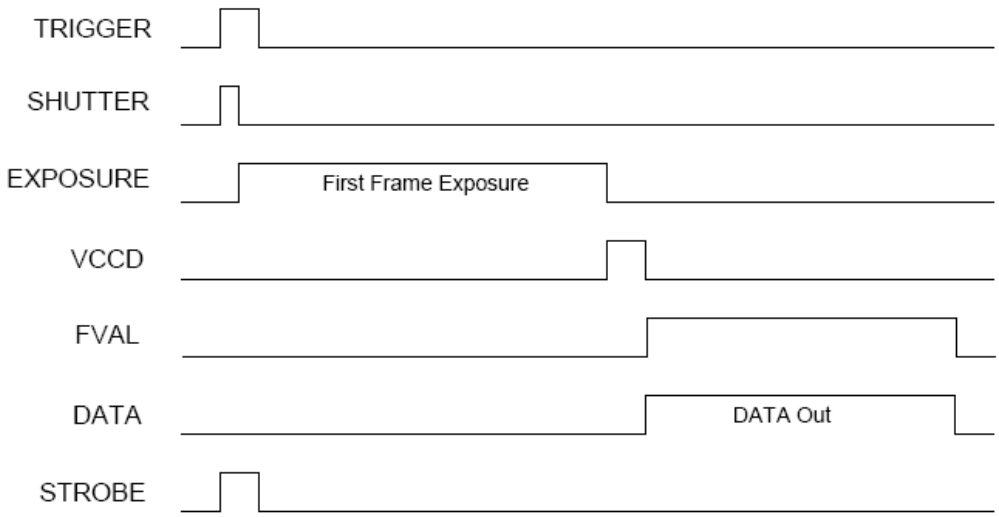


图 4.15 标准触发时序图

如果用 CC1 输入，CC1 的触发脉冲的持续时间也可以用来决定第一帧的曝光时间。当第一帧曝光结束后，相机就处于自由状态，这时帧速率是由曝光时间来决定。触发脉冲过后，相机可以继续采集图像最高达 250 帧，由编程来设定，或者也可以使相机处于自由模式。伴随着遮蔽脉冲，相机会发送一个开关脉冲，持续时间为 200 微秒，这是用来同外部的开关相同步，这个脉冲一般会在外部触发的模式出现。

4.1.4.2 快速同步触发——Rapid Capture

快速同步采集可以使相机工作在被动模式下，同时使几个相机在一个主触发信号下同步采集。这种模式也能够使相机运行在接近正常的帧速率上。如果软件和硬件模式被填加到这种模式时，相机处于空闲模式并等待从选定的源到来的触发脉冲，这个源可以是外部连接器，或者是 CC1。一旦相机接收到触发信号，相机就开始积分，一直到下一个触发信号的到来。然后这些数据被转移的到寄存器并且被读出。在读数据的同时，下一帧被曝光，如图 4.16 所示：

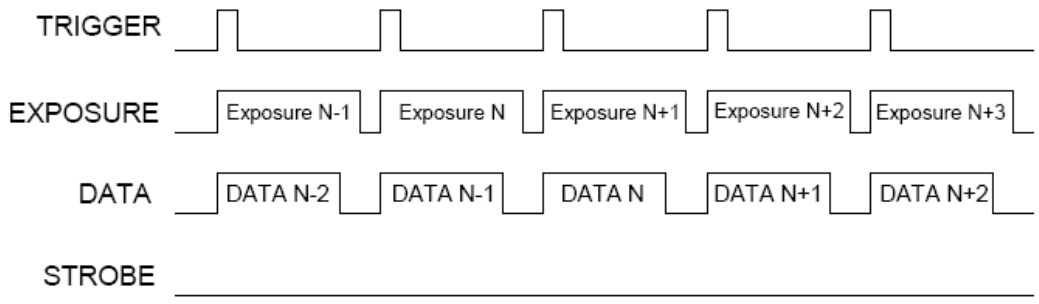


图 4. 16 快速同步触发时序图

4. 1. 4. 3 双曝光触发

双曝光模式能够应用一个触发脉冲来触发相机采集两张图像。在这种模式下相机处于空闲状态等待着一个触发信号，触发信号由选中的源发出（从外部连接器或者 CC1）。一旦接收到触发信号，相机就会清空水平和垂直寄存器，通过发送一个 5 微秒的快门脉冲。然后就开始积分。对于第一帧的曝光时间可以编程设置为 1 微秒到 65 毫秒之间（以 1 微秒为单位增量），用命令’sde’(Set Double Expose)。如果用 CC1 输入触发脉冲信号，则触发脉冲的持续时间能够决定了第一帧的曝光时间。在触发脉冲的上升沿和积分开始之间有一个 5 微秒的延迟，这是由于 5 微秒的遮蔽脉冲造成的。一点接收到触发信号，相机就开始第一帧的积分，当积分完成后，传送信息到垂直寄存器，然后开始采集第二帧图像。在采集下一帧图像的同时，读出上一帧的数据。当采集完第二幅图像曝光完成后，数据被传送到垂直寄存器然后被读出。如图 4.17 所示：

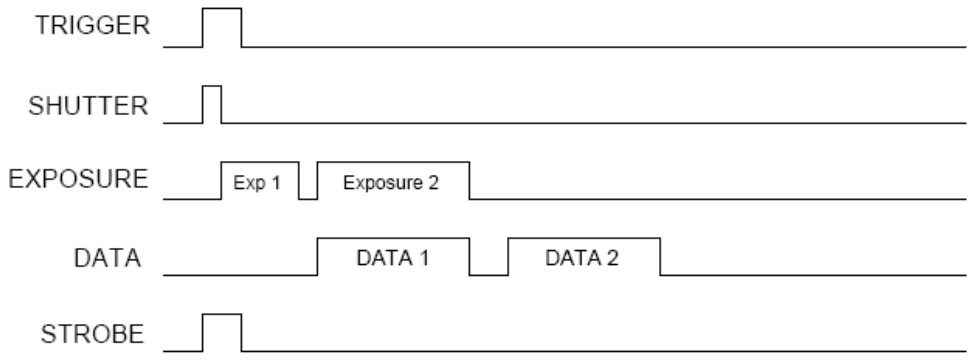


图 4. 17 双曝光触发时序图

4.1.5 增益和底值

相机两个模拟信号处理器，每个通道一个，是两个独立的 12 位 40MHz 的处理器，每个处理器包含一个微分输入采样保持放大器（SHA），离散的数字化的控制可变增益放大器（VGA），底值和一个 12 位的 ADC。可编程的内嵌 AFE 寄存器包含有独立的增益和底值调节，其中增益调节包括 1024 级（gcode 0 到 1023），底值调节包括 256 级（ocode 0 到 255）。图 4.18 所示为视频输出信号级别同增益底值的关系：

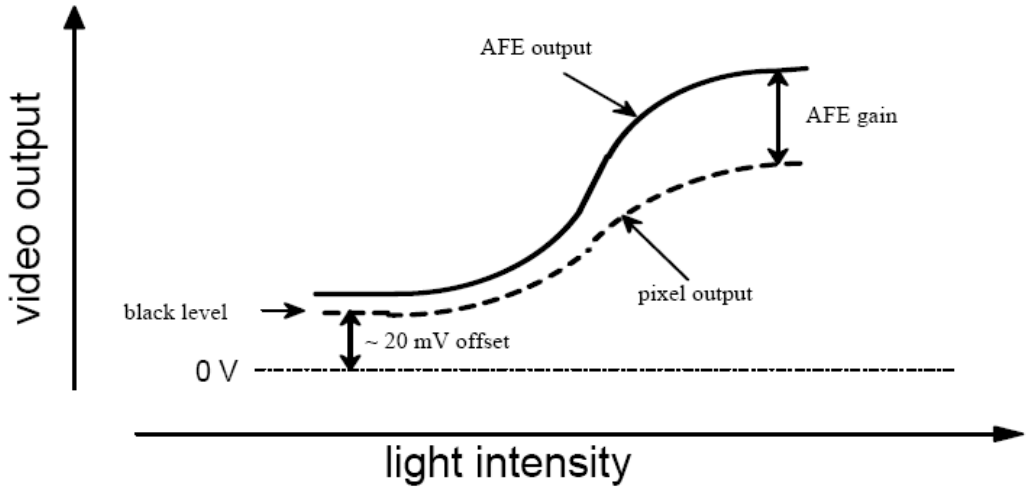


图 4.18 增益底值关系图

理论上说，底值应该设置在 0 伏，增益仅仅只是增加视频信号的幅度。由于相机的 CCD 有两个视频输出，在这两个输出之间就会存在一些偏移量的不平衡，这样的话，改变 AFE 增益就会导致偏移级别的改变，从而使两个信号之间的差异更大。为了校正两个通道之间的增益平衡，需要经常的调节两个通道的底值，整个相机的增益可以通过下面的公式计算：

