A noise detection, noise-motion separation and a cancer recognition theory and algorithm

Alexander Popovich

University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation


https://digitalscholarship.unlv.edu/rtds/1386

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
A NOISE DETECTION. NOISE-MOTION SEPARATION AND A CANCER RECOGNITION THEORY AND ALGORITHM

by

Alexander Popovich

Bachelor of Science
University of Nevada, Las Vegas
1998

A thesis submitted in partial fulfillment of the requirements for the

Master of Science Degree
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
August 2002
The Thesis prepared by

ALEXANDER POPOVICH

Entitled

A NOISE DETECTION, NOISE-MOTION SEPARATION AND A CANCER RECOGNITION THEOREY AND ALGORITHM.

is approved in partial fulfillment of the requirements for the degree of

MASTERS DEGREE IN COMPUTER SCIENCE

Examination Committee Chair

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

A Noise Detection, Noise-Motion Separation and a Cancer Recognition
Theory and Algorithm

by

Alexander Popovich

Dr. E. A. Yfantis, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

In this thesis we describe a noise detection and a motion-noise separation algorithm, as well as the stochastic properties of the noise. The difference between corresponding pixels subject to one type of noise, of two frames, has mean vector equal to (0, 0, 0), and variance covariance matrix with relatively small variances, for the (R, G, B) difference values. The other type of noise is a result of disturbance of the light equilibrium due to motion in neighboring or nearby pixels. In this type of noise the mean of the difference is non-zero, so when the bias vector is subtracted from the (R, G, B) differences the new mean is zero. Every pixel not included in the one type of noise or the other is part of the motion set between the two frames. The pixels are organized in macroblocks, so macroblocks containing pixels with motion are applied motion estimation and motion compensation methods first and subsequently the difference between the corresponding macroblocks of the two frames is obtained.

This thesis furthermore describes an algorithm of cancer recognition of ultrasound images. As we pointed out in our previous work, in order for cancer to survive it develops its own blood supply system, which is different than the supply system of normal tissue. The velocity of the blood flowing through the cancerous blood vessels
is different than the velocity of the blood flowing through blood vessels of normal tissue. Due to this fact the ultrasound signal is absorbed differently in the cancerous areas than in the normal tissue areas. The energy of the signal, the continuity of the signal, the autocorrelation function and frequency domain properties are different in the normal tissue than in the cancerous tissue. All of these indicators are weighted here for the purpose of classifying the image of the tissue as being cancerous or non-cancerous. Preliminary results based on limited number of ultrasound images show that our method has the ability to recognize cancer in ultrasound images.
# TABLE OF CONTENTS

ABSTRACT ................................................................. iii
LIST OF FIGURES ......................................................... vii
ACKNOWLEDGMENTS .................................................... viii

CHAPTER 1 INTRODUCTION .............................................. 1
  A Noise Detection and Noise-Motion Separation ...................... 1
  A Cancer Recognition Theory and Algorithm ......................... 2

CHAPTER 2 A NOISE DETECTION AND NOISE-MOTION SEPARATION 5
  Stochastic Attributes of Noise ........................................... 5
  Summary ........................................................................ 9
  Introduction ................................................................... 10
  Motion Detection Motion Estimation Motion Compensation .... 11
  Summary and Conclusions .............................................. 17

CHAPTER 3 A CANCER RECOGNITION THEORY AND ALGORITHM 18
  Introduction ................................................................... 18
  The Recognition Algorithm ........................................... 19
  Summary and Conclusions .............................................. 26

CHAPTER 4 AN APPLICATION OF NOISE DETECTION AND SEPARATION ALGORITHM ......................................................... 27
  A Video Surveillance and Security System ............................ 27
  An Introduction ................................................................ 27
  Design of a video surveillance and security system ............... 27
    The Server ................................................................... 29
    The Client ................................................................... 29
  Implementation .................................................................. 29
  Conclusions and Future Work ......................................... 30
    Conclusions .................................................................. 30
    Future Work .................................................................. 30

CHAPTER 5 AN APPLICATION OF A CANCER RECOGNITION THEORY AND ALGORITHM ......................................................... 32
  The Recognition of Cancerous Tissue ................................. 32
  Conclusions and Future Work ......................................... 32
    Conclusions .................................................................. 32
    Future Work .................................................................. 32

BIBLIOGRAPHY ............................................................. 34

APPENDIX .......................................................................... 36

VITA .................................................................................. 118
LIST OF FIGURES

3.1 Cancerous image ......................................................... 18
3.2 Histogram of 21x21 block with no cancer ......................... 24
3.3 Histogram of 21x21 block with cancer ............................ 25
3.4 The 21x21 moving pixel spyglass has recognized cancer .... 26
4.1 The Security System Architecture .................................... 28
ACKNOWLEDGMENTS

I would like to thank Dr. E. A. Yfantis for giving me the opportunity to work on such interesting areas, projects and finally, providing me with the insight into new theories and technologies.

Furthermore, I would like to thank the committee reviewing my thesis. Dr. L. Larmore, Dr. W. W. Bein, and Dr. P. Stubberud.

Special thanks to Angelos Angelopoulos and Theodores Lazarakis for all their help and support. I would also like to thank my family and friends in Yugoslavia for their support and love.

Many thanks to my friends Vladan Jankovic, Michael Kekerovic and Kiyomi Kogane.
CHAPTER 1

INTRODUCTION

1.1 A Noise Detection and Noise-Motion Separation

The dominant technology for digital image capture today is based on the charged
couple device (CCD) semiconductor. The CCD semiconductor has three basic func­
tions: Charge collection, charge transfer, and the conversion of charge into measurable
voltage. The building block of the CCD is the metal oxide semiconductor (MOS) ca­
pacitor, which is also called a gate. Manipulation of the gate voltages enables the
gate either to store or transfer charge. The MOS capacitor which stores the charge is
called photogate. Usually silicon photogates or photodiodes are used as electromag­
netic radiation detectors and CCD semiconductors are used for charge collection,
charge transfer, and conversion into measurable voltage. All of these are included
on a single chip. The most common detectors are sensitive to the visible region of
the spectrum. Cameras digitize in different resolutions. The basic building block of
the CCD is the metal-insulator-semiconductor (MIS) capacitor, which is also called
a gate. A very common MIS is the MOS. The charge created at a pixel site is pro­
portional to the incident light intensity and the exposure duration. A competing
technology is based on active pixels which are fabricated with complementary-metal-
oxide-semiconductor (CMOS). In this technology one or more active transistors are
integrated into the pixel. Their advantage is that each pixel is addressable and there­
fore one can perform digital image processing at the chip level. The photodetection
occurs as soon as the absorbed photon creates an electron-hole pair. The absorption
is inversely proportional to the wavelength of the incident light. General purpose
cameras usually have an infrared blocking filter to limit the camera's spectrum to that of the visible spectral band. Digital cameras have their own internal analog to digital converter (ADC). Single chip color cameras usually have an unequal number of red, green, and blue detectors. One of the problems of these cameras is aliasing. In order to reduce the aliasing problem digital cameras use anti-aliasing filters. To increase versatility and appeal to many users across the globe, many cameras provide output signal in NTSC, RGB, PAL and SECAM video formats. Many research papers have been written on motion estimation and motion compensation. Only a few of those references mentioned here are (Clarke, 1985), (L. D. Vos, M. Stegherr, and T. G. Noll, 1989), (E. A. Yfantis, M. Y. Au, and G. Miel, 1992), (E. A. Yfantis, et al 1998), (E. A. Yfantis, Et. al. 1997, 1996a, 1996b)

