基于改进的 H.264 的视频监控系统①

毛剑飞[1], 张 杰[1], 蒋 莉[1], 王子仁[1], 马祥音[1], 廖智蓉[2]

[1](浙江工业大学 计算机科学与技术学院, 杭州 310023)
[2](浙江长征职业技术学院 计算机系, 杭州 310023)

摘 要: 嵌入式技术的飞速发展推动了视频采集领域的技术革新, 使得视频监控系统越来越朝着小型化、智能化、嵌入式化、远程化的方向发展. 文中以基于 S3C6410 处理器的 ARM 板为硬件平台, 采用 Linux 为嵌入式操作系统, 建立了一个视频采集服务器; 以 PC 机为客户端, 实时地远程显示视频---建立了一个远程视频监控系统. 系统利用 Linux 内核提供的视频数据采集接口 V4L2 控制 USB 摄像头来采集图像, 利用网络套接字将编码后的图像数据传输至客户端; 在客户端将图像解码并转化为 RGB 格式后, 利用 GTK 把转化后的图像显示出来. 针对 H.264 运动估计算法的两点不足, 提出了动态搜索范围策略和搜索点分组策略. 实验结果证明: 在不影响重建图像质量和编码码率的前提下, 改进的算法有效地降低了算法的编码时间和运动估计时间, 提高了编码过程的实时性. 通过本文的设计方案能够获得稳定清晰的图像, 实现了远程视频监控.

关键词: 视频压缩; USB 摄像头; 视频监控; 远程显示; H.264

Video Monitoring System Based on Improved H.264

MAO Jian-Fei[1], ZHANG Jie[1], JIANG Li[1], WANG Zi-Ren[1], MA Xiang-Yin[1], LIAO Zhi-Rong[2]

[1](Department of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)
[2](Department of Computer Science, Zhejiang Changzheng Vocational and Technical College, Hangzhou 310023, China)

Abstract: The rapid development of embedded technology promotes innovation in the field of video capturing, making video surveillance system towards smaller, smarter, more embedded and longer-distance fashion. A video acquisition server is established with ARM board based on S3C6410 processor and Linux served as hardware and software system respectively. Together with a PC client, remotely and real-time displaying video, a long-distance video surveillance system is set up. By control of USB camera through video data acquisition interface, V4L2, the system can acquire images. The encoded image data is sent to clients through Internet sockets. The image is displayed on the client after the accepted image has been already decoded and converted to RGB format. Two methods, dynamic search window and search points-grouping, have been proposed to solve two short points of motion estimation algorithm in H.264. The experimental result shows compared to the original algorithm, the improved algorithm can greatly decrease encoding time and motion estimation time, while maintain the image quality and encoding bitrate. The system designed here can get continued images, implementing video monitoring remotely.

Key words: video compression; USB camera; video surveillance; long-distance display; H.264

视频监控系统在工业、军事、民用等领域有着广泛的应用, 为这些行业的安全防范和环境监控发挥着重要的作用[1]. 传统的视频监控系统一般基于 PC 平台和数据采集卡, 虽然具有优良的性能, 但是此类系统造价高、体积庞大, 越来越难以满足当今时代的需求. 随着嵌入式技术和视频编解码技术的不断发展, 高度

① 基金项目:浙江省自然科学基金(LY12F02037);浙江省科技厅公益项目(2012C23122, 2013C33055); 浙江省可视媒体智能处理技术研究重点实验室开放基金(2013050).

收稿时间:2013-09-05;收到修改稿时间:2013-10-21

集成化、小型化、高性能、价格低廉、实时性高的嵌入式监控系统得到广泛的应用.

本文系统采用华天正科技公司生产的基于三星公司 S3C6410 芯片的 ARM 板为硬件平台, 以开源免费的Linux为操作系统, 这样做既简化了硬件设计, 也保证了系统的稳定性和实时性.

视频监控系统采集到的视频序列数据量非常大, 直接传输势必会浪费大量的网络带宽, 且不方便存储. 本文在深入分析视频编解码标准 H.264 的运动估计算法 UMHexagonS 的基础上, 提出了两点改进策略并将改进的算法应用到视频压缩过程中, 有效地节省了运动估计时间, 提高了系统的实时性.