$$\text{VGA Gain[dB]} = \text{FG[dB]} + 0.0351 \times \text{gcode}$$

其中对于相机 IPX-1M48 来说 FG（Fixed gain）=0

4.1.6 数据的输出格式

相机内部的 CCD 数据处理的位数为 12 位，相机输出的格式可以为 12，10，8 位，在标准位缩减过程中，截取掉最不重要的位。

12 位输出 ：如果原相机数据是 D0 到 D11，则如果相机输出设置为 12 位，则 12 位的输出被映射到 D0 到 D11。

10 位输出：将被映射到 D2 到 D11。

8 位输出： 将被映射到 D4 到 D11。

如图 4.19 所示：

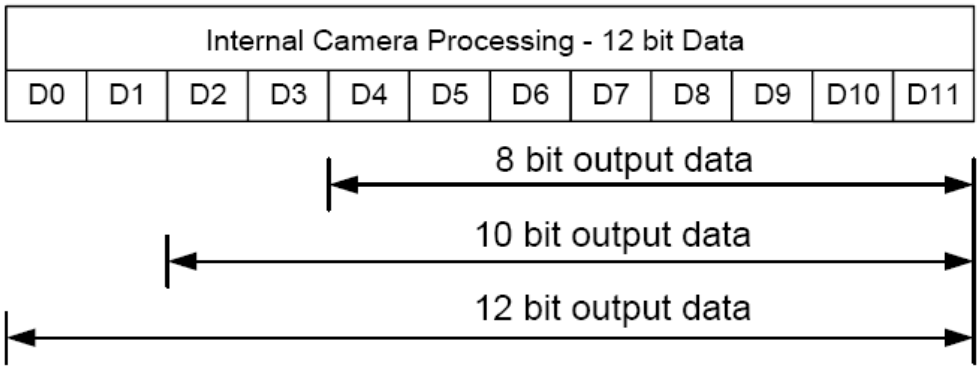


图 4.19 数据结构图

4.1.7 传输函数校正——LUT（LookUp Table）

通过定义 LUT 可以让原始像素转换成任意的值输出，如图 4.20 所示：

图 4.20 LUT 输入输出图

任意的一个 12 位的值可以转换为另一个 12 位值。相机支持两个独立的 LUT，每个包含 2048 个单元，每个单元为 12 位宽。第一个 LUT 为相机出厂时设定的标准伽玛 0.45 校正，第二个 LUT 没有预定义。两个 LUT 都可以修改，也可以

通过 Lynx 终端生成和上传新的 LUT。

4.1.8 相机的设置

4.1.8.1 Lynx Configration 及 Lynx 终端

Lynx 相机具有高度的可编程和扩展性，相机的所有属性和操作参数都可以来设置，利用简单的 ASCII 命令，通过 Camera Link 串口接口和相机通讯，来设置和控制相机。相机的所有资源包括内部寄存器和外设都可以通过串口通讯来控制和操作。串口通讯的格式是异步的 8 位数据位，1 位停止位，无奇偶校验，无握手，速率为 9600bps。通讯方式为双向的，通过输入相机命令，相机返回相应的状态和信息。对相机的设置可以通过 Lynx configuration 的图形界面，如图 4.21 所示：

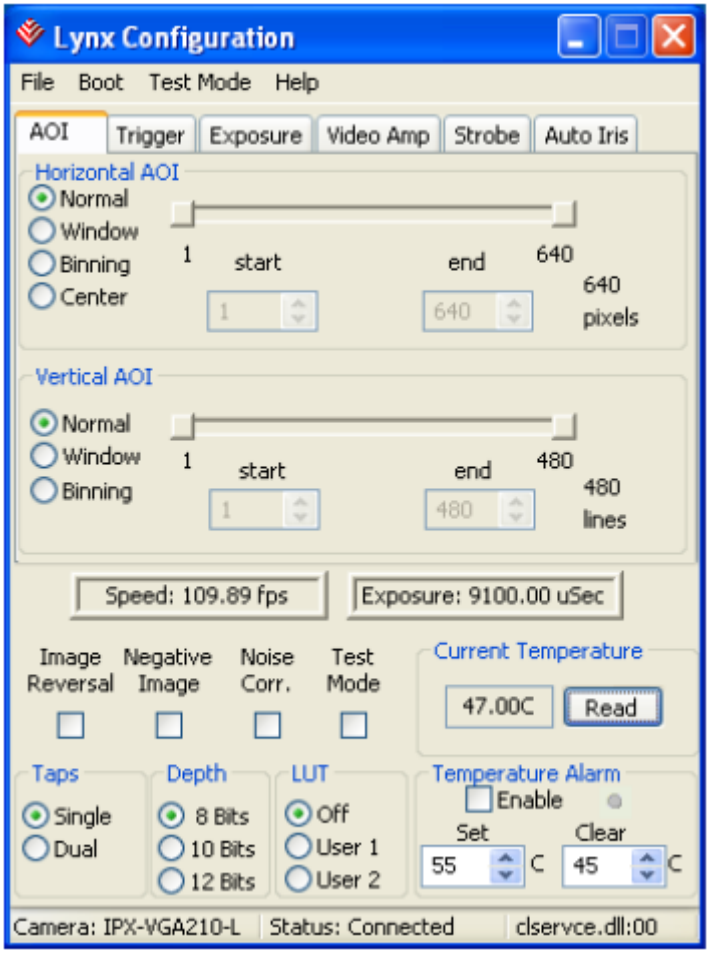


图 4. 21 LynxConfiguration 界面

也可以用 Lynx 的终端，如图 4.22 所示：

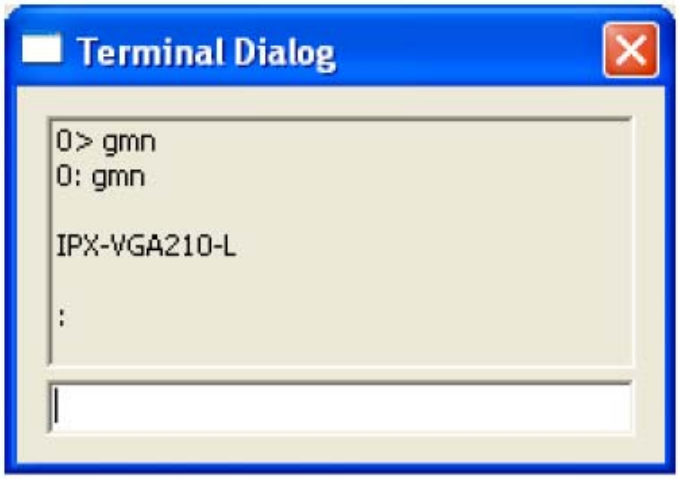


图 4. 22 Lynx 终端

还可以使用其它任意的终端。

4.1.8.2 设置的存储

相机的设置存储分为四个部分：work-space，factory-space，user-space#1，user-space#2。Work-space 存放了当相机开机时的设置，所有相机的寄存器都放置在这个空间里，可以通过输入命令获得和设置这些寄存器中存储的相机设置和当前状态的值。Workspace 是基于 RAM 的，当电源关掉后，相机中所有寄存器中的值都被清空。Factory-space 是基于 ROM 的，其中包含了相机的默认设置，是被写保护的，这个空间只允许读操作。User-space#1 和 user-space#2 是用来存放自定义的设置。一旦通电，相机就会从 factoryspace,user-space#!或者 userspace#2 中读取并加载到 workspace 中，这由放置在设置存储里的 boot control 来决定，可以使用命令 'sbf'(set boot from) 来设置。我们可以随时从 factoryspace,user-space#1 或者 userspace#2 导入到当前的 workspace 中，同样的可以随时将当前 workspace 中的设置保存到 user-space#1 或者 userspace#2 中。

4.1.8.3 命令的格式

命令字符串的格式包括一个主命令和至多两个参数。

命令串的格式为：

<command> <parm1><parm2><cr.><lf>

相机为了响应命令，除了执行命令操作外，还会返回一个响应字符串。根据命令的类型不同，相机也会返回相应的状态或者相机参数信息，状态响应反映了相机执行命令的成功或者失败，而信息响应则返回命令所要求的相机的信息。

状态响应字符串的格式为：

OK<cr><lf>：如果命令执行成功；

Error: <text><cr><lf>：如果命令执行过程中出现错误，其中<text>是对错误的描述和解释；

信息响应字符串的格式为：

<response><cr><lf>：根据不同的命令，<response>会包含有相应的相机信息。

4.2 Punix TM-1400 相机

PULNiX TM-1400 的可用帧速率为：15，30，56，98fps，电子快门的速度可达 1/16，000sec，可以通过外部脉冲控制异步传输。该相机拥有动态的域控制函数，通过设置外部可选的 LUT（look-up table）来转换 10 位输入到 8 位输出。相机拥有 8 位的，RS-644 数字信号输出，来同外部的图像处理系统来连接。所有的主要功能都通过 RS-232C 来控制。相机有 Camera Link 的输出模式，其主要功能也是通过串口通信的 Camera Link 来控制的[9]。