1.2 A Cancer Recognition Theory and Algorithm

As opposed to invasive methods such as X-rays and CAT-SCAN's, that all use cameras sensitive to radioactive material to obtain images of sections of the human body, ultrasound is non-invasive. In ultrasound the produced sound signal is directed via a probe to a section of the body. Part of the sound signal is absorbed, part of it is reflected back directly, and part of it bounces onto one or more points before is reflected back to the receiver. The received sound intensity is quantized usually on a gray scale between 0 and 255 and therefore transformed from the sound space to the image space. It is the image space signal that we operate on. The ultrasound signal is absorbed differently by the various parts of the body. Thus soft tissue reflects the signal differently than muscle or bone. So the fact that ultrasound is non-invasive allows the medical doctor to obtain several images of the part of the body which is of particular interest; furthermore, the acquisition of multiple images from different angles also allows for a 3-D image reconstruction using stereo image algorithms. Due to the multiple reflections of the sound signal (scattering of the signal) significant noise is incorporated with the signal. If we obtain several images of the same tissue sections by stabilizing the instrument and the patient then we can reduce the noise.
considerably. The changes in the noise component are positive and negative. They are the result of the fluctuation of the voltage, the fact that the patient is not totally still and due to the inability of the instrument to deliver two identical digital images (compared pixel by pixel) when the conditions remain constant. For k images obtained from the same view of the part of the body, the noise can be reduced by discarding the lowest and highest of the pixel values obtained and averaging the remaining k-2 values. This noise reduction is due to the fact that the positive and negative values of the noise will cross each other out and the signal will be the same for each replication. Furthermore the ultrasound is very inexpensive and therefore affordable by all medical practitioners. Due to the non-invasive nature of ultrasound no special precaution measures have to be taken. In addition, ultrasound machinery is smaller, and lighter than its counterparts (CAT-SCAN, X-rays, and others), demanding facile installation, use, maintenance and storage. An area of an ultrasound image of a prostate tissue can be classified as hypoechoic, isoechoic or hyperechoic depending on the intensity of the pixels in the area. Hypoechoic are the areas with relatively low pixel values, isoechoic the areas with pixel values about equal to the average pixel value of the image, and hyperechoic are areas with relatively high pixel values compared to the average pixel values of the image. In order to classify that a tissue area represented by an ultrasound image is cancerous or non-cancerous first we classify the area as hypoechoic, isoechoic, hyperechoic or as a mixture of more than one type. Second, if it is a mixture we separate the sections and we investigate each section independently from the other(s). A vector of attributes are defined for each area to be recognized as cancerous or non-cancerous. Subsequently a distance measure between the vector of attributes of the area of interest and two predefined centroids for that type of tissue, one for the cancerous and one for the noncancerous, are obtained. If the distance from the noncancerous centroid is smaller than that of the cancerous centroid, then the section of the image is classified as cancerous, otherwise if the distance between the section of the image attributes and the cancerous centroid is smaller than the image is classified as cancerous. Our algorithm classifies correctly if the section of the
tissue represented by the ultrasound image has cancer and we classify it as having cancer. or if the section of the tissue represented by the ultrasound image does not have cancer and we classify it as not having cancer. Our algorithm misclassified a section of a tissue represented by the ultrasound image if this section has cancer and the algorithm concludes that no cancer is present (false negative). or this section has no cancer and the algorithm concludes that cancer is present (false positive).
CHAPTER 2

A NOISE DETECTION AND NOISE-MOTION SEPARATION

2.1 Stochastic Attributes of Noise

As we pointed out in the abstract the noise of pictures captured by digital cameras depend on such environmental conditions as temperature, humidity, shock, vibration, air circulation, the current fluctuation of the camera's power source, etc. If we consider k-digital frames of digital video of the same image then the corresponding pixels in each one of the frames differ by the noise. The noise for color images is a three dimensional vector including the noise for the red, the noise for the green, and the noise the blue. The noise is wavelength depended. Since we have no evidence that the noise of a pixel in one frame correlates with the noise of a pixel in another frame of the video, we will assume that the noise of a pixel in one frame is independent of the noise of the same pixel in another frame. Another way of stating this is that we assume no temporal or time cross correlation of the noise. The noise associated with a pixel of a frame has three components: namely, one for red one for green and one for blue, which are correlated with one another. We will denote here by $\rho_{RB}$, $\rho_{RB}$, $\rho_{GB}$, the normalized correlation of the noise for red with green, red with blue and blue with green, respectively.

Theorem 2.1 If k-frames of the same image are produced with a digital camera and if the $(i,j)$ pixel in the frame $f$ has the value

$$(R_{ij}^f, G_{ij}^f, B_{ij}^f) = (R_{ij}, G_{ij}, B_{ij}) + (n_{Rij}^f, n_{Gij}^f, n_{Bij}^f)$$

where $(n_{Rij}^f, n_{Gij}^f, n_{Bij}^f)$ has mean vector consisting of all zeros and variance covari-
\begin{equation}
\Sigma = \begin{bmatrix}
\sigma_R^2 & \rho_{RG}\sigma_R\sigma_G & \rho_{RB}\sigma_R\sigma_B \\
\rho_{RG}\sigma_R\sigma_G & \sigma_G^2 & \rho_{GB}\sigma_B\sigma_G \\
\rho_{RB}\sigma_R\sigma_B & \rho_{GB}\sigma_B\sigma_G & \sigma_B^2
\end{bmatrix}
\end{equation}

Then the ensemble average

\begin{equation}
(R_{ij}^*, G_{ij}^*, B_{ij}^*) = \left( \frac{\sum R_{ij}^f}{k}, \frac{\sum G_{ij}^f}{k}, \frac{\sum B_{ij}^f}{k} \right)
\end{equation}

has mean \((R_{ij}, G_{ij}, B_{ij})\) and variance covariance

\begin{equation}
\frac{1}{k} \begin{bmatrix}
\sigma_R^2 & \rho_{RG}\sigma_R\sigma_G & \rho_{RB}\sigma_R\sigma_B \\
\rho_{RG}\sigma_R\sigma_G & \sigma_G^2 & \rho_{GB}\sigma_B\sigma_G \\
\rho_{RB}\sigma_R\sigma_B & \rho_{GB}\sigma_B\sigma_G & \sigma_B^2
\end{bmatrix}
\end{equation}

The difference between any two frames has mean vector consisting of all zeros, and variance covariance matrix

\begin{equation}
\frac{2}{k} \begin{bmatrix}
\sigma_R^2 & \rho_{RG}\sigma_R\sigma_G & \rho_{RB}\sigma_R\sigma_B \\
\rho_{RG}\sigma_R\sigma_G & \sigma_G^2 & \rho_{GB}\sigma_B\sigma_G \\
\rho_{RB}\sigma_R\sigma_B & \rho_{GB}\sigma_B\sigma_G & \sigma_B^2
\end{bmatrix}
\end{equation}

Proof.

We have that

\[ E(R_{ij}^*, G_{ij}^*, B_{ij}^*) = (R_{ij}, G_{ij}, B_{ij}) + E\left( \frac{\sum n_{Rij}^f}{k}, \frac{\sum n_{Gij}^f}{k}, \frac{\sum n_{Bij}^f}{k} \right) \]

or

\[ E(R_{ij}^*, G_{ij}^*, B_{ij}^*) = (R_{ij}, G_{ij}, B_{ij}) + \frac{\sum E(n_{Rij}^f)}{k}, \frac{\sum E(n_{Gij}^f)}{k}, \frac{\sum E(n_{Bij}^f)}{k} \]

Since

\[ E(n_{Rij}^f) = E(n_{Gij}^f) = E(n_{Bij}^f) = 0 \]

then

\[ E(R_{ij}^*, G_{ij}^*, B_{ij}^*) = (R_{ij}, G_{ij}, B_{ij}) \]

Since the noise between frames is independent the variance covariance matrix of the vector \((R_{ij}^*, G_{ij}^*, B_{ij}^*)\) is equal to

\begin{equation}
\frac{1}{k} \begin{bmatrix}
\sigma_R^2 & \rho_{RG}\sigma_R\sigma_G & \rho_{RB}\sigma_R\sigma_B \\
\rho_{RG}\sigma_R\sigma_G & \sigma_G^2 & \rho_{GB}\sigma_B\sigma_G \\
\rho_{RB}\sigma_R\sigma_B & \rho_{GB}\sigma_B\sigma_G & \sigma_B^2
\end{bmatrix}
\end{equation}
Now the difference between the corresponding pixels of any two frames $s$ and $r$ is:

$$(R_{ij}^s, G_{ij}^s, B_{ij}^s) - (R_{ij}^r, G_{ij}^r, B_{ij}^r) = (n_{R_{ij}^s}, n_{G_{ij}^s}, n_{B_{ij}^s}) - (n_{R_{ij}^r}, n_{G_{ij}^r}, n_{B_{ij}^r})$$

From the above we see that the mean vector of the difference has three zero components and the variance covariance matrix of the difference is:

$$
2 \begin{bmatrix}
\sigma_R^2 & \rho_{RG}\sigma_R\sigma_G & \rho_{RB}\sigma_R\sigma_R \\
\rho_{RG}\sigma_R\sigma_G & \sigma_G^2 & \rho_{GB}\sigma_G\sigma_G \\
\rho_{RB}\sigma_R\sigma_R & \rho_{GB}\sigma_G\sigma_G & \sigma_B^2 \\
\end{bmatrix}
$$

(2.5)

**Theorem 2.2** Consider any two consecutive frames $s$, $s+1$, and $s+2$, $s=1,2,...,k-1$. Consider the random vectors $(n_{R_{ij}}^s - n_{R_{ij}}^{s-1}, n_{G_{ij}}^s - n_{G_{ij}}^{s-1}, n_{B_{ij}}^s - n_{B_{ij}}^{s-1})$, and $(n_{R_{ij}}^{s-1} - n_{R_{ij}}^{s-2}, n_{G_{ij}}^{s-1} - n_{G_{ij}}^{s-2}, n_{B_{ij}}^{s-1} - n_{B_{ij}}^{s-2})$, then

$$E(n_{R_{ij}}^{s-1} - n_{R_{ij}}^{s})(n_{R_{ij}}^{s-1} - n_{R_{ij}}^{s+2}) = \sigma_R^2$$

$$E(n_{G_{ij}}^{s-1} - n_{G_{ij}}^{s})(n_{G_{ij}}^{s-1} - n_{G_{ij}}^{s+2}) = \sigma_G^2$$

and

$$E(n_{B_{ij}}^{s-1} - n_{B_{ij}}^{s})(n_{B_{ij}}^{s-1} - n_{B_{ij}}^{s+2}) = \sigma_B^2$$

Where $n_{R_{ij}}^s$ is the noise of the red component of the $(i, j)$ pixel at the $s$th frame, $n_{G_{ij}}^s$ is the noise of the green component of the $(i, j)$ pixel at the $s$th frame and $n_{B_{ij}}^s$ is the noise of the blue component of the $(i, j)$ pixel at the $s$th frame.