1 构建嵌入式开发环境

1.1 ARM 板上的硬件系统

实验用的 ARM 板主要的系统硬件结构图如图 1 所示.

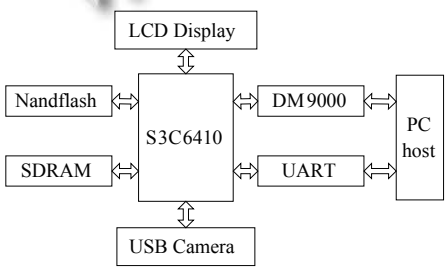

图 1 系统硬件结构图

硬件系统采用基于 ARM11 架构的 S3C6410 处理器, 其采用 ARM1176JZF-S 的核, 包括 16KB 指令和 16KB 数据 Cache, 16KB 指令和 16KB 数据 TCM, 具有强大的计算处理能力[2].

1.2 嵌入式软件系统的构建

从软件的角度可将嵌入式 Linux 系统划分为 4 个层次, 由低到高分别为系统引导加载程序、操作系统内核、文件系统和用户应用程序[3]. 嵌入式软件系统的构建主要分为以下三个部分: Bootloader 的移植、内核的移植、根文件系统的移植.

Bootloader 是在嵌入式系统运行之前运行的一段程序. 通过这段 bootloader 小程序可以初始化硬件设备、建立内存空间的映射图, 从而将系统的软硬件环境带到一个合适的状态, 以便为最终调用操作系统内核准备好正确的环境[4].

在烧写内核之前, 需要编译出正确的内核. 本系统使用的是 linux2.6.28.6, 首先修改 Makefile 文件, 然后执行“make menuconfig”命令进入内核配置界面. 在配置界面上需要根据华天正科技公司的 Real6410 开发板选择相应的配置选项, 在配置驱动选项的时候尤其注意选择 V4L USB devices 选项下的 USB Video Class(UVC)、Video For Linux 选项下的 Enable Video For Linux API compatible Layer 和 FrameBuffer 等选项, 使得编译出的内核能够识别 UVC 驱动的摄像头、打开视频设备的采集接口 V4L2、支持帧缓冲机制等. 此外, 要取消选择系统并不需要的配置选项以减小最终生成的内核大小. 最后执行“make clean; make dep; make zImage”命令, 编译成功后会在/arch/arm/boot/目录下生成 zImage 内核镜像文件, 将其烧写进 nandflash 即可[5].

仅有 U-boot 和 Linux 内核是不能和目标板进行交互的, 还需要构建文件系统. 文件系统包含了 Linux 系统能够运行所需的应用程序和库[6]. 本系统使用的是 ubifs 格式的根文件系统.

以上就是嵌入式软件系统构建的过程. 构建成功后就可以正确启动 ARM 上的 Linux 了, 通过 DNW 调试工具可以查看系统正确启动过程的相关信息.

2 软件系统的设计和实现

2.1 软件系统的总体设计

软件系统体系结构基于 C/S 架构, 即客户机和服务器(Client/Server)架构. 通过它可以充分利用两端软硬件环境的优势, 将任务合理分配到 Client 端和 Server 端来实现, 降低了系统的通讯开销.

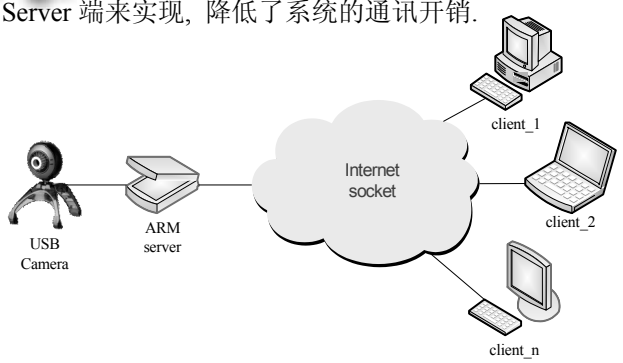

图 2 系统整体结构示意图

如图 2 所示, 在本系统中 ARM 及外围设备充当服务器. 服务器端程序利用内核提供的视频数据采集接口 V4L2 操纵 USB 摄像头采集图像; 服务器接入网络,

并建立套接字(socket)用以监听客户端发来的连接请求, 成功建立 socket 连接之后, 服务器就可以通过 socket 将图像数据发送至客户端.