4.2.1 相机特性

图像的分辨率为 1392×1040 的有效像素，4.65×4.65 平方微米像素，相机的输出为 8 位。

相机还可实现高分辨率的长时间曝光图像，曝光时间可以比常规的扫描时间 1/30 秒要长，这种特性在环境光很暗的条件下很有用处。

4.2.2 相机的接口

图 4.23 为相机的组装图：

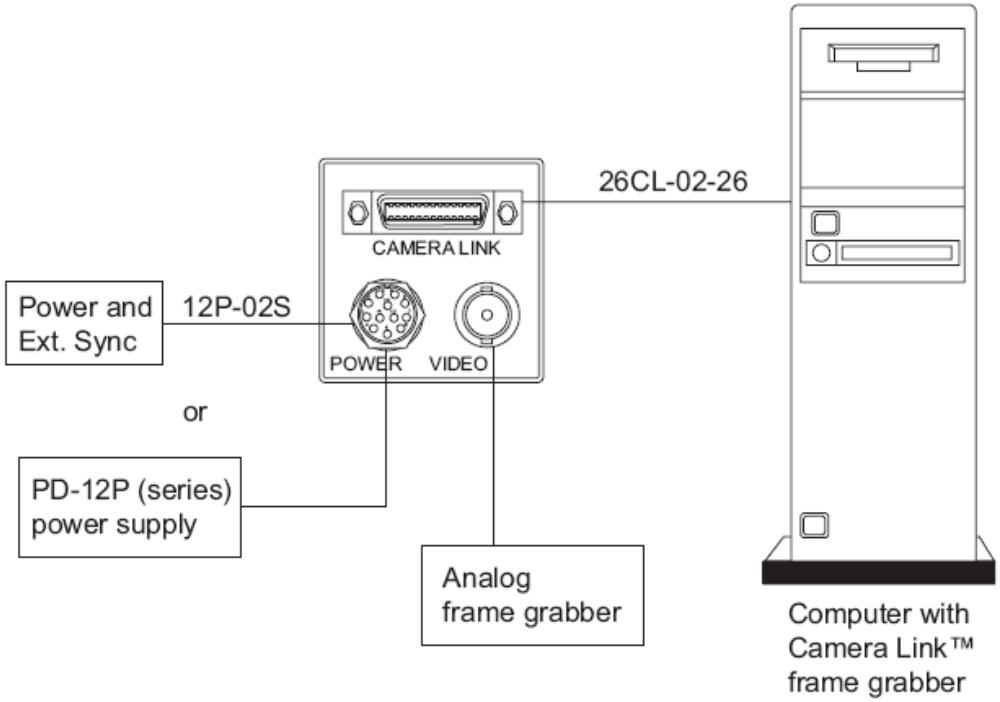


图 4. 23 相机组装图

图 4.24 为相机背部电源连接：

Pin	Description	Pin	Description
1	GND (power)	7	VD in
2	+12V DC	8	Reserved
3	GND (analog)	9	HD in
4	Video out	10	N/C
5	GND (digital)	11	Integration Control/ROI
6	VINIT in	12	N/C

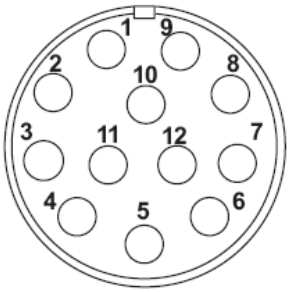
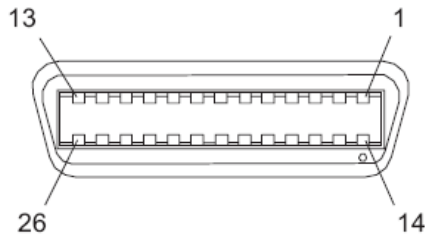


图 4. 24 相机背部电源

图 4.25 为 Camera Link 连接器：



Camera Link Connector MDR 26-pin Connector 10226-6212VC					
Pin #	**Description**	**I/O**	**Pin #**	**Description**	**I/O**
1	GND		14	GND	(Shield)
2	Tx OUT 0-	Out	15	Tx OUT 0+	Out
3	Tx OUT 1-	Out	16	Tx OUT 1+	Out
4	Tx OUT 2-	Out	17	Tx OUT 2+	Out
5	Tx CLK OUT-	Out	18	Tx CLK OUT+	Out
6	Tx OUT 3-	Out	19	Tx OUT 3+	Out
7	SerTC+	In	20	SerTC-	In
8	SerTFG-	Out	21	SerTFG+	Out
9	VINIT- (CC1-)	In	22	VINIT+ (CC1+)	In
10	INTEG+ (CC2+)	In	23	INTEG- (CC2-)	In
11	EX_HD-	In	24	EX_HD+	In
12	EX_HD+	In	25	EX_HD-	In
13	GND		26	GND	

图 4. 25 Camera Link 针脚明细图

4.2.3 相机控制

4.1.3.1 连续扫描

标准的电视是 525 行的隔行扫描，每个区域的的隔行扫描频率为 60 HZ，完成一帧（两个区域）的扫描频率为 30HZ，由于是隔行扫描，不考虑水平分辨率，CCD 相机的垂直分辨率被限定在 350 电视行，当电子快门被应用时，CCD 每次曝光只能保留一个域的电量，所以电子快门相机的垂直分辨率仅仅是 244 个电视行。对于 TM-1400CL 来说，它采用了一种稳定的连续扫描方案，从顶到底顺序扫描所有的行，并且帧速率为 30HZ，这样就可产生稳定的图像，图像的垂直分辨率可达 1040 行。

4.1.3.2 相机异步重置模式

异步重置通过接受外部水平驱动脉冲ＨＤ（horizontal drive）来锁定相位。当 VINT 负脉冲（5V）到来时，在 VINT 的下降沿且 HD 的高位时，会有一个放电脉冲（discharge pulse），相机会重置扫描并且通过放电清除 CCD 中存储的电荷，然后相机进入曝光，当传输门脉冲（transfer gate）到来时，曝光结束，相机通过连续扫描系统对数据进行输出操作。（如图 4.26 所示）

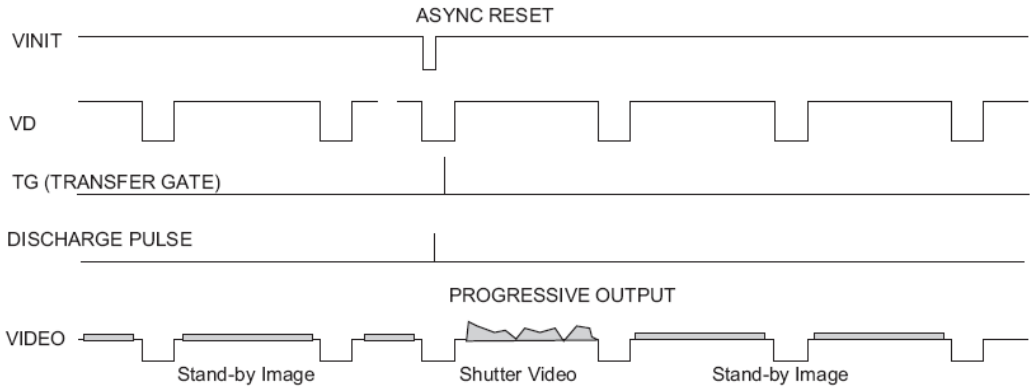


图 4. 26 异步重置时序图

对相机曝光时间的控制，即对电子快门的控制有两种模式，第一种模式是通过 VINIT 的脉冲宽度来决定曝光时间，通过设置模式 9 来实现，由外部事件触发来产生一个 VINIT 信号，则这个内部重置脉冲会锁定 HD，在第一个 HD 的高位同时在 VINIT 的下降沿会产生一个放电脉冲，通过此脉冲来清理残留在 CCD 中的图像，此后相机开始曝光，在 VINIT 的上升沿产生一个内部 VD（vertical drive），此时快门的速度是和外部触发脉冲的宽度一致的，但是曝光时间会延迟一个 HD。（如图 4.27 所示）