**Proof.**

We have that

$$E(n_{R_{ij}}^{s-1} - n_{R_{ij}}^{s})(n_{R_{ij}}^{s-1} - n_{R_{ij}}^{s+2}) = E(n_{R_{ij}}^{s-1}n_{R_{ij}}^{s-1}) - E(n_{R_{ij}}^{s-1}n_{R_{ij}}^{s+2})$$

$$-E(n_{R_{ij}}^{s}n_{R_{ij}}^{s+1}) + E(n_{R_{ij}}^{s}n_{R_{ij}}^{s+2})$$

Since

$$E(n_{R_{ij}}^{s+1}n_{R_{ij}}^{s+1}) = \sigma_R^2$$
It implies that

\[ E(n_{Rij}^{s+1}n_{Rij}^{s+2}) = E(n_{Rij}^{s}n_{Rij}^{s+1}) = E(n_{Rij}^{s}n_{Rij}^{s+2}) = 0 \]

Similarly

\[ E(n_{Bij}^{s+1} - n_{Bij}^{s})(n_{Bij}^{s+1} - n_{Bij}^{s+2}) = \sigma_B^2 \]

and

\[ E(n_{Gij}^{s+1} - n_{Gij}^{s})(n_{Gij}^{s+1} - n_{Gij}^{s+2}) = \sigma_G^2 \]

The above theorem provides us with the means to estimate the variance of the noise for each one of the three components. Thus an unbiased estimate of the noise can be obtained if we take the sums:

\[
\hat{\sigma}_R^2 = \frac{\sum(n_{Rij}^{s+1} - n_{Rij}^{s})(n_{Rij}^{s+1} - n_{Rij}^{s+2})}{M}
\]

where \( M \) is the number of products included in the above sum. The expected value of \( \sigma_R^2 \) is equal to \( \sigma_R^2 \) hence the \( \hat{\sigma}_R^2 \) is an unbiased estimate of \( \sigma_R^2 \). In an analogous fashion we can estimate the \( \sigma_G^2 \) and \( \sigma_B^2 \).

During digital video recording the pixels near where the motion takes place are affected by the disturbance of the flow of light due to the nearby motion and the loss of the equilibrium of the light. This phenomenon introduces a different type of noise whereby corresponding pixels of two consecutive frames have differences with mean other than zero. This bias once added or subtracted from the pixel of its corresponding consecutive frame produces a difference of mean zero. All the MPEG (Moving Picture Experts Group) algorithms use macroblocks of 16x16 pixels to assess motion between frames. However, no separation between motion and noise is made. Here we propose to classify the macroblocks in three categories. The first category consists of no motion and the difference between the pixels of the corresponding macroblocks of the two frames under consideration has mean zero and small variance. The second category consists of differences having a mean other than zero but if that mean is
subtracted from the macroblock of the second frame then the mean would be zero and if the variance is relatively small then the block can be characterized as a noise block with a bias. The third category consists of the macroblocks not included in the previous two categories and therefore include motion. In third category variance of the macroblock difference is relatively large. The larger the variance of the difference the more intense the motion is. After performing motion compensation for those blocks and obtain the difference we compress the difference which has smaller energy than the difference without the motion compensation.

2.2 Summary

A noise detection and noise-motion separation algorithm was described in this research. The difference between corresponding pixels of two frames subject to one type of noise, has mean vector equal to (0,0,0) and variance covariance matrix with relatively small variances for the (R, G, B) difference values. The other type of noise is a result of disturbance of the light equilibrium due to motion in neighboring or nearby pixels. In this type of noise the mean of the difference is non-zero so when the bias vector is subtracted from the (R, G, B) differences the new mean is zero. Every pixel not included in the one type of noise or the other is part of the motion set between the two frames. The pixels are organized in macroblocks so macroblocks containing pixels with motion are applied motion estimation and motion compensation methods first and subsequently the difference between the corresponding macroblocks of the two frames is obtained.

Motion compensation algorithms have not addressed properly the issue of motion detection. Also no separation is made between motion and noise. In this thesis we separate noise from motion. we also define a norm of the degree of motion. we estimate the motion and we provide a motion vector to compensate for the motion. A comparison of the motion compensation algorithm, which is a refinement of our previous work, with some of the most popular algorithms, is given. The motion vector provides better than half a pixel accuracy. Our approach is capable of separating
the motion from noise, provides better than half a pixel accuracy, and it takes less computations than such popular block matching algorithms as logarithmic search, conjugate direction search, three step search, and hierarchical search.

2.3 Introduction

In an experiment we performed we placed a CCD camera in the corner of a room with no windows so that the camera can view almost all the room. We closed the doors and we let the lights on. The camera recorded the room for five minutes at thirty frames a second and the images obtained were stored in a computer. The image resolution was 640 by 480 true color uncompressed video. After comparing the corresponding pixel (R.G.B) values of pairs of frames we found out that due to noise no two frames were alike. Since the camera was still and the room was closed and empty, theoretically all frames should be the same and therefore the (R.G.B) values of the corresponding pixels should be exactly the same. The histogram of the differences of the (R.G.B) values of the corresponding pixels had its maximum at zero and was symmetric about zero, declining very rapidly. The probability for a difference value to be less than -8 or more than 8 was very small. When we opened the door of the room from the outside due to the disturbance of the light equilibrium the probability distribution of the differences of the frames obtained with the room closed and with the door open introduced a shift in the positive or negative direction depending on the lighting of the room compared to the lighting of the hallway. Motion compensation algorithms, like the ones used in MPEG, perceived the noise introduced due to the change in the equilibrium of the light as motion and in the process of compensating introduced artifacts which distorted the restored video. The variance of the histogram difference of the Y-component is a good indicator if the macroblock under consideration is equal to the corresponding macroblock of the previous image plus noise, or the new macroblock is a result of motion of the pixels of the corresponding and neighboring macroblock of the previous frame, or the new macroblock represents totally new information not included in the previous frame. The logarithmic search
tracks block motion along the direction of minimum distortion. One at a time search, tracks the motion alternately horizontally and vertically. The three step search is a fine-coarse search. In the first step we move in all nine locations by four pixel in the direction of each of the eight locations. The ninth location is the center of the block and corresponds to no motion at all. We select the direction of motion with the minimum mean square error. In the second step we move by two pixels from the new locations in all eight locations. with the ninth corresponding to no motion. In the third step we move by one pixel in all nine locations. The composite direction with the minimum mean square error is the one we choose.