PC 机是客户端, 可以在不同的 PC 上分别运行客户端程序, 也可以在同一个 PC 上运行多个客户端程序, 实现多终端显示视频. 客户端接入网络后可以向服务器发送 socket 连接请求, 建立连接后就可以接收服务器采集的图像, 连续的图像刷新显示便构成了视频, 这样便实现了视频的远程监控.

2.2 图像的采集

在服务器端采集数据是通过内接提供的 V4L2 接口实现的. V4L(Video For Linux)是 Linux 下用于获取视频和音频数据的 API 接口, 涉及打开和关闭视频设备、设置采集图像的格式、采集并处理图像信息等[7]. V4L2(Video Linux For Two)是 V4L 的改进版本, 修复了 V4L 的部分设计 bug.

视频设备是字符设备, 在 Linux 中对应于/dev/videox(x=0,1…)设备文件.

利用 V4L2 操作视频设备的一般过程如图 3 所示:

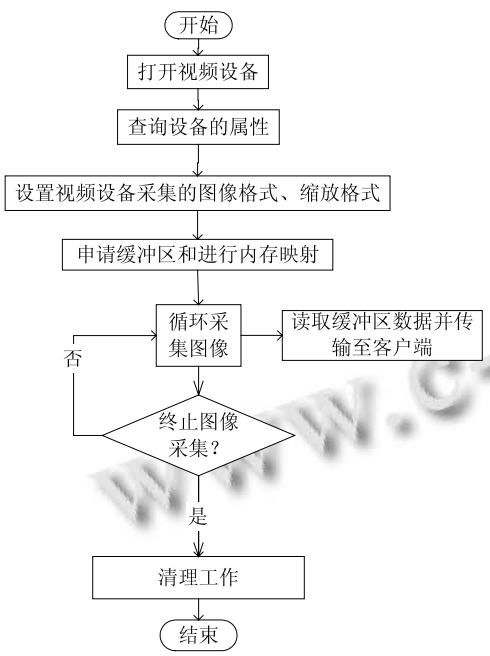

图 3　基于 V4L2 的图像采集流程

视频采集的具体过程如下:

1) 打开视频设备

通过 int fd=open("/dev/video0",O_RDWR);来打开视频设备, 其中 fd 为视频设备成功打开后返回的文件描述符. 若打开失败返回-1.

2) 查询设备的属性

这一步主要是为了获得 v4l2_capability 结构体中的 capabilities 字段的值, 以此来判断视频设备是否支持视频捕获操作和流输入输出方法. 可通过 ioctl(fd,VIDIOC_QUERYCA-P,&cap)系统调用来实现.

3) 设置视频设备采集的图像格式和缩放格式

通过 ioctl(fd,VIDIOC_ENUM_FMT,&fmtdesc)操作可以枚举出设备支持的所有图像格式, 本系统采用的摄像头只支持 YUV4:2:2 格式的图像采集. 因此要设置 fmt.fmt.pix.pixelformat=V4L2_PIX_FMT_YUYV, 此外还要设置数据流的类型和数据帧的宽和高即 fmt.type=V4L2_BUF_TYPE_VIDEO_CAPTURE,fmt.fmt.pix.width=320,fmt.fmt.pix.height=240, 最后通过 ioctl(fd,VIDIOC_S_FMT,&fmt)使得设置生效.

对于缩放特性, 只需设置采集图像的类型字段即可, 其它选项选用默认的. cropcap->type=V4L2_BUF_TYPE_VIDEO_CAPTURE, 然后调用 ioctl(fd,VIDIOC_S_CAP,&crop); 使能设置.