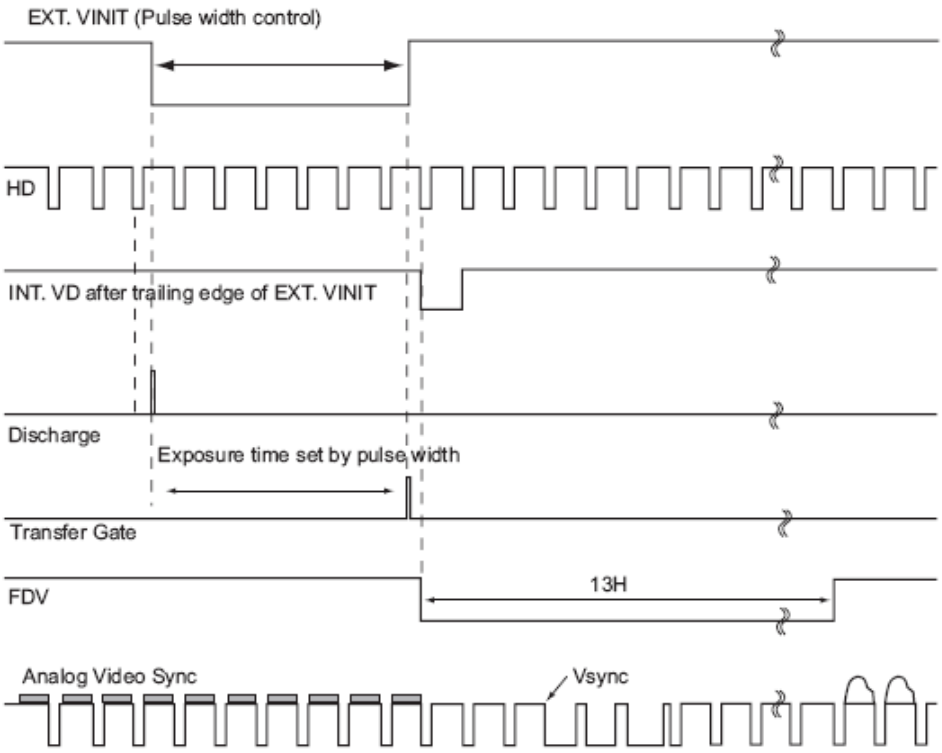


图 4. 27 异步重置脉冲宽度模式时序图

另一种模式是内部快门时间控制模式，通过设置相机模式为 1 到 8，当外部触发到来时，内部 VD 锁定 HD，通过延迟 VD 建立快门的时间区域。（如图 4.28 所示）

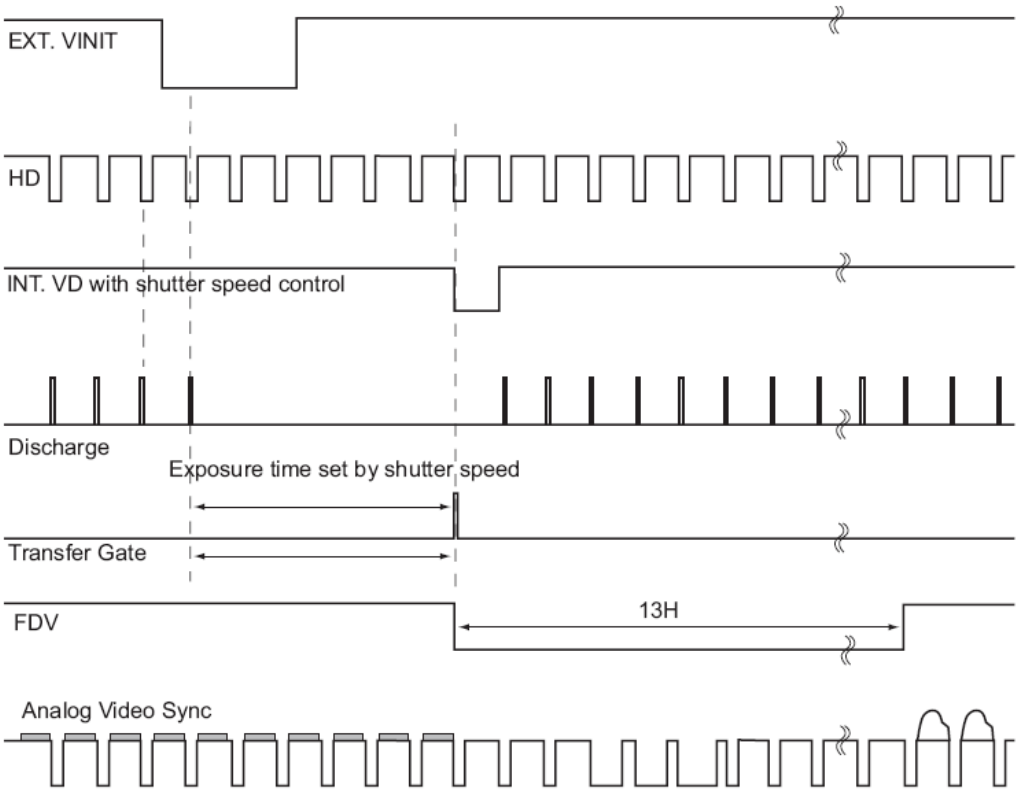


图 4. 28 相机快门控制模式时序图

4. 2. 4 相机命令

传输相机命令是通过串口通信利用 Camera Link 来完成的。命令的格式，一般以字符“：”来开头，以<CR>为结尾。如果要将相机设置为异步脉冲宽度模式，发送的命令为：“：SA9<CR>”，图 4.29 为相机的命令列表及其参数：

	First Character	Second Character	Third Character	Response	Functions
1	"S" (Shutter)	"M" (Manual)	"0" - "9" Mode	ACK	Manual Shutter Mode
		"A" (ASYNC)	"0" - "8" Mode	ACK	Async Shutter Mode
			"9" (Pulse Width Mode)	ACK	Async Pulse Width Mode
		"X"	"000" - "419"	ACK	Direct Shutter Mode
2	"G" (Gain)	"M"	"00" - "FF"	ACK	Gain Control
3	"V" (A/D Vref)	"T" (Top)	"00" - "FF"	ACK	Vtop reference setting
		"B" (Bottom)	"00" - "FF"	ACK	Vbtm reference setting
4	"W" (Write)	"P" (Page)	"0" - "6"	ACK	Write current setting to Page EEPROM
		"U" (User)	"A" - "D"	ACK	Write current setting to User EEPROM
		"S" (System)	"A" - "D"	ACK	Write current setting to System EEPROM
5	"L"	"P" (Page)	"0" - "6"	ACK	Restore setting from Page EEPROM
		"U" (User)	"A" - "D"	ACK	Restore setting from User EEPROM
		"S" (System)	"A" - "D"	ACK	Restore setting from System EEPROM
		"N" (kNee)	"0" - "9"	ACK	Load Preset Knee Table
6	"R" (Report)	"P" (Page)	"0" - "6"	ACK	ACK + "P" + ("9" - "F") + 16 bytes
		"U" (User)	"A" - "D"	ACK	ACK + "U" + ("A" - "D") + 6 bytes
		"S" (System)	"A" - "D"	ACK	ACK + "S" + ("A" - "D") + (6 bytes)
		"R" (Current)		ACK	ACK + "RR" + 16 bytes
		"X" (Execute)		ACK	Set Camera with loaded data
		"D" (Date)		info	Report CPU program version
7	"T" (Table)	"N" (kNee)	X1 + Y1 + X2 + Y2	ACK	(X1, Y1) coordinate for knee 1 X1, Y1, X2, Y2: "00 - FF" (X2, Y2) coordinate for knee 2
		"M" (Gamma)		ACK	
		"L" (Linear)		ACK	
		"C" (Switch A, B Table)	"0" or "1"	ACK	
8	"N"	"0" (Normal)		ACK	Normal Scan Formal
		"3" (Binning)		ACK	Double Speed Binning

图 4. 29 相机命令图表

一个字节由两个 ASCII 码值组成 0x3A 就是“3A”或者 0x3341，

<CR>=0x0D　　命令或者响应的终止符

<ACK>=0x06　　接收命令成功

<NAK>=0x15　　接受命令失败

如果发送命令“RR”，则返回的值为“RR”+ 16 字节 + <CR>，其中的 16 字节的格式如图 4.30 所示：

Byte 1	MGCL (1 byte)	--	CDS Gain
Byte 2	Vtop(1 byte)	--	A/D reference voltage Top
Byte 3	Vbtm(1 byte)	--	A/D reference voltage Bottom
Byte 4	XA1 (1 byte)	--	X-Coordinate of right knee for table A
Byte 5	YA1 (1 byte)	--	Y-Coordinate of right knee for table A
Byte 6	XA2 (1 byte)	--	X-Coordinate of right knee for table A
Byte 7	YA2 (1 byte)	--	Y-Coordinate of right knee for table A
Byte 8	XB1	--	X-Coordinate of left knee for table B
Byte 9	YB1	--	Y-Coordinate of left knee for table B
Byte 10	XB2	--	X-Coordinate of left knee for table B
Byte 11	YB2	--	Y-Coordinate of left knee for table B
Byte 12	FUNCFLAG1 (1 byte)	--	function flag #1
Byte 13	FUNCFLAG2 (1 byte)	--	function flag #2
Byte 14	SHTRNUM (1 byte)	--	current shutter number
Byte 15, 16	SHTRVAL (2 byte)	--	manual/direct shutter value