2.4 Motion Detection Motion Estimation Motion Compensation

As we pointed out above the noise has several components. For video captured indoors for example some of the variance components contributing to the noise are the fluctuation of the AC current, the inability of the light bulbs to deliver exactly the same luminance under constant and fixed current conditions, the inability of the charged coupled devises (CCD) to provide the same (R, G, B) values under constant lightening conditions and the inability of the video capture card to give the same output under the same input conditions. Another variance component that is of great importance especially in outdoors video capture is the fluctuation of the air density. Motion in a space could affect not only the moving parts of the video and the shading they cast, but other parts of the scene which are neither in motion nor they have shading of the moving parts casted onto them. This effect is due to the fact that the moving objects disturb the light equilibrium and the light flow. We can divide animation in two different categories: synthetic and natural. The synthetic category is computer graphics animation and the natural category is movies or animation obtained via a video camera. In the synthetic category, the pixel intensities are controlled by the computer graphics expert or user. Therefore, in computer graphics animation any changes in the intensity or color between two consecutive images is due to animation. In the natural category of animation any
changes between two frames could occur due to the changes in the light intensity. Light intensity changes are random phenomena which could be a result of changes in the voltage for artificial light or changes in the cloud coverage for natural light. Depending on the position of the light source(s) changes in the light intensity have different effect in different locations of the image. As we mentioned above we divide the two consecutive frames into corresponding small squares. In order to decide if two corresponding squares of the two consecutive images are effected by the noise we only compute the difference of the corresponding pixels. The distribution of the differences due to noise only, were found to have a relatively small mean, which represents the increase or decrease of the light intensity for that region of the image, and a relatively small variance. The distribution has one major mode which coincides with the mean. The distribution is also symmetric about the mean. The histogram of individual squares approximates this distribution. For the case of true color the trivariate distribution is centered in the vector mean of the differences, with variance covariance matrix having relatively small variances of the corresponding differences for \((R, G, B)\) and high correlation function between the \((R, G, B)\) components. In order to decide if the macroblock under consideration and its corresponding macroblock of a previous frame differ only by noise, we consider the variance of the histogram of the differences of the \(Y\)-component. If variance is less than a lower limit then the differences are due to the noise. The upper limit is chosen to be greater than the lower limit. If the variance is higher than the upper limit then the macroblock of the current frame and the corresponding macroblock of the previous frame are not the same. If much of the current frame was generated from the previous frame by moving the pixels of the previous frame then the motion vector for the macroblock of the current frame is estimated. If the variance of the current macroblock is between the lower limit and the upper limit then if most of the neighboring macroblocks were the result of motion of the macroblocks of the previous frame then the current macroblock is considered to be the result of motion of macroblocks of the previous frame and the motion compensation algorithm described below is used to estimate...
the motion vector.

It is easy to prove that any combination of translations, rotations, scalings and shearing transformations applied to a pixel with coordinates \((x, y)\)

results in an expression of the form:

\[
x' = ax + by + c \\
y' = dx + ey + f
\]

If the current frame is the \(n^{th}\) frame and the previous frame is the \((n - 1)^{th}\) frame and we consider the difference of the intensities

\[
\Delta I = I_{n-1}(x, y) - I_n(x, y)
\]

then if the point \((x, y)\) during the transition from the \((n - 1)\) frame to the \(n\) frame was subject to a transformation the difference can not be attributed to the noise or light change. In such a case the point \((x, y)\) of the \((n - 1)^{th}\) frame has become \((x', y')\) in the \(n^{th}\) frame, where:

\[
x' = ax + by + c \\
y' = dx + ey + f
\]

Thus:

\[
\Delta I = I_n(ax + by + c, dx + ey + f) - I_n(x, y)
\]  

We expand the \(I_n(ax + by + c, dx + ey + f)\) in the above expression in Taylor series about \((x, y)\) discarding the terms higher than the first degree. The parameters \(a, b, c, d, e, f\) in the above equation are estimated by minimizing the expected mean square error. Notice that our approach results in a translation if \(a = 1, b = 0, d = 0, e = 1\) which has been investigated many authors [2], [1], [3], [4], [5], [6], [7]. Our research presented here is also more general than the work presented in more recent papers.
\[ I_n(ax + by + c. dx + ey + f) = I_n(x, y) + \]
\[ \left( \frac{\partial I_n}{\partial x} \right)(ax + by + e - x) + \]
\[ \left( \frac{\partial I_n}{\partial y} \right)(cx + dy + f - y) + R \]  

(2.7)

(2.8)

\[ R \text{ in the above equation is the remaining terms of the Taylor series expansion plus the noise.} \]

Substituting equation (2) to equation (1) we obtain:

\[ \Delta I = \left( \frac{\partial I_n}{\partial x} \right)(ax + by + e - x) + \]
\[ \left( \frac{\partial I_n}{\partial y} \right)(cx + dy + f - y) + R \]  

(2.9)

An estimate of \( \frac{\partial I_n}{\partial x} \) is:

\[ \frac{\partial I_n}{\partial x} = I_n(x + 1, y) - I_n(x, y) = \Delta x \]  

(2.10)

\[ \frac{\partial I_n}{\partial y} = I_n(x, y + 1) - I_n(x, y) = \Delta y \]  

(2.11)

From the above we obtain:

\[ R = \Delta I - \Delta x(ax + by + e - x) - \Delta y(cx + dy + f - y) \]  

(2.12)

The \( R \) in the above equation depends on the parameters \( a, b, c, d, e, f \). The parameters \( a, b, c, d, e, f \) are optimal when the sum of \( R^2 \) is minimum.

Thus if we consider the function

\[ g(a, b, c, d, e, f) = \sum (\Delta I - \Delta x(ax + by + e - x) \]
\[ -\Delta y(cx + dy + f - y))^2 \]  

(2.13)

The system of six equations with six unknowns obtained from the above equation by taking the partial derivatives of \( g(a, b, c, d, e, f) \) with respect to each one of the
parameters $a, b, c, d, e, f$ and setting the resulted partial derivatives to zero is:

\[ a \sum (x \Delta x)^2 + b \sum xy(\Delta x)^2 \]
\[ + e \sum x(\Delta x)^2 + c \sum x^2 \Delta x \Delta y \]
\[ + d \sum xy \Delta x \Delta y + f \sum xy \Delta x = \]
\[ \sum \Delta I \Delta x + \sum (x \Delta x)^2 \]
\[ + \sum xy \Delta x \Delta y \] (2.14)

\[ a \sum xy(\Delta x)^2 + b \sum (y \Delta x)^2 \]
\[ + e \sum y(\Delta x)^2 + c \sum xy \Delta x \Delta y \]
\[ + d \sum y^2 \Delta x \Delta y + f \sum y \Delta x \Delta y = \]
\[ \sum y \Delta I \Delta x + \sum xy(\Delta x)^2 \]
\[ + \sum y^2 \Delta x \Delta y \] (2.15)

\[ a \sum x(\Delta x)^2 + b \sum y(\Delta x)^2 \]
\[ + e \sum (\Delta x)^2 + c \sum x \Delta x \Delta y \]
\[ + d \sum y \Delta x \Delta y + f \sum \Delta x \Delta y = \]
\[ \sum \Delta I \Delta x + \sum x(\Delta x)^2 \]
\[ + \sum y \Delta x \Delta y \] (2.16)

\[ a \sum x^2 \Delta x \Delta y + b \sum xy \Delta x \Delta y \]
\[ + e \sum x \Delta x \Delta y + c \sum (x \Delta y)^2 \]
\[ + d \sum xy(\Delta y)^2 + f \sum x(\Delta y)^2 = \]
\[ \sum x \Delta I \Delta y + \sum xy(\Delta y)^2 \]
\[ + \sum x^2 \Delta x \Delta y \] (2.17)
\[
\begin{align*}
&\ a \sum xy \Delta x \Delta y + b \sum y^2 \Delta x \Delta y \\
&\ + e \sum y \Delta x \Delta y + c \sum xy (\Delta y)^2 \\
&\ + d \sum (y \Delta y)^2 + f \sum y (\Delta y)^2 = \\
&\ \sum y \Delta I \Delta y + \sum xy \Delta x \Delta y \\
&\ + \sum (y \Delta y)^2 \\
\end{align*}
\]
\(2.18\)

\[
\begin{align*}
&\ a \sum x \Delta x \Delta y - \sum y \Delta x \Delta y \\
&\ + e \sum \Delta x \Delta y + c \sum x (\Delta y)^2 \\
&\ + d \sum y (\Delta y)^2 + f \sum (\Delta y)^2 = \\
&\ \sum \Delta I \Delta y + \sum y (\Delta y)^2 \\
&\ + \sum x \Delta x \Delta y \\
\end{align*}
\]
\(2.19\)

Notice that if the solution of the above system is \(a = 1, b = 0, c = 0, d = 1\) then the motion corresponds to translation. Thus our algorithm includes as a particular case the translation algorithms. If \(b = 0, c = 0, e = 0, f = 0\) then it corresponds to scaling. If \(a = d, b = -c\) and if the determinant of the two by two matrix with first row \(a b\) and second row \(c d\) is equal to one then the motion corresponds to rotation. In general the motion would be a composition of one or more transformations. The algorithm was tested using the image of Lisa. Lisa blinks her eyes and rotates her head as she is talking on the phone. Our algorithm was able to detect the parts of the consecutive frames which were moving and those with no motion. Furthermore, it was able to detect the type of motion and compensate appropriately for it. The coder is given the motion vector. In order to estimate the motion vector, we apply the center coordinates \((x_c, y_c)\) of the macroblock in the equations

\[
x_c' = ax_c + by_c + c \\
y_c' = dx_c + ey_c + f
\]
then the estimate of the motion vector is

\[ u_x = x'_c - x_c \]
\[ u_y = y'_c - y_c \]

The above estimate is a real number as such it has accuracy better than half a pixel.