4) 申请图像缓冲区和进行内存映射

设置 reqbuffers.count 为要申请的缓冲区数目, reqbuffers.type 要与设置图像格式时设置的流类型相同, 即 V4L2_BUF_TYPE_VIDEO_CAPTURE, reqbuffers.memory 设置成内存映射方式. 通过 ioctl(fd,VIDIOC_REQBUFS,&reqbuffers); 申请缓冲区. 内存映射, 简而言之就是将用户空间的一段内存区域映射到内核空间, 映射成功后, 用户对这段内存区域的修改可以直接反映到内核空间; 相反, 内核空间对这段区域的修改也直接反映用户空间. 那么对于内核空间和用户空间两者之间需要大量数据传输等操作的话效率是非常高的. 可以通过 mmap()系统调用实现内存映射操作, 并且获得缓冲区的首地址, 以用于将缓冲区入列操作.

5) 循环获取图像

通过 VIDIOC_STREAMON 和 VIDIOC_STREAMOFF 可以启动和停止图像采集, 在图像采集之前要先通过 ioctl(fd,VIDIOC_QBUF,&type)系统调用将申请的缓冲区放入队列当中.

图像采集完成后, 假若此时服务器已与客户端建立 socket 连接, 则可将图像数据传输到客户端.

6) 清理工作

停止采集图像后要进行必要的清理工作, 如释放申请的缓冲区、反内存映射、释放动态申请的内存单元等等, 最后要通过 close(fd);来关闭视频设备.

2.3 图像数据的编码

在视频编码中, 运动估计时间大约占到总编码时间的 80%左右. 在不影响图像质量和编码码率的情况下, 设法减少编码算法的运动估计时间就能有效地提高视频编码的效率. 全搜索算法的做法是在搜索窗口中穷举匹配当前编码图像块, 这样做能够获得最高的编码效率, 但是其时间复杂度非常高. 因此提出了一些快速运动估计算法来降低视频编码的时间复杂度, 比如三步搜索法、新三步搜索法[8]、四步搜索法[9]、菱形搜索法[10]、六边形搜索法[11]等, 这些算法在不同程度上会容易陷入局部最优. 清华大学提出的非均匀十字型多层次六边形格点搜索算法 (Unsymmetrical-cross Multi-Hexagon-grid Search Algorithm, UMHexagonS)[12]是视频编解码标准 H.264/AVC 的标准运动估计算法, 能有效解决局部最优问题. UMHexagonS 算法是一种层次型的运动搜索策略[13], 之所以称之为混合型搜索算法是因为 UMHexagonS 算法包含四个步骤, 并且每个步骤使用不同类型的搜索模板: 1) 起始点预测; 2) 非对称十字型搜索; 3) 非均匀多层次六边形搜索; 4) 扩展的六边形搜索. UMHexagonS 算法搜索过程如图 4 所示.

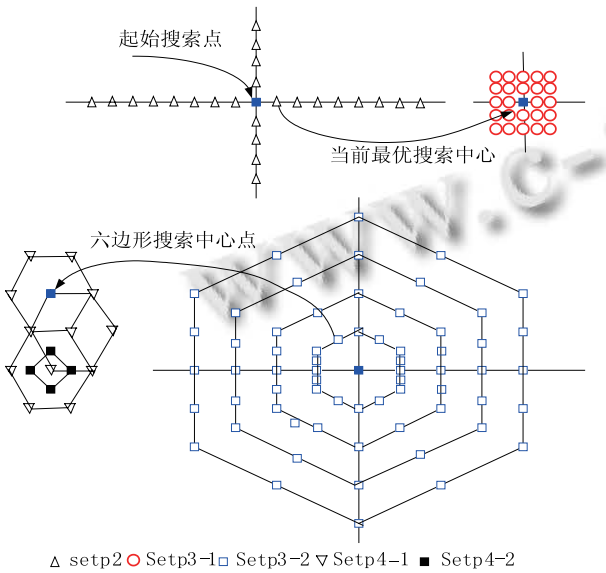

图 4 UMHexagonS 算法搜索过程

仔细研究 UMHexagonS 算法的搜索步骤, 可以发现此算法存在许多不足之处. 下面从两个方面进行讨论, 并给出解决方法.