图 4. 30 16 字节结构图

4.3 三通道望远镜中三个相机的同步采集

在三通道太阳磁场望远镜中使用了三个 CCD 相机，其中两个为 LYNX 系列的 IPX-1M48 相机，另一个为 Pulnix 的 TM-1400 相机，为了获得太阳不同层次的数据，每个相机需要采集多幅图像，累计加和来获得最终数据，在每采集一幅图像之后，有一个高压反转加载给滤波器，然后再采集下一幅图像，在采集图像时要保证三个相机都采集完成，然后高压反转，再继续采集下一幅图像，直到采集数目达到指定数值。流程如图 4.31 所示：

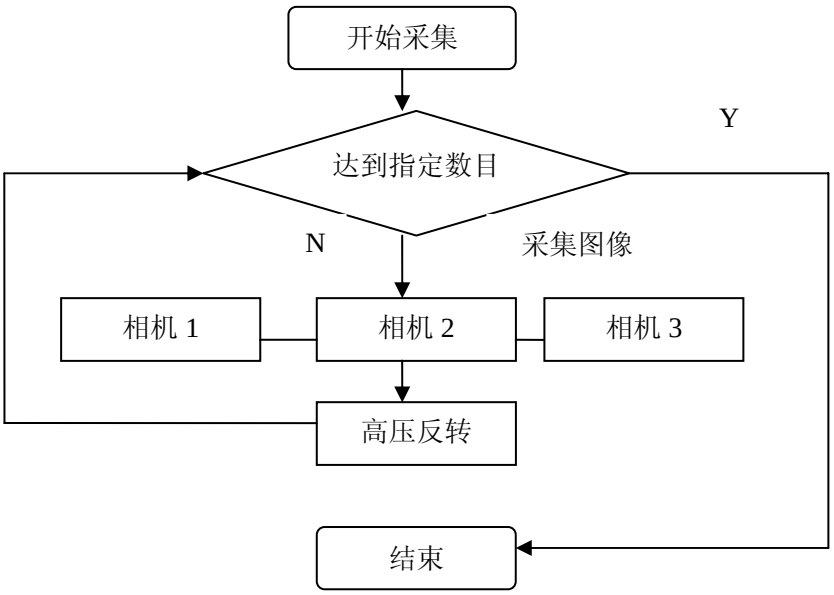


图 4. 31 相机同步采集流程图

实现方案如下：通过创建管理类来对三个相机进行控制，开启一个线程来采集图像，采用 Event 内核事件对三个相机的采集图像进行同步，Event 内核事件可以保证在某个条件没有达到之前，不再往下继续执行，程序会处于挂起状态，直到所有条件都满足后，程序方可继续运行，这样就可以保证当三台相机的采集一幅图像的条件满足后，再执行高压反转指令，然后再进行下一幅图像的采集，保证三台相机的同步采集。

以下为单开启的线程中同步三个相机采集的源代码：

```
while(1)
{
    WaitForMultipleObjects(countCon,hd,true,INFINITE);//等待三个CCD同时开始采集
    if(addstate)
    {
        switch(addstate)
        {
        case 1:
            manager ->m_pIPX1->isaddimage=false;
            manager ->m_pIPX2->isaddimage=false;
            manager ->m_pPulnix->isaddimage=false;
            addstate++;
            addcnt=manager->Runnum;
```

```
        break;
case 2:
        addstate++;
        switchkdp(0x70);                    //高压反转
        break;
case 3:                                     //丢弃掉前几幅图像
        addstate++;
        switchkdp(0x30);
        break;
……
case 6:
        manager ->m_pIPX1->isaddleft=true;      //高压和反高压替换采集
        manager ->m_pIPX2->isaddleft=true;
        manager ->m_pPulnix->isaddleft=true;

        manager ->m_pIPX1->isaddright=false;
        manager ->m_pIPX2->isaddright=false;
        manager ->m_pPulnix->isaddright=false;

        addstate++;
        switchkdp(0x70);

        addcnt--;
        if(!addcnt)
        {
            addstate=0;
        }
        break;
case 7:
        manager ->m_pIPX1->isaddleft=false;
        manager ->m_pIPX2->isaddleft=false;
        manager ->m_pPulnix->isaddleft=false;

        manager ->m_pIPX1->isaddright=true;
        manager ->m_pIPX2->isaddright=true;
        manager ->m_pPulnix->isaddright=true;

        addstate--;
```

```
                switchkdp(0x30);
                break;
            }
        }
        else
        {
                manager ->m_pIPX1->isaddleft=false;
                manager ->m_pIPX2->isaddleft=false;
                manager ->m_pPulnix->isaddleft=false;

                manager ->m_pIPX1->isaddright=false;
                manager ->m_pIPX2->isaddright=false;
                manager ->m_pPulnix->isaddright=false;
                switchkdp(0x0);
                break;
        };
        manager->m_pIPX1->trigger(manager->m_pIPX1,"1",false);//采集图像
        manager->m_pIPX2->trigger(manager->m_pIPX2,"1",false);
        manager->m_pPulnix->trigger(manager->m_pPulnix,"1",false);
        ResetEvent(hd[0]);
        ResetEvent(hd[1]);
        ResetEvent(hd[2]);
    }
```

4.4 小结

本章主要从相机的特性，相机的外部接口，工作模式，以及相机命令来对这两部相机做了全面的分析和介绍，由于对太阳图像采集的要求为多张图像累加保存，因此选择相机的软件触发模式，通过软件触发单个脉冲，来触发相机采集单张的图像，利用操作系统内核事件来保证三台相机的同步采集，通过累积加和来完成一组图像数据的采集任务。

第五章 三通道望远镜终端系统对相机类的模块化封装

本章首先对软件 Sapera LT 作简单介绍[10]，接着讨论了三通道望远镜远程观测终端系统中对相机类的封装，使得望远镜终端系统模块化，当望远镜更新或者增加新的相机时，可以快速方便的更新终端系统，同时也可以大大提高新的望远镜终端系统开发的速度。

Camera Link 的标准是由数家工业摄影机及影像卡大厂共同制定出来的，它有业界统一的 API[11],本系统采用 Camera link 接口的软件编程，通过计算机和采集卡的通信完成对相机的控制。如图 5.1 所示：

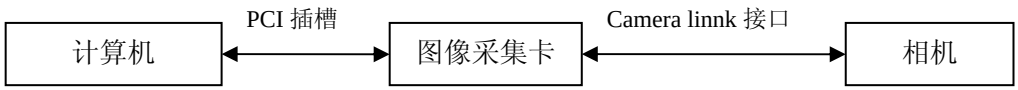


图 5.1 相机控制图

相机通过图像采集卡和电脑通讯，三通道望远镜所使用的图像采集卡为 coreco 公司的 X64-CL Dual ，使用的软件是 Sapera。

5.1 Sapera LT 简介

Sapera LT 是为了进行图像处理和机器视觉而专门编写的基于 C++的类库。其下又可以分为两大类，一个基类，另一个是基于 GUI 的类库。

Sapera 软件支持利用 C，C++ ，以及 ActiveX 等编程环境的应用，对应的 Sapera LT 有三个 API，如图 5.2 所示:

1. Sapera++ class (基于 C++语言)
2. Sapera Standard API（基于 C 语言）
3. Sapera ActiveX Controls

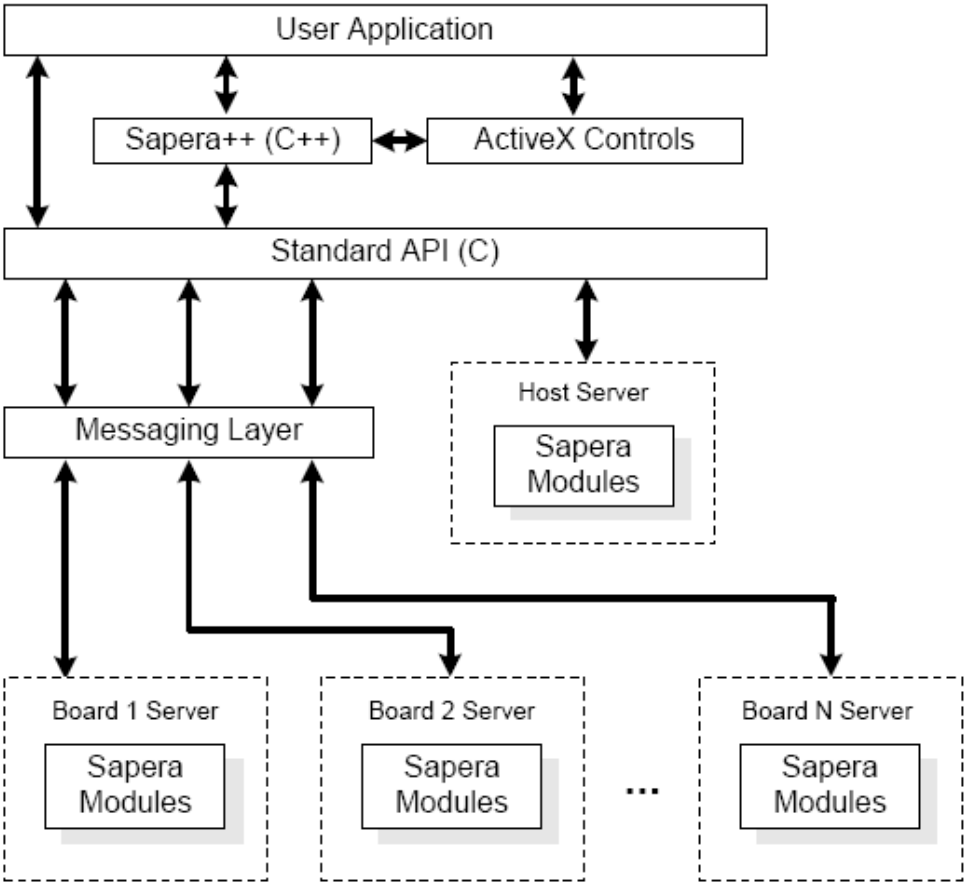


图 5. 2 应用架构图

在三通道望远镜中主要利用了基于 C 语言的标准 API 编程来控制相机。

5. 1. 1 术语模块架构图

服务器(server)：通常一个 Coreco 的图像卡即为一个服务器。

静态资源（static resource）：同物理器件相关联的资源称为静态资源，如获取资源（acquisition resource），显示资源(display resource)，和处理资源(processor resource)等。

动态资源（dynamic resource）：动态资源是数据存储的一个抽象表示，比如：缓存（buffer），lookup table，或者是同静态资源关联的数据存储空间。同静态资源相比，动态资源是独立于物理器件的。

模块（module）：模块是一系列控制动态和静态资源功能的集合。

5.1.2 模块架构图

图 5.3 为 Sapera 类库的模块架构。其中标准的长方形（虚线之上）表示了动态资源模块，而圆角的方形（在虚线之下）表示了静态资源模块。

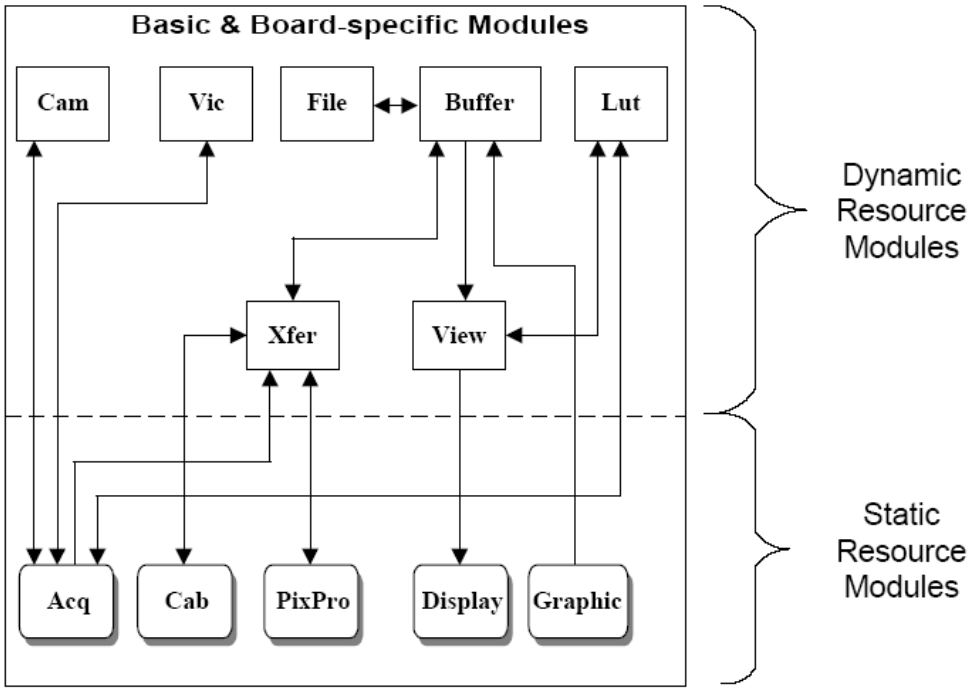


图 5.3 模块架构图

Camera Module（Cam）是一个动态资源，用它来存储和相机有关的参数，如视频格式，输入位数，以及像素时钟等。

Video—Input Conditionding Module（VIC）也是一个动态资源，用它来存储数字设备的参数，比如亮度（brightness），对比度（contrast），饱和度（saturation）等。

Buffer Module（Buffer）是一个动态资源，包含了可派生的缓存，可以用来存放一维的矢量和而维的图像。

LookUp Table Module（Lut）是一个动态资源，利用这个模块可以控制和生成一个 LookUp table。

Transfer Module（Xfer）是一个动态资源，用它来建立一个连接，完成从源到目的资源之间来传送数据。

View Module（View）是一个动态资源，用它来建立在缓存和显示模块之间的连接，通过这个连接来传送数据。

File Module（File）是一个动态资源，用来交换缓存和文件中的图像。其中支持图像的格式有：TIFF，BMP，RAW， JPEG，AVI 等。

Acquisition Module（Acq）是静态资源，用它来读取和写入采集图像的参数，同 Cam 和 VIC 模块协同工作，同时 Transfer 模块要和它进行同步来控制采集图像进程的启动和停止。

Display Module （Display）是静态资源，用来控制计算机显卡上显示部件。用它可以读取和写入显示的参数。

Graphic Module（Graphic）是静态资源，用来控制一个图像或进程部件，这个部件在 CPU 上。用它可以在一个缓存中画图形，矢量，文字等。

5.3 关于 Sapera LT 基于 C 的标准 API 编程

三通道望远镜的服务器端是通过基于 C 的标准 API 编程实现的，下面着重介绍 Sapera LT 基于 C 的标准 API 编程。

5.3.1 API 的命名规则

函数：API 的函数遵循标准的命名规则，首先，所有的 API 函数都以“Cor”开头，接下来紧跟着上面介绍的模块的名字，然后是函数的名字。所有的函数返回一个错误代码，下面为其语法结构：

CORSTATUS Cor<module name><funtinon name>(…)

比如:

```
CORSTATUS status;                    //status code
stadus = CorBufferClear(…)       //Clear function of buffer module
stadus = CorXferStart(…)             //Start function of transfer module
```

句柄：所有的 API 函数都和服务器或者模块的句柄相关联。服务器的句柄为 CORSERVER,句柄命名的语法为：

COR<module name>

比如：

```
CORSERVER hserver;                   //Handle to a server
```

```
CORBUFFER hbuffer;                //Handle to a buffer
CORACQ hAcq;                      //handle to an acquisition
```

性能和参数：每个资源都有一系列的性能和参数，他们遵循以下语法：

对于一个性能: COR<module name>_CAP_<capability name>

对于一个参数: COR<module name>_PRM_<capability name>

对于他们的值：COR<module name>_VAL_ <capability name>_<value deccreption>

比如：

```
CORACQ_CAP_CHANNEL //Capability channel of acquisition module
CORACQ_PRM_CHANNEL      //Parameter channelof acquisition module
CORACQ_VAL_CHANNEL_DUAL   //Dual channel value for acquisition Module
```

5.3.2 句柄的操作

在 Sapera 中，只能通过访问 API 函数来访问资源，因此服务器和资源都有一个句柄，句柄就是一个包含进入资源内部的必要信息的结构体。获得一个 资源的句柄须要两个步：

首先获得服务器的句柄，然后获得资源的句柄。

5.3.2.1 服务器句柄

在 Sapera 中有三种途径获得一个服务器的句柄：

1.对于应用程序正在运行的默认服务器，通过下面的函数来获得其句柄：

```
CORSERVER hServer;              //declare a server handle
 //get the default server handle
 hServer = CorManGetLocalServer();
```

2.也可以列举出所有可用的 Sapera 服务器，从 0 到 nServer-1 来标示其索引，其中 nServer 是通过 API 来获得的服务器数目。

```
CORSTATUS status;          //Declare status code
UINT32 nCount;             //Declare a server count
```

```
UINT32 nIndex;                  //Declare a server index
char szName[64];          //Declare a character string for returned name
CORSERVER hServer   ;      //Declare a server handle
//Get the server count
status = CorManGetServerCount(&nCount);
//Get the server handle from an index
status = CorManGetServerByIndex(nIndex,szName,&hServer);
```

3.可以通过指定服务器的名字来获得：

```
CORSTATUS status;                  //Declare status code
CORSERVER hServer;                 //Declare a server handle
//Get the server handle by specifying a name
status = CorManGetServerByName("Bandit_II_1",&hServer);
```