The method presented here is based on the estimation of the six parameters described above and is much faster than the trial and error methods used by other motion compensation algorithms. After motion estimation vector is found for every of the corners of the macroblock an average value of the motion vector is found between its own vector and the vectors of the adjacent macroblocks. This information is transmitted to the decoder. The decoder now having four vectors for each of the macroblocks, performs a bilinear interpolation of the pixel values within each macroblock. This gives us a capability to describe an affine transformations on the macroblocks, such as rotation and scaling. Furthermore, our method is capable of discriminating between noise and motion. In case of noise if the mean of the difference of the R components is not zero and therefore shifted in the positive or negative direction then we add that shifted value to every R component of the macroblock.

### 2.5 Summary and Conclusions

A motion detection algorithm was presented in this thesis. We also introduce a method to separate noise from motion. If the noise has mean zero the current macroblock is perceived to be the same as the macroblock of the previous frame and therefore the current macroblock is not being encoded. If the noise has non-zero mean then the current macroblock is not encoded but the non-zero mean is passed to the encoder. When the decoder reconstructs this macroblock and the encoder adds to the corresponding macroblock of the previous frame the shifted value. Furthermore, besides the translation we consider the other transformations such as rotation and scaling in our method. This greatly reduces the entropy of the resulting signal.
CHAPTER 3

A CANCER RECOGNITION THEORY AND ALGORITHM

3.1 Introduction

The univariate distribution function for each one of the attributes used for recognition is obtained both as a histogram and as a mathematical function with the appropriate parameters. Furthermore, the variance covariance matrix of the multivariate distribution of all the attributes is estimated. Also the centroid of the distribution in the multidimensional space is computed. Although the distribution of each one of the attributes is not normal, which implies that their multivariate distribution is also not normal, the distribution of their centroid is normal due to the extension of the central limit theorem to the multivariate space. The normalized cross-correlation function between attributes is scale independent and expresses the structural characteristics of the section of the image that the data was obtained from. The principal compo-
nent analysis of the attributes used produces eigenvectors with directions depended on presence or absence of cancer in the tissue. Research related to this area is sparse and some of the papers are [8], [9], [10], [11], [12], [13], [14]. The attributes considered here are the mean of the section of the tissue, the range, the estimated parameters of the probability distribution function of the autocorrelation function with lag one and the first ten components of the cepstra.

3.2 The Recognition Algorithm

In our previous work we considered a sample support of 20 by 20 pixels; here we consider every pixel as the center of a square with side 21 pixels. Our theory is that cancer creates discontinuities in the tissue. In order for cancer to survive it develops its own blood supply system which is different than the supply system of normal tissue. The velocity of the blood flowing through the cancerous blood vessels is different than the velocity of the blood flowing through blood vessels of normal tissue. Due to this fact the ultrasound signal is absorbed differently in the cancerous areas than in the normal tissue areas. The energy of the signal, the continuity of the signal, autocorrelation function, frequency domain properties, are different in the normal tissue than in cancerous tissue. All these indicators are weighted here for the purpose of classifying the image of the tissue as being cancerous or non-cancerous. The discontinuities alter the probability distribution of the autocorrelation function. The distribution of the autocorrelation function of a cancerous tissue (explained below) is somewhere between that of independent random variables and the distribution of the autocorrelation function of an image representing healthy tissue. Notice that the normalized autocorrelation function is scale independent. The distributions of the pixels in a neighborhood representing cancerous tissue is shifted towards the small values. This probability distribution function is modeled here using a gamma probability distribution function with parameters $\alpha$ and $\beta$. For the case of cancerous area the parameter $\alpha$ is very close to one, so the distribution function approaches the exponential probability distribution function. The gamma probability distribution
function is of the form
\[ f(x) = \frac{x^\alpha e^{-x/\beta}}{\beta^\alpha \Gamma(\alpha)} \]  
where \( \alpha > 0 \), \( \beta > 0 \). The mean \( \mu \) of the above distribution is \( \mu = \alpha \beta \) and the variance \( \sigma^2 = \alpha \beta^2 \). Estimates of the parameters \( \alpha > 0 \), \( \beta > 0 \), can be obtained using the method of moments. Let \( \hat{\mu} \) be the estimate of the mean and \( \hat{\sigma}^2 \) be the estimate of the variance. \( \hat{\mu} = \hat{\alpha} \hat{\beta} \) and \( \hat{\sigma}^2 = \hat{\alpha} \hat{\beta}^2 \). From the above equations we obtain
\[ \hat{\alpha} = \frac{\hat{\mu}^2}{\sigma^2} \]  
\[ \hat{\beta} = \frac{\hat{\sigma}^2}{\mu} \]  
(3.2)  
(3.3)

The parameter \( \alpha \) is scale independent.

For hypoechoic areas with cancer \( \hat{\alpha} = 1 \), and \( \hat{\beta} = \hat{\sigma} \)

**Lemma 3.1** Let \( X_1, X_2, X_3, \ldots, X_n \) be independent identically distributed random variables with mean zero and variance \( \sigma_X^2 \) then the distribution of the random variable \( Y = X_i X_{i-k} \) has mean zero and variance equal to \( \sigma_Y^2 = \sigma_X^4 \).

**Proof.**

The mean \( \mu_Y = E(X_i X_{i-k}) = E(X_i)E(X_{i-k}) = 0 \). The variance of \( Y \) is
\[ \sigma_Y^2 = E(Y^2) = E(X_i^2)E(X_{i-k}^2) = \sigma_X^4 \]  
(3.4)

As we see here the variance is independent of the space lag.

**Theorem 3.1** Let \( X_1, X_2, X_3, \ldots, X_n \) be a sequence of random numbers obtained from a wide sense stationary random process with mean zero, and variance \( \sigma_X^2 \). Let also the space lag between \( X_i \) and \( X_{i+1} \) be fixed and denoted by \( \Delta X \). Then the random variable \( Y = X_i X_{i-k} \) has mean positive and depended on the lag \( k \). As \( k \) increases the mean of \( Y \) approaches to zero. The variance \( \sigma_Y^2 = E(X_i^2 X_{i-k}^2) - R^2(k) \) and \( E(X_i^4) > \sigma_Y^4 \), \( E(X_i^4) > \sigma_X^4 \).
Proof.

We have that

\[
E(X_i^2 - E(X_i^2))^2 = E(X_i^4) - 2X_i^2E(X_i^2) + E^2(X_i^2)
\]

from the above equation we obtain

\[
E(X_i^2 - E(X_i^2))^2 = E(X_i^4) - E^2(X_i^2)
\]  \hspace{1cm} (3.5)

or

\[
E(X_i^2 - E(X_i^2))^2 = E(X_i^4) - \sigma_{X_i}^4
\]  \hspace{1cm} (3.6)

From the above equation, since the left side is nonnegative we obtain

\[
E(X_i^4) > \sigma_{X_i}^4
\]  \hspace{1cm} (3.7)

Now

\[
\sigma_{Y_i}^2 = E(X_i^2X_{i-k}) = E(X_{i-k}^2) - R^2(k)
\]  \hspace{1cm} (3.8)

as the space lag increases the correlation between \(X_i, X_{i-k}\) becomes smaller and tends to zero in which case

\[
\sigma_{Y_i}^2 = E(X_i^4)
\]  \hspace{1cm} (3.9)

on the other hand as \(k\) goes to zero

\[
\sigma_{Y_i}^2 = E(X_i^4) - \sigma_{X_i}^4
\]  \hspace{1cm} (3.10)

In cancerous images \(\sigma_{Y_i}^2\) is relatively large and very close to \(E(X_i^4)\). The autocorrelation function for any lag has positive and negative values for cancerous images.

\(\square\)
Theorem 3.2 Consider the pixels $X_{ij}$ and $X_{ij-1}$. let $Y = X_{ij} - X_{ij-1}$. then the 
$E(Y) = E(X_{ij} - X_{ij-1}) = 0$ and $\sigma_Y^2 = 2\sigma_X^2(1 - \rho)$ and $\frac{\sigma_Y^2}{\sigma_X^2} = 2(1 - \rho_1)$

\[ \sigma_Y^2 = E(X_{ij} - X_{ij-1})^2 = 2\sigma_X^2(1 - \rho) \quad (3.12) \]
and

\[ \frac{\sigma_Y^2}{\sigma_X^2} = 2(1 - \rho_1) \quad (3.13) \]

Theorem 3.3 Now consider the pixel $X_{ij}$ and the pixels $X_{ij-1}$, $X_{ij+1}$, $X_{ij-1}$, $X_{ij+1}$ which have lag 1 from $X_{ij}$ then consider $Y = X_{ij} - 0.25(X_{ij-1} + X_{ij+1} + X_{ij-1} + X_{ij+1})$
then the expected variable of $Y$ is zero and the variance of $Y$ is

\[ \sigma_Y^2 = \sigma_X^2(1.25 - 2\rho_1 + 0.25\rho_2 + 0.5\rho_3) \quad (3.14) \]

Therefore

\[ \frac{\sigma_Y^2}{\sigma_X^2} = 1.25 - 2\rho_1 + 0.25\rho_2 + 0.5\rho_3 \quad (3.15) \]

The estimate and distribution of $\sigma_Y^2$ is different for cancerous and non-cancerous areas. Furthermore the distribution of the normalized data $\frac{r_{X_{ij-1}X_{ij}}}{\sigma_Y^2}$ is different for the cancerous and non-cancerous areas.