(1) 动态搜索窗口策略

UMHexagonS 算法使用固定大小的搜索窗口, 其值由配置文件 encoder_main.cfg 中的参数 SearchRange 决定, 而在 H.264 标准中, 运动估计中分块有 4*4 到 16*16 等七种不同的大小. 对于不同大小的分块使用同样大小的搜索窗口是不合理, 因为对于小分块来讲这样必然会增大运动估计的时间复杂度, 而对于大分块来说会降低运动估计的匹配精度. 可以采用动态改变搜索窗口的方法来解决这个问题. 根据文献[14], 可以采用中值预测模式和上层预测模式来求动态搜索范围的大小.

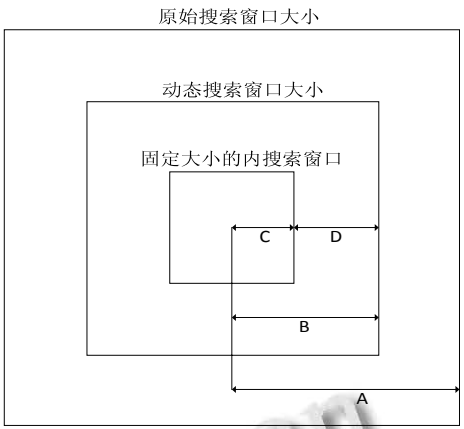

图 5 动态搜索窗口

动态搜索窗口策略如图 5 所示, 其中:

A: 配置文件中参数 SearchRange 的大小, 即原始搜索窗口大小(input_search_range);

B: 动态搜索窗口的大小(prpsd_DSR);

C: 动态搜索窗口中固定部分窗口大小(fixed_part);

D: 动态搜索窗口中可变部分窗口大小(dynamic_part);

$$prpsd\_DSR=fixed\_parl+dynamic\_part \quad (1)$$

$$fixed\_part=input\_search\_range/8 \quad (2)$$

$$dynamic\_part=\max(abs(MVP_{uplayerX}-MVP_{medianX}), abs(MVP_{uplayerY}-MVP_{medianY})) \quad (3)$$

(2) 分组策略

UMHexagonS 算法的第四个步骤是执行扩展的六边形搜索, 当将搜索模板由大六边形搜索模板转换成小六边形搜索模板后, 每次搜索过程需要匹配如图 6

所示的 2、4、6、8 四个匹配点, 而忽略了 1、3、7、9 四个点. 经过 UMHexagonS 算法前几个步骤, 全局最优点很有可能就位于当前搜索模板内的某个搜索点上. 这样忽略掉的 1、3、7、9 四个点很有可能就是全局最优点.

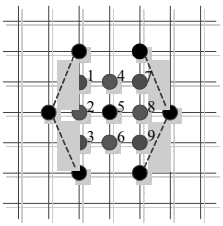

图 6　六边形内搜索点分布

如果采用全搜索策略必然会增加匹配点数, 进而增加算法的时间复杂度. 在这里, 采用分组策略来解决这一问题.

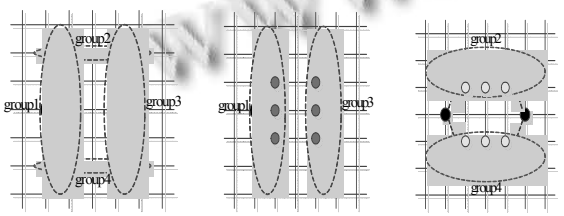

图 7　大六边形搜索模板的分组策略

分组策略基于这样的理论: 在围绕全局最优点的很小范围内, 误差曲面是单调递减的. 将大六边形搜索模板内的搜索点按如图 7 所示分成四个分组: group1, group2, group3, group4, 其中 group1 和 group3 包含三个匹配点, group2 和 group2 包含两个匹配点, 并将每个分组的各个匹配点的绝对误差(Sum of Absolute Difference, SAD)和分别记为 SADgroup1, SADgroup2, SADgroup3, SADgroup4. 比较四个分组的 SAD, 找出其中最小值, 记为 SADgroup_min. 如果 SADgroup_min= SADgroup1, 则只需匹配图 6 中的 1, 2, 3 三个点; 如果 SADgroup_min= SADgroup2, 则只需匹配图 6 中的 1, 4, 7 三个点; 如果 SADgroup_min= SADgroup3, 则只需匹配图 6 中的 7, 8, 9 三个点; 如果 SADgroup_min= SADgroup4, 则只需匹配图 6 中的 3, 6, 9 三个点.