当完成使用操作后，用下面的函数来释放服务器句柄：

```
CORSTATUS status;                  //Declare status code
CORSERVER hServer;                 //Declare a server handle
//Release the specified server handle
status = CorManReleaseServer(hServer);
```

5.3.2.2 资源句柄

对于静态资源，是和服务器上的器件相关的，因此对于不同的服务器就有不同数目的静态资源，每个静态资源包含了一个资源计数的函数。

```
CORSTATUS status;           //Declare status code
CORSERVER hServer;          //Declare a server handle
UINT32 nAcqCount;           //Declare a acquisition count
UINT32 nDisplayCount;//Declare a displsy count
//Get server handle
…
status = CorAcqGetCount(hServer,&nAcqCount);              //Get acquisition count
status = CorDisplayGetCount(hServer,&nDisplayCount);//Get display count
```

这样就可以得到一个资源的句柄，其索引值从 2 到 nxxxCount-1，当不再使

用一个句柄时，必须将其释放。

```
CORSTATUS status;          //Declare status code
CORSERVER hServer;         //Declare a server handle
CORACQ hAcq;               //Declare an acquisition handle
CORDISPLAY hDisplay;       //Declare an display handle
//Get server handle
…
//Get resource handle
status = CorAcqGetHandle(hServer,0,&hAcq);
status = CorDisplayGetHandle(hServer,0,&hDisplay);
//Use them
…
//Release handles when finished
Status = CorAcqRelease(hAcq);
Status = CorDisplayRelease(hDisplay);
```

对于动态资源是和硬件器件无关的，所以每个动态资源都有创建的能力，下面为创建一个 buffer 和 lookup table 资源的代码：

```
CORSTATUS status;          //Declare a status code
CORSERVER hServer;         //Declare a server handle
CORBUFFER hbuffer;         //Declare a buffer handle
CORLUT hLut;               //Declare a LUT handle
//Get server handle
…
//Create resource handle
status = CorBuffNew(hServer,640,480,CORBUFFER_VAL_FORMAT_UINT8,
                                0,&hBuffer);
status = CorLutNew(hServer,256,CORLUT_VAL_FORMAT_UINT8,&hLut);
//Use them
…
//Free handles when finished
status = CorBufferFree(hBuffer);
status = CorLutFree(hLut);
```

5.3.3 性能和参数（Capability and parameters）

性能就是用一个值或者一组值来描述一个资源的功能。他们都是只读的，可以通过 Cor<module name>GetCap 函数来获得资源模块的性能，其原型如下：

CorxxxGetCap(CORxxx handle,UINT32 cap,void *value)

handle: 资源的有效句柄

cap : 资源的有效性能

value：一个合适大小的缓存，用来存放性能的值。

参数表示了一个资源的特征，可以用来读写，或者是只读的。一个资源的参数可用函数 Cor<module mame>GetPrm 函数来读取，其原型为：

CorxxxGetPrm(CORxxx handle,UINT32 prm,void *value)

handle : 资源的有效句柄

prm : 资源的有效参数

value: 合适大小的缓存，用来存放参数的值

可以用函数 Cor<module name>SetPrm 和 Cor<module name>SetPrmEx 来设置资源的参数，他们的原型如下：

CorxxxSetPrm(CORxxx handle,UINT32 prm,UINT32 value)

CorxxxSetPrmEx(CORxxx handle,UINT32 prm,const void *value)

handle : 资源的有效句柄

prm: 资源的有效参数

value: 合适大小的缓存，用来存放参数的值

其中“Ex”函数用来写入参数大于四个字节的值。

5.3.4 获取图像

要获取图像，需要下面三个模块：Acquisition,Buffer,Tranfer，获取实时的图像需要两个文件来配置硬件资源,即图像采集卡,一个CAM 文件和一个 VIC 文件。前者定义了相机的特性，后者定义了相机和采集卡怎么被应用。当文件 CAM 和 VIC 初始化完成后，读取文件中的参数来，根据参数来创建相应的缓存。在开始传输前，必须创建一个缓存和采集卡之间的路径，当结束传输之后，必须调用函数 CorXferWait 等待传输过程完全结束。

以下为获取实时图像的代码：

```
//Transfer callback function: called each time a complete frame is transferred
CORSTATUS CCONV XferCallback(void *context,UINT32 eventType,UINT32
                                        eventCount)
{
    //Display the last transferred frame
    CorViewShow（*(CORVIEW*) context）;
    Return CORSTATUS_OK;
}
Main()
{
    CORSTATUS status;              //error code
    CORSERVER hSystem;      //system server handle
    CORSERVER hBoard;              //Board server handle
    CORCAM hCam;            //CAM handle
    CORVIC hVic;                   //VIC handle
    CORACQ hAcq;                   //Acquisition handle
    CORBUFFER hBuffer;             //Buffer handle
    CORXFER hXfer;                 //Transfer handle
    CORVIEW hView;          //View handle
    CORDISPLAY hDisplay;           //Display handle
    UINT32 width,height,format;

    //Get server handles (system and board)
    hSystem = CorManGetServer();
    status = CorManGetServerByName(“Bandit_II_1”,&hBoard);

    //Get acquisition handle
    status = CorAcqGetHandle(hBoard,0,&hAcq);//0=first instance

    //Create CAM/VIC handles
    status = CorCamNew(hSystem,&hCam); //Camera
    status = CorVicNew(hSystem,&hVIC);//Video-Input-Conditioning

    //Load CAM/VIC parameters from file into system memory
    //the acquisition hardware is not initialized at this point
    status = CorCamLoad(hCam,”rs170,cca”);
```

```
status = CorVicLoad(hVic,”rs179.cvi”);

//Download the CAM/VIC parameters to the acquisition module
//the acquisition hardware is now initialized
status = CorAcqSetPrms(hAcq,hVic,hCam,FALSE);

//Create a buffer compatible to acquisition
status=CorAcqGetPrm(hAcq,CORACQ_PRM_SCALE_HORZ,&width);
status=CorAcqGetPrm(hAcq,CORACQ_PRM_SCALE_VERT,&height);
status=CorAcqGetPrm(hAcq,CORACQ_PRM_OUTPUT_FORMAT,
                                &fomat);
status=CorBufferNew(hSystem,width,height,format,
CORBUFFER_VAL_TYPE_SCATTER_GATHER,&hBuffer);
//Create a transfer handle to link acquisition to buffer
status = CorXferNew(hBoard,hAcq,hBuffer,NULL,&hXfer);
//Register a callback function on “end-of-Frame” events
status = CorXferRegisterCallback(hXfer,
                                CORXFER_VAL_EVENT_TYPE_END_OF_FRAME,
                                XferCallback,(void*)&hView);
//Activate the connection between acquisition and buffer
status = CorXferConnect(hXfer);

//Get display handle
status = CorDisplayGetHandle(hSystem,0,&hDisplay);

//Create a continuous transfer(live grab)
status = CorXferStart(hXfer,CORXFER_CONTINUOUS);

printf(“press any key to stop grab\n”);
getch();//wait until a key has been hit

//Stop the transfer and wait (timeout = 5sec)
status = CorXferStop(hXfer);
status = CorXferWait(hXfer,5000);

//Break the connection between acquisition and buffer
status = CorXferDisconnect(hXfer);
```

```
    printf("press any key to terminater\n");
    getch();

    //release handles when finished (in the reverse order)
    CorViewFree(hView);
    CorDisplayRelease(hDisplay);
    CorXferFree(hXfer);
    corBufferFree(hBuffer);
    CorVicFree(hVic);
    CorCamFree(hCam);
    CorAcqrelease(hAcq);
    return 0;
}
```

5.4 三通道望远镜远程观测终端系统对相机类的封装

由于现在大部分相机都使用了 Cameralink 的工业标准的控制接口，不同厂家不同型号的相机的不同之处就在于相机的命令不同，利用 C++语言继承派生，以及多态性的特点，将控制相机的共同部分抽象出来，作为相机的一个基类，通过继承基类，再添加相机相对应的命令，就能够很快速的方便的生成对应相机的类来。这对于以后望远镜因需要增加或者更换新的相机时，就可以快速的升级终端控制系统，对于开发新的望远镜的终端系统，也同样适用。

由以上章节可知，对于望远镜相机的基类的封装，大体可以包含以下几个部分，首先是相机的初始配置文件的读取，然后是相机运行时资源的一些配置；包括动态资源和静态资源，在三通道中主要封装的有缓存资源，图像采集模块；还有就是串口初始化，用来建立图像采集卡和计算机之间的通信；再将远程传输功能作为内嵌对象添加到相机的基类中去，使得相机具有接受远程控制的能力，其框架如下图 5.4 所示：