From the above we infer that images with continuity and strong autocorrelation have normalized distribution of the lag product with variance which is greater than 1, where as the correlation decreases due to the discontinuities the normalized distribution of the lag product has variance close to 1. That implies if we consider the histogram of the normalized lag product multiplied by a factor 10, then in the non-cancerous areas we expect more energy namely bigger spread than the cancerous areas.

Also from the above, since the variance of the edges is higher when the continuity breaks down, cancerous areas are expected to have more edges and the inter-arrival
time between edges is smaller. We define an activity level associated with a pixel to the number of edges emanating from that pixel. Thus the activity level of cancerous areas is different than the activity level of non-cancerous areas. The attributes associated with one area constitute a vector in the multivariate space. Based on a relatively large sample of hypoechoic normal areas and hypoechoic cancerous areas we calculate the centroid of the above defined attributes and their associated variance covariance matrices. For a new hypoechoic section of an image we estimate the above attributes and take the distance from the cancerous centroid and the non-cancerous centroid. We classify the section of the image as cancerous if the distance of the vector of attributes of the section has smaller distance from the cancerous centroid than the non-cancerous centroid; otherwise we classify the section as non-cancerous. The recognition can be achieved by calculating the principal components of the variance covariance matrices of the cancerous, non-cancerous and test image-section and then taking the Mahalanobis distance of the test image-section from the cancerous and non-cancerous centroids.

1. The estimate of the parameters $\alpha$ and $\beta$ of the probability distribution function.

2. The variance of the pixels within the block represents the energy of the block, and in general cancerous blocks have less energy than non-cancerous blocks.

3. If

$$Y_{ij} = r_{ij} - \bar{x} \quad (3.16)$$

$$S_Y^2 = \frac{\sum_{i=1}^{21} \sum_{j=1}^{21} Y_{ij}^2}{411} \quad (3.17)$$

and

$$r_{ij} = \frac{Y_{ij}}{S_Y} \quad (3.18)$$

then the histogram of the autocorrelation function $\rho_1$ of $r_{ij}$ depends on the presence of cancer or its absence in the block.
4. The autocorrelation function and the variance of the autocorrelation function is another indicator.

5. $X_{ij}$ are the pixel values of the block. Let $Y_{ij} = x_{ij} - x_{i-1,j}$ then $E(Y_{ij}) = 0$ and variance of $Y_{ij}$ is $\sigma_Y^2 = 2\sigma_X^2 (1 - \rho_1)$ and

$$\frac{\sigma_Y^2}{\sigma_X^2} = 2(1 - \rho_1) \tag{3.19}$$

The presence of cancer reduces the $\rho_1$ so the above ratio is larger for cancer than non-cancer.

6. If we consider the

$$Y_{ij} = x_{ij} - 0.25(x_{i,j-1} + x_{i,j-1} + x_{i-1,j} + x_{i-1,j}) \tag{3.20}$$

then

$$\frac{\sigma_Y^2}{\sigma_X^2} = 1.25 - 2\rho_1 + 0.25\rho_2 + 0.5\rho_3 \tag{3.21}$$

For non-cancerous this is close to zero where for cancerous blocks is close to 1.

Figure 3.2: Histogram of 21x21 block with no cancer.

An area of interest can be classified as hypoechoic, hyperechoic, isoechoic, or if it is a mixed area it can be divided into its basic component areas by estimating the
mean pixel intensity and the range. All the attributes form a vector. This vector is compared to two different spaces: one space is the cancerous space for these types of areas (hypoechoic, isoechoic or hyperechoic), the other is the non-cancerous space. We calculate the distances of the vector with the two spaces and we classify the vector as belonging to the space with the minimum distance. For example, the way we calculate the cancerous space for hypoechoic images is by obtaining hypoechoic image sections with cancer and by estimating the attribute vector for each image; then we use these attribute vectors to estimate the centroid and the variance covariance matrix of the hypoechoic cancerous sections. We repeat the process for the hypoechoic non-cancerous image sections. Thus we can use these estimates to classify an unknown hypoechoic image section. A similar process can be used for the isoechoic and hyperechoic sections. Preliminary results based on limited number of ultrasound images show that our theory works. Each pixel of the tested images is the center of a 21x21 block. The stochastic attributes described above are computed for that block and based on the result, the block is classified as cancerous or non-cancerous. Figure 3.4 shows a position of the 21x21 block. The moving block intellectually is similar to the needle testing except that it is non invasive. Figure 3.1 shows the part
of the image that has cancer based on our algorithm. Figures 2 and 3 show the histogram of a cancerous block and that of a non-cancerous block, respectively.

![Image of cancer recognition](image)

Figure 3.4: The 21x21 moving pixel spyglass has recognized cancer.

### 3.3 Summary and Conclusions

A cancer recognition algorithm of ultrasound prostate tissue was developed in this thesis. The algorithm is based on calculating a number of attributes in the section of interest of the image and finding the distance in the multivariate space of the vector of attributes of the current section from two predetermined centroids, one for non-cancerous regions of the same type and one obtained from cancerous regions of the same type (isoechoic, hypoechoic or hyperechoic). If the distance from the noncancerous centroid is smaller than the distance from the cancerous centroid, the region is classified as noncancerous, otherwise it is classified as cancerous.
CHAPTER 4

AN APPLICATION OF NOISE DETECTION AND SEPARATION ALGORITHM

4.1 A Video Surveillance and Security System

In this chapter we describe the application that uses Noise Detection and Noise-Motion Separation algorithm described in Chapter 2 as one of its major components.

4.2 An Introduction

In this day and age one's personal security as well as security of one's assets plays very important role in our society. It is natural that with the today's powerful computers we try to replace old analog video recording technology with the modern digital technology. In attempt to do so we have to cross many obstacles such as noise detection and noise-motion separation as well as an object recognition. Since the video signal carries a lot of redundant information it is natural that we try to reduce the amount of this information in order to save storage space and/or reduce the bandwidth requirements for the video transmission. The theories developed above help us to greatly reduce the amount of the redundant information in the video. We have developed the remote video surveillance and security system.

4.3 Design of a video surveillance and security system

As shown in Figure 4.1 our system consists of two main parts: Video Camera Server and Video Camera Client. These two separate systems are connected through Wide Area Network (WAN) or Local Area Network (LAN). A specialized protocol was developed to allow the error-free and real-time data flow from the client to the
Figure 4.1: The Security System Architecture.
server and vice versa. The system was developed to be run under Microsoft Windows operating systems, but it is designed to be easily ported to other platforms and OS-es such as Linux, UNIX and MAC-OS.

4.3.1 The Server

The server machine contains our multiple camera PCI video capture card that we also developed. This hardware card converts analog video signals that are presented at its input to digital uncompressed form that is suitable for further analysis and processing. The video capture card can handle up to eight different video signals. These signals are time multiplexed and captured in round-robin fashion. The server also contains a RS422 PCI card that allows the communications with the Pelco series in-ceiling pan, tilt and zoom dome cameras. The Remote Watch Server software has built in all the drivers needed to allow the program to capture a frame as well as to send the specific command to each of the eight cameras. The server machine is configured with the static IP address.

4.3.2 The Client

The client machine does not need to have a static IP address but it needs to have an access to the network that the server machine is on (WAN, LAN or ATM). The client is implemented fully in software. The client software accepts and processes all the user input, compresses it and sends to the server machine over the appropriate network protocol. The client software also decompresses the frames received from the server and updates the corresponding camera view windows.

4.4 Implementation

The video server and client are implemented as a single program that acts as one of them depending how its ran. The code written to be compiled with Microsoft Visual C++ 6.0. The video capture was done through Microsoft Video For Windows
API calls, and networking using MFC CSocket classes. The source code is included in Appendix ???. The sections are broken down by the code logic and functionality to ease its reading and understanding.

4.5 Conclusions and Future Work

4.5.1 Conclusions

A result from this work is that the noise detection and noise-motion separation can be used in many areas including the video security. Here we have shown that our algorithm performs well both in speed and in recognizing motion, and therefore making it suitable for the real-time applications. Furthermore, our implementation of the above described algorithm can be easily added to the existing systems and ported to a new platforms. The existing systems just to name a few are: video telephony, security, computer vision, computer automation, target tracking, etc.