在编码库 X264 中实现上述改进的算法, 即可用于编码采集的图像.

2.4 图像数据的传输

Unix 不同主机的进程之间可通过套接字进行通信. 套接字(socket)有很多种类型, 如基于 TCP 协议的 SOCK_STREAM 套接字和基于 UDP 协议的 SOCK_DGRAM 套接字等.

SOCK_STRAM 套接字是有序、可靠、双向的面向连接的字节流, 能够保证数据在网络间可靠、有序、无差错的传输, 故本文采用基于 TCP 协议的 SOCK_STREAM 套接字进行视频数据传输. 利用套接字进行数据通信的工作过程流程图如图 8 所示.

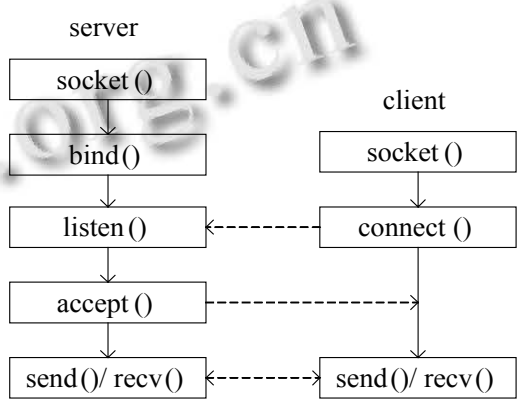

图 8　socket 工作流程图

Socket 的工作过程简述如下:

在 server 端, 首先通过 socket(AF_UNSPEC,SOCK_STREAM,0)函数创建一个 SOCK_STREAM 类型的套接字, 成功后返回套接字描述符; 接着通过 bind(sock_fd,&sockaddr_server,sizeof(sockaddr_server)) 函数将套接字与 server 端的某一个端口进行绑定, 如端口 3499; 接下来 server 就要进入监听状态, 通过 listen(sock_fd,10)函数侦听成功建立的套接字, 等待客户端的连接请求; 当 server 接收到来自客户端的连接请求后, 便可以通过 accept 函数接受连接请求, 函数调用成功后返回新的套接字描述符以用于和客户端进行数据通信; 此时便可以通过 send/recv 函数与建立连接的客户端进行数据通信了.

与服务器相比, 在 client 端建立 socket 通信过程要简单很多. 只需要通过 socket 函数建立套接字, 然后通过 connect 函数向 server 发出连接请求, 被接受后就可以通过 send/recv 函数与 server 通信了.

2.5 图像格式的转化

客户端接收到的图像数据是 YUV(422)格式的, 若要进行显示先要转为 RGB 格式.

RGB888 图像数据在内存中的存储方式是这样的:

每个像素点的 R、G、B 颜色数据各占 8 个比特, 即一个字节, 每个像素点数据占三个字节.

YUV422 图像中每两个相邻像素点共享 U、V 颜色数据, 且每个颜色数据占 8 比特, 即 1 个字节. 其在内存中是这样存储的: Y0、U0、Y1、V0、Y2、U1、Y3、V2……这样 YUV 图像的每个像素点平均占 2 个字节, 相比于 RGB888 图像来讲节省了内存存储空间.

图像 YUV 到 RGB 格式的转化公式如(4)所示.

$$\begin{bmatrix} B \\ G \\ R \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 2.018 \\ 1.164 & -0.813 & -0.391 \\ 1.164 & 1.596 & 0 \end{bmatrix} \begin{bmatrix} Y-16 \\ V-128 \\ U-128 \end{bmatrix} \tag{4}$$

2.6 图像的显示

本系统是利用 GTK 显示图像的.