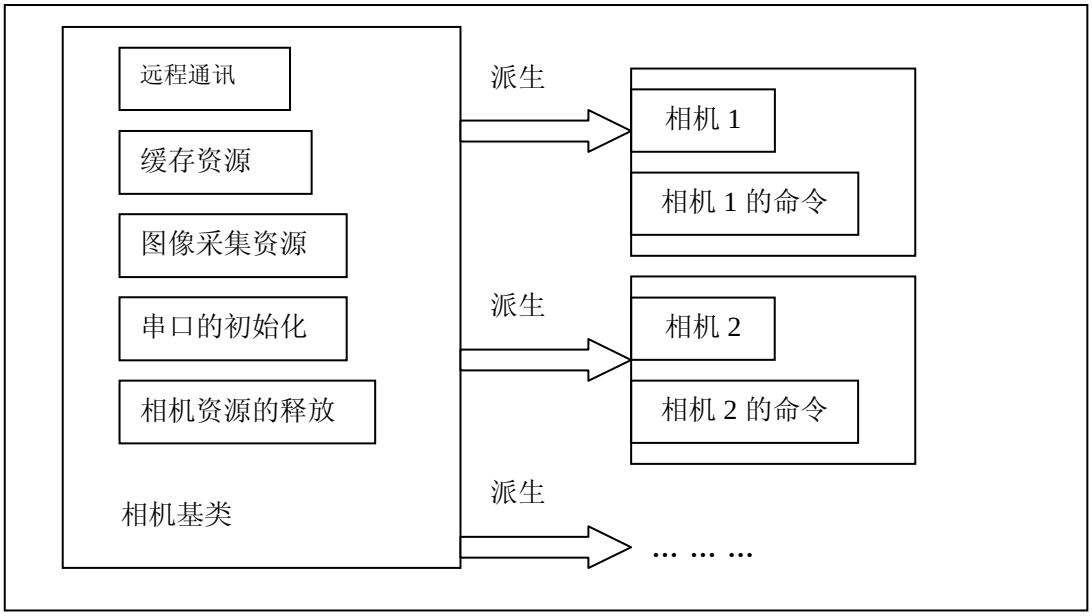


图 5.4 相机基类封装架构图

5.5 小结

本章主要从软件方面介绍了控制相机的方法。由于三通道望远镜终端系统设计的服务器端主要利用基于 C 编程，因此介绍了软件 Sapera LT 基于 C 的 API 的库的控制相机的基础函数，来对相机的参数和模式进行设置，最后获取图像。以及在此基础上讨论了利用 C++语言的继承派生和多态的特性将相机的控制模块化，把相机控制的共性封装到一个相机的基类中，大大提高了开发望远镜终端系统的效率和灵活性。

第六章 结论和展望

6.1 结论

目前系统已经实现了怀柔太阳观测基地两个观测室之间局域网内对三通道望远镜的图像数据和相机控制命令的传输等远程观测功能，并取得了初步的观测结果。

图 6.1 为利用该终端系统获取的太阳活动区的实时图像：

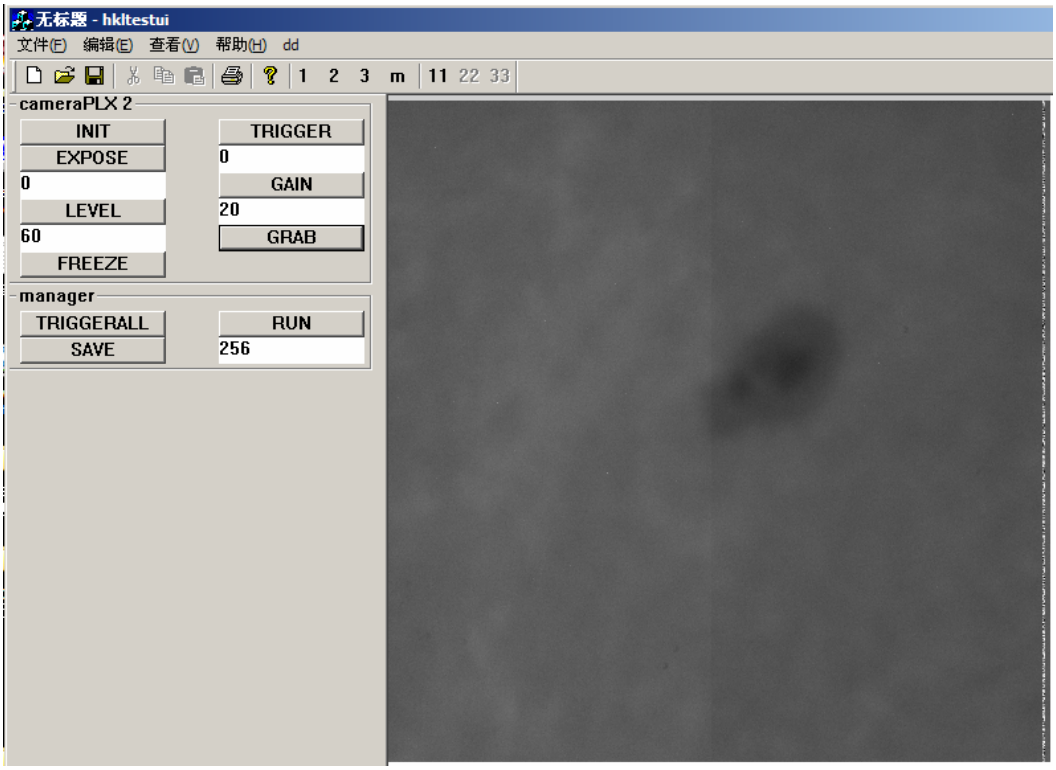


图 6.1 太阳活动区实时图像

6.2 后续工作的展望

后续的工作主要有三大部分，（1）把远程观测拓展到广域网中，要解决的问题有：数据的安全传输，即对传输数据的加密,保证数据的安全传输；用户权限认证，即对于不同的用户，赋予其不同的操作权限，保证望远镜的安全操作；（2）将保存的数据同数据库接轨，将采集的数据直接保存到数据库中，方便查询；（3）

将该终端系统升级，提升其跨平台的兼容性，使得其在 Linux，以及在嵌入式系统中也能够运行，使其应用更灵活，应用范围更广泛。

参考文献

[1]. George Kosugi, Toshiyuki Sasaki, Masafumi Yagi et al. Remote observing capability with Subaru Telescope . Proceedings of SPIE - The International Society for Optical Engineering, v 5496, Advanced Software, Control, and Communication Systems for Astronomy, 2004, p 695-700

[2]. 侯俊杰 《深入浅出 MFC》华中理工大学出版社 2001

[3]. Jeffrey Richter 《Windows 核心编程》机械工业出版社 2006

[4]. R.J.Hanisch,A.farris,E.WGreisen et al. Definition of the Flexible Image Transport System(FITS).Astronomy & Astrophysics A&A 376,359-380(2001)

[5]. AnthonyJones，Jim Ohlund.《Windows 网络编程》清华大学出版社 2002

[6]. http://www.imperx.com
LYNX_Users_Manual.pdf

[7]. 王庆有 孙学珠 CCD 应用技术 天津大学出版社 1993

[8]. 王以铭 电荷耦合器件原理与应用 科学出版社 1987

[9]. http://mv.dalsa.com
Datasheet_TM-TMC-1400.pdf

[10]. http://www.imaging.com
SaperaBasic.pdf

[11]. http://www.china-vision.net

发表文章目录：

申基，胡柯良，林佳本，邓元勇. 怀柔太阳观测基地三通道太阳望远镜局域网内远程观测终端系统设计，天文研究与技术（已录用）

致谢

三年的硕士生涯即将结束，在此期间，得到了许多老师、同学和朋友的关心、支持和帮助，才使我得以能够顺利地完成学业，在这里衷心的感谢他们。

首先，我要由衷地感谢邓元勇老师和胡柯良老师三年来对我的精心指导，教育了我如何从事科研工作。邓老师严谨的治学态度，精益求精的工作作风，诲人不倦的高尚师德，严以律己、宽以待人的崇高风范，胡老师朴实无华、平易近人的人格魅力对我影响深远，一生受用。

还要感谢怀柔组的王东光老师，林钢华老师，李威老师课题上和论文写作上的帮助和生活上的关怀，他们治学、工作和为人处世的风格也使我受益匪浅。

感谢怀柔的林佳本，杨尚斌，肖江，刘健，郝娟，徐海青，孙瑛姿，陈洁，高欲，王栋，玄维佳，王传宇，谢文斌等同学和朋友在生活与学业上的关心和帮助。

感谢同班的杨志光，孙京海等同学在做课题的过程中对我的帮助。

感谢怀柔基地的汪国萍，高其峰两位观测员在工作中的协助和配合。

特别感谢国家天文台杜红荣老师和艾华老师在三年的学习生活中给我的关心。

最后，感谢我的父亲、母亲以及家人的关心，你们的支持与鼓励是我前进的巨大动力。