4.5.2 Future Work

Future work can be done on many areas of the currently implemented system:

- **Video compression**: The server’s video compressor can support different compression rates and different codecs.

- **Networking**: The RTSP (Real-Time Streaming Protocol) could be used in all client-server communication to further minimize the delay caused by currently used and slower TCP/IP network protocol.

- **Server scalability**: A server could be designed in such a way that will provide digital video streaming to multiple clients at multiple bit rates, thus allowing the users to use wide spectrum of connection speeds ranging from modem at 56kbits/s, all the way to T1.

- **Server security**: A server-client connection could be designed to use some form of secure communication, implemented using some form of encryption.
• **More simultaneous video inputs:** The hardware video capture card could be expanded to allow for connection of more video cameras. So instead of currently used eight inputs we could design a card with total of 25 video inputs.
CHAPTER 5

AN APPLICATION OF A CANCER RECOGNITION THEORY AND ALGORITHM

5.1 The Recognition of Cancerous Tissue

We developed a software that uses A Cancer Recognition Theory and Algorithm described in Chapter 3. This program detects a cancerous tissue by analyzing ultrasound images of the prostate. The program was designed in a way that allows to be used interactively or as an automated system. In the interactive mode the user, using the mouse input positions the cursor over the area of interest. The program then crops the area of the image around the cursor and analyze it. In the automated mode the program itself searches for the possible cancerous areas of the image, detects them and paints them in different and distinct colors.

5.2 Conclusions and Future Work

5.2.1 Conclusions

The cancer recognition program performs well on the ultrasound images of the prostate. We are successfully detecting the areas of the images that are cancerous using our algorithm described in Chapter 3.

5.2.2 Future Work

The program however, could be improved. Future work can be done on different parts of the current program.
• **Input File Filters:** The cancer recognition program could be modified to accept the images in file formats other than bmp. One of the most used formats in the medical field is DICOM file format. Our program can be extended to use this industry standard image file format.

• **Portability:** The cancer recognition program could be ported to other platforms.

• **Automation:** The cancer recognition program could be modified to work completely automatic without any user intervention.

• **On-Line Analysis:** Furthermore, our program could be made accessible to doctors all around the globe by building a WEB interface into it.

• **Reliability:** The recognition algorithm could be further improved by taking into consideration more features that could be obtained by scanning the same part of the patient’s body using an ultrasound of different frequencies. This method would allow for multi-resolution analysis and thus better performance in cancer classification.
BIBLIOGRAPHY


APPENDIX

Graphical User Interface (GUI)

CRemoteWatch definition file

```c
#ifndef _REMOTEWATCH_H_
define _REMOTEWATCH_H_
#endif

#ifndef __AFXWIN_H__
  #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h" // main symbols
#include "VersionInfo.h"
#include "RegSettings.h"
#define GetRS((CRemoteWatchApp*)AfxGetApp())->m_rs
#define GetV((CRemoteWatchApp*)AfxGetApp())->m_v1

class CRemoteWatchApp : public CWinApp
{
  public:
    VersionInfo *m_v1;
    CRegSettings *m_rs;
    CRemoteWatchApp();
    void GetMyIPAddress(CString &ip);
    void InstallCodec();
    void RemoveCodec();

  // Overrides
  // ClassWizard generated virtual function overrides
  ////{{AFX_VIRTUAL(CRemoteWatchApp)
  public:
    BOOL InitInstance();
    BOOL OnIdle(ULONG lCount);
    virtual int ExitInstance();
 //}}AFX_VIRTUAL

  // Implementation

  ////{{AFX_MSG(CRemoteWatchApp)
  // NOTE - the ClassWizard will add and remove member functions here.
  // DO NOT EDIT what you see in these blocks of generated code !
  ////}}AFX_MSG
  DECLARE_MESSAGE_MAP()

  #endif

CRemoteWatchDlg implementation file

```
// NOTE - the ClassWizard will add and remove mapping macros here.  
// DO NOT EDIT what you see in these blocks of generated code!

//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////////////// 
// CRemoteWatchApp construction
CRemoteWatchApp::CRemoteWatchApp()
{
  INIT_PTR(m_vi);
  INIT_PTR(m_rs);
}

///////////////////////////////////////////////////////////////////////////// 
// The one and only CRemoteWatchApp object
CRemoteWatchApp theApp;

///////////////////////////////////////////////////////////////////////////// 
// CRemoteWatchApp : initializaion

BOOL CRemoteWatchApp::InitInstance()
{
  if(!AfxSocketInit())
  {
    CMString str(IDS_FATAL_SOCKET_INIT_FAILED);
    AfxMessageBox(str, MB_ICONSTOP);
    return FALSE;
  }
  // create version info object...
  m_vi=new VersionInfo;
  OutputDebugString("Program Startup");
  // open our registry settings folder key...
  m_rs=new CRegSettings;
  if(!m_rs->Open(IDS_APP_REG_KEY,AfxAppName()))
  {
    DELETE_PTR(m_vl);
    CMString str(IDS_FATAL_RS_OPEN);
    AfxMessageBox(str, MB_ICONSTOP);
    return FALSE;
  }
  #ifdef _AFXDLL
  Enable3dControls();   // Call this when using MFC in a shared DLL
  #else
  Enable3dControlsStatic();  // Call this when linking to MFC statically
  #endif

  // install the codec to use...
  InstallCodec();

  if(GetRS()->LoadString(IDS_CONNECT_MODE)="/Server" ||
    !GetRS()->LoadDWORD(IDS_SERVER_AUTOSTART))
  {
    // Popup program about-box...
    CAaboutDlg dlg;
    dlg.DoModal();
  }
  // Our codeless dialog box application...
  m_pMainWnd=new CRemoteWatchD1g;
  ((CRemoteWatchD1g*)m_pMainWnd)->Create(CRemoteWatchD1g::IDD);
  if(IS_VALID_PTR(m_pMainWnd))
  {
    m_pMainWnd->ShowWindow(SW_SH0W);
  
  }
int CRemoteWatchApp::ExitInstance()
{
    DELETE_PTR(m_rs);
    DELETE_PTR(m_v1);
    return CWinApp::ExitInstance();
}

void CRemoteWatchApp::InstallCodec()
{
    CString AppPath(m_pszHelpFilePath), cFile;
    CMString cHttpIDS.CDDEC_HANDLER_FMT);
    CMString cHttpIDS.CDDEC_FILE_FMT);
    for(int i = 1; i<256;++i)
    {
        CString cH;
        cH.Format(cHttpIDS(i));
        DWORD ch = GetRSC()->LoadDWORD((LPCTSTR)cH);
        if(ch==0)
            break;
        CSF cF;
        cF.Format(cHttpIDS, i);
        cFile.Format(AppPath.Left(AppPath.ReverseFind('.'))+
                     GetRSC()->LoadString(cF));
        ICInstall(ICY_TYPE_VIDEO,
                    ch,
                    (LPARAM)(LPCTSTR)cFile,
                   (fabsm, INSTALL_DRIVER);
    }
}

void CRemoteWatchApp::RemoveCodec()
{
    ICRemove(ICY_TYPE_VIDEO,GetRSC()->LoadDWORD(IDS_CODEC_HANDLER),0);
}

// this one connects to the remote site and gets our IP address
// every time you call this function it will retrieve the
// current IP address of the server
void CRemoteWatchApp::GetMyIPAddress(CString &ip)
{
    CInternetSession session;
    CMString IPUrl(IDS_IP_URL);
    CHttpConnection *server = session.GetHttpConnection(IPUrl);
    CMString IPPFile(IDS_IP_FILE);
    CHttpFile *f=server->OpenRequest(CHttpConnection::HTTP_VERB_GET,
          IPPFile, NULL, NULL, NULL,
          INTERNET_FLAG_EXISTING_CONNECT,
          INTERNET_FLAG_DONT_CACHE,
          INTERNET_FLAG_RELOAD);
    DWORD dwReturnCode;
    if(f->SendRequest())
    {
        if(f->QueryInfoStatusCode(dwReturnCode))
            if(dwReturnCode==200)
            {
                f->SetReadBufferSize(2000);
                CString line;
                CString text;
                while(f->ReadString(line))
                    text=text+line;
                CString searchStr="<body>";
int s=text.Find(searchStr,0)+searchStr.GetLength();
int e=text.Find("\n",s);
p=text.Mid(s,e-s);
}
f->Close();
DELETE_PTR(f);
DELETE_PTR(server);
session.Close();
}

BOOL CRemoteWatchApp::OnIdle(LONG lCount)
{
    BOOL res=CWinApp::OnIdle(lCount);
    ((CRemoteWatchDlg*)m_pMainWnd)->OnIdle();
    return TRUE;
}
CRoemoteWatchDlg definition file

```cpp
#define _REMOTEWATCHDLG_H_
define _REMOTEWATCHDLG_H_

#include "CaptWnd.h"
#include "PlayWnd.h"
#include "Csockobj.h"
#include "Tranobj.h"

CRoemoteWatchDlg dialog

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

    // Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };  
    CStatic _programinfo;
   //}}AFX_DATA

    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
   //}}AFX_VIRTUAL

    // Implementation
    protected:
        //AFX_MSG(CAboutDlg)
        DECLARE_MESSAGE_MAP()
    ;

}/******
 // CMyObjectSocket
class CMyObjectSocket: public CObjectSocket
{
public:
    CMyObjectSocket() : CObjectSocket() {} 
    // void OnForcedDrop();
};

CRoemoteWatchDlg dialog

#define MAX_CAM 8

class CRoemoteWatchDlg : public CDialog
{
public:
    CMyObjectSocket m_ObjSocket;
    CaptWnd m_CaptWnd;
    PlayWnd m_PlayWnd[MAX_CAM];
    CEvent m_cev_m_trhexit;
    char m_packet[PACKET_LEN];
    int m_type,m_codecidx;
    int m_idle_sleep;
    enum{T.NONE=1,T.SERVER,T.CLIENT};

    public:
        // Construction
        CRoemoteWatchDlg(CWnd* pParent = NULL);
        void UpdateStatus();
```
void OnDlglIdle();
void SetFocusWindow(PlayWnd *, BOOL flag);

// Dialog Data
//}}AFX_DATA(CRemoteWatchDlg)
enum { IDD = IDD_REMOTEWATCH_DIALOG };
CListBox m_list;
CButton m_connectb;
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//}}AFX_VIRTUAL(CRemoteWatchDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);
virtual void OnDestroy();
//}}AFX_VIRTUAL

// Implementation
protected:
HICON _hIcon;
void EnableSysMenuItem(int menuId, int flag);

// Generated message map functions
//}}AFX_MSG(CRemoteWatchDlg)
virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
afx_msg void OnServer();
afx_msg void OnClient();
afx_msg void OnDisconnect();
afx_msg BOOL OnMouseWheel(UINT nFlags, short zDelta, CPoint pt);
afx_msg void OnCodecConfigure();
afx_msg void OnVideoInput();
virtual void OnCancel();
virtual void OnOK();
afx_msg void OnConnect();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

CRemoteWatchDlg implementation file

#include "stdafx.h"
#include "resource.h"
#include "RemoteWatch.h"
#include "RemoteWatchDlg.h"
#include "Connect.h"
#include "ClientConnect.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

void CMyObjectSocket::OnForcedDrop()
{
    CObjectSocket::OnForcedDrop();
    ((CRemoteWatchDlg*)((AFXApp->pMainWnd))->UpdateStatus();
}
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //AFX_DATA_INIT(CAboutDlg)
    //AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //AFX_DATA_MAP(CAboutDlg)
    DDX_Control(pDX, IDC_PROGRAMINFO, m_programinfo);
    //AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    // No message handlers
END_MESSAGE_MAP()

BOOL CAboutDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    CString prginfo;

    CString ver = GetVI()->FileVersion();
    // get the version
    ver.Replace('.', ' ');
    CString build = ver.Left(ver.ReverseFind('.') + 1);
    ver = ver.Left(ver.ReverseFind('.'));

    prginfo = GetVI()->ProductName() + 
                "Version " + ver + " (build " + build + 
                "Developed by " + GetVI()->CompanyName() + 
                "' + GetVI()->Copyright() + "' + GetVI()->Comments();
    m_programinfo.SetWindowText(prginfo);
    return TRUE;
}

// CRemoteWatchDlg dialog
CRemoteWatchDlg::CRemoteWatchDlg(CWnd* pParent /*=NULL*/) : CDialog(CRemoteWatchDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    for(int i = 0; i < MAX_CAM; i++)
    { // initialize channels
        m_playWnd[i] = .type = T_NONE;
        memset(m_packet, 0, PACKET_LEN);
    }

    void CRemoteWatchDlg::DoDataExchange(CDataExchange* pDX)
    {
    CDialog::DoDataExchange(pDX);
    //AFX_DATA_MAP(CRemoteWatchDlg)
    DDX_Control(pDX, IDC_LIST, m_list);
    DDX_Control(pDX, IDC_CONNECT, m_connect);
    //AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CRemoteWatchDlg, CDialog)
    //{(AFX_MSG_MAP(CRemoteWatchDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_SERVER, OnServer)
    ON_BN_CLICKED(IDC_CLIENT, OnClient)
    ON_BN_CLICKED(IDC_DISCONNECT, OnDisconnect)
    ON_WM_MOUSEWHEEL()
    ON_BN_CLICKED(IDC_CODEC_CONFIGURE, OnCodecConfigure)
    ON_BN_CLICKED(IDC_CONNECT, OnConnect)
    ON_WM_DESTROY()
    //})AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_MESSAGE_MAP(CRemoteWatchDlg, CDialog)
    // CRemoteWatchDlg message handlers
    BOOL CRemoteWatchDlg::OnInitDialog()
    {
        m_hicon=TRUE); // Set big icon
        m_hicon=FALSE); // Set small icon
        if(GetRS()->LoadString(IDS.CONNECT_MODE)="Server")
            CMString bttxf(IDS_SERVER_CONNECT_BUTTON);
            m_connectb.SetWindowText(bttxf);
        else if(GetRS()->LoadString(IDS.CONNECT_MODE)="Client")
            CMString bttxf(IDS_CLIENT_CONNECT_BUTTON);
            CString bttx;
            bttx.Format(bttxf,GetRS()->LoadString(IDS_SERVER_IP));
            m_connectb.SetWindowText(bttx);
        UpdateStatus();
        m_idle_sleep=GetRS()->LoadDWORD(IDS_SERVER_IDLE_TIME);
if((GetRS()->LoadString(IDS_CONNECT_MODE)=="Server") && GetRS()->LoadWORD(IDS_SERVER_AUTOSTART))
    PostMessage(WM_COMMAND,IDC_CONNECT);
return TRUE;
}

void CRemoteWatchDlg::EnableSysMenuItem(int id, int flag)
{
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    pSysMenu->EnableMenuIten(id, MF_BYCOMMAND((flag?MF_ENABLED:MF_GRAYED));
}

void CRemoteWatchDlg::PostNcDestroy()
{
    OutputDebugString("PostNcDestroy(): Program Exit");
    delete this;
}

void CRemoteWatchDlg::OnSysComaand(UINT nID, LPARAM lParam)
{
    switch(nID&0xFFFF)
    {
    case IDM_ABOUTBOX:
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        break;
    case IDM_VIDEOSELECT:
        {
            OnVideoInput();
        }
        break;
    case IDM_CODEC_CONFIG:
        {
            OnCodecConfigure();
        }
        break;
    case IDM_DISCONNECT:
        {
            OnDisconnect();
        }
        break;
    default:
        CDialo::OnSysComaand(nID, lParam);
        break;
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.
void CRemoteWatchDlg::OnPaint()
{
    if(IsIconic())
    {
        CPaintDC dc(this); // device context for painting
        SendMessage(WM_ICONERASEBGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1)/2;
int y = (rect.Height() - cylcon + 1) / 2;

// Draw the icon
dc.DrawIcon(x, y, m_hIcon);
else
{
    CDialog::OnPaint();
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CRemoteWatchDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CRemoteWatchDlg::OnConnect()
{
    CString txt;
    m_connectb.GetWindowText(txt);
    CMString btxt(IDS.SERVER.CONNECT.BUTTON);
    if (txt == btxt)
        OnServer();
    else
        OnClient();
}

void CRemoteWatchDlg::OnServer()
{
    OutputDebugString("Server Startup");
    CConnect cConDlg(GetRS()->LoadDWORD(IDS.SERVER_PORT), this);
    if (IDOK == cConDlg.DoModal())
    {
        CWaitCursor wc;
        if (m_ObjSocket.Connect(&cConDlg.m_ListenSock))
        {
            m_type = T_SERVER;
            UpdateStatus();
            return;
        }
        CMString sMsg(IDS_SERVER_CONNECTIONFAILED);
        AfxMessageBox(sMsg);
    }
    m_type = T_NONE;
}

void CRemoteWatchDlg::OnClient()
{
    OutputDebugString("Client Startup");
    CMString sMsg(IDS_CLIENT_SERVERNOTFOUND);
    do
    {
        m_type = T_NONE;
        CWaitCursor wc;
        CClientConnect cConDlg(m_ObjSocket,
            GetRS