GTK+(GIMP Toolkit)是一套跨多种平台的图形工具包, 按 LGPL 许可协议发布的, 目前已发展为一个功能强大、设计灵活的一个通用图形库. 特别是被 GNOME 选中使得 GTK+广为流传, 成为 Linux 下开发图形界面的应用程序的主流开发工具之一.

GTK+是一种图形用户界面(GUI)工具包. 也就是说, 它是一个库(或者, 实际上是若干个密切相关的库的集合), 它支持创建基于 GUI 的应用程序. 可以把 GTK+想像成一个工具包, 从这个工具包中可以找到用来创建 GUI 的许多已经准备好的构造块.

GTK+是事件驱动(event-driven)的, 主要工作原理是信号回调机制. 通过 g_signal_connect()函数, 它将信号与函数联系起来, 当某个事件发生时便产生某种特定的信号. GTK+可以捕获这一信号并执行与信号联系的函数. 在客户端程序中, 利用 GTK+显示从服务器接收的视频数据时关键用到 drawingarea 控件和 expose_event 信号. 当每次成功接收到一帧数据时, 客户端程序调用 yuv_rgb()函数将图像由 YUV 转化为 RGB 格式, 然后调用 gtk_widget_queue_draw()函数使 drawingarea 控件产生 expose_event 信号. 主函数可以捕获这一信号并执行与其关联的 gdk_draw_rgb_image()函数显示图像. 通过连续不断地接收、解码、转化、显示帧数据, 客户端就能够实现实时显示 USB 摄像头采集的视频.

3 实验结果及分析

UMHexagonS 改进算法的实验选用六种不同运动程度的 qcif 测试序列进行实验: container、mobile、mother-daughter、coastguard、foreman、news.

在视频编解码标准测试软件 JM86 中用 C 语言实现上文中提到的两个创新点-动态搜索策略和分组策略, 选择软件包 bin 目录下的 encoder_main.cfg 配置文件, 其主要的配置参数设置如下: FramesToBeEncoded=100(编码帧数), FrameRate=30(编码码率), SourceWidth=176(帧宽度), SourceHeight=144(帧高度), SearchRange=16(搜索窗口大小), UseHadamard=1(使用 Hadamard 变换), NumberBFrames=0(不采用 B 帧, 即编码帧序列采用 IPPP).

表 1 实验结果

测试序列	算 法	PSNR(db)	编码码率(kbit/s)	编码时间(s)	编码时间提高率(%)	运动估计时间(s)	运动估计时间提高率(%)
container	原始的 UMHex	37.005	132.587	84.052	7.11	18.312	27.05
	改进的 UMHex	37.005	132.580	78.078		13.359	
mother-daughter	原始的 UMHex	38.019	99.486	82.611	6.73	18.336	23.61
	改进的 UMHex	38.013	99.355	77.055		14.007	
mobile	原始的 UMHex	34.027	472.860	111.354	6.38	22.718	26.38
	改进的 UMHex	34.023	472.428	104.246		16.725	
coastguard	原始的 UMHex	34.760	240.493	93.049	6.09	25.424	22.39
	改进的 UMHex	34.764	240.698	87.721		19.732	
foreman	原始的 UMHex	37.002	168.203	84.052	5.73	22.737	20.05
	改进的 UMHex	36.998	168.568	78.078		18.179	
news	原始的 UMHex	37.541	163.798	83.064	3.78	16.903	23.11
	改进的 UMHex	37.531	164.012	79.921		12.997	

分析表 1 的实验结果可以看出: 改进的 UMHexagonS 算法与原 UMHexagonS 算法相比, PSNR(峰值信噪比)和 bitrate(编码码率)没有很大的改变, 而编码时间和运动估计时间有着明显的减小. 编码时

间最高提高率达到 7.11%, 最低提高率达到 3.78%, 平均提高率达到 5.97%; 运动估计时间最高提高率达到 27.05%, 最低提高率达到 20.05%, 平均提高率达到 23.765%.

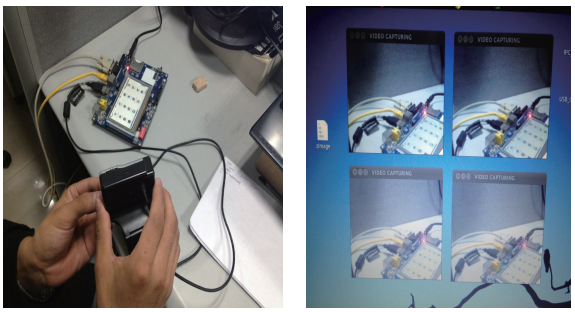

图 9 实验效果图

视频监控系统最终实验效果图如图 9 所示. 本系统能够获得连续流畅的图像, 并能够多客户端显示视频, 能够应用于需要进行监控的场合.

4 结语

本文设计并实现了一种基于 S3C6410 和 H.264 的嵌入式视频监控系统方案, 对嵌入式硬件平台、开发环境的建立等做了简单的介绍, 重点阐述了软件系统, 包括与图像的采集、编解码、传输、格式转化、显示相关的知识. 在分析 UMHexagonS 算法的基础上, 提出了动态搜索范围策略和搜索点分组策略改进了原算法的不足. 改进的 UMHexagonS 算法与原算法相比, 在没有增加编码码率和保证图像质量的情况下有效地节省了运动估计时间, 降低了算法的时间复杂度, 提高了编码器的实时性.

和市场上其它视频监控系统相比, 本系统具有开发周期短、价格低廉、功耗小、体积小等特点, 适用于对视频图像要求不高的场合. 下一步准备将采集的图像经过压缩后通过 Wi-Fi 或者 3G 等无线方式传输数据, 以提高系统的灵活性、可用性.

参考文献

1 李侃,廖启征.基于 S3C2410 平台与嵌入式 Linux 的图像采集应用.微计算机信息,2006,22(3-2):125–127.

2 王刚,毛剑飞,田青,毛飞飞.基于 ARM11 的无线视频监控系统.计算机系统应用,2011,20(8):18–22.

3 刘刚,赵剑川.Linux 系统移植.北京:清华大学出版社,2011:64–65.

4 田磊.基于 ARM 的嵌入式 Linux 操作系统的移植[硕士学位].西安:西安电子科技大学,2009.

5 淦克亮.基于 ARM 嵌入式的图像采集与显示系统设计.工业控制计算机,2011,24(8):10–12.

6 刘升,马进.S3C2440 平台上的视频监控系统的研究与实现.计算机技术与发展,2010,20(7):240–243,249.

7 戴雯惠.基于嵌入式 Web 技术的远程视频监控系统的研究.现代计算机:上半月版,2012,(13):78–80.

8 Li RX, Zeng B, Liou ML. A new three-step search algorithm for block motion estimation. IEEE Trans. on Circuits and Systems for Video Technology, 1994, 4(4):438–442.

9 Po LM, Ma WC. A novel four-step search algorithm for fast block motion estimation. IEEE Trans. on Circuits and Systems for Video Technology, 1996, 6(3):313–317.

10 Zhu S, Ma KK. A new diamond search algorithm for fast block matching motion estimation. International Conference on Information, Communications and Signal Processing Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications. 1997. 292–296.

11 Zhu C, Lin X, Chau LP. Hexagon-based search patten for fast block motion estimation. IEEE Trans. on Circuits and Systems for Video Technology, 2002, 12(5):349–355.

12 Chen ZB, He Y, Xu JF. Hybrid unsymmetrical cross multi-hexagon-grid search strategy for integer pel motion estimation in H.264. Proc. Picture Coding Symposium, Saint-Malo. 2003. 17–22.

13 Chen ZB, Xu JF, He Y, Zheng JL. Fast integer-pel and fractional-pel motion estimation for H.264/AVC. Journal of Visual Communication and Image Representation, Special issue on emerging H.264/AVC video coding standard. 2006, 17(2): 264–290.

14 Chen ZX, Song Y, Ikenaga T, Goto S. A dynamic search range algorithm for variable block size motion estimation in H 264/AVC. 2007 6th International Conference on Information, Communication & Siqnal Processing. 2007. 1–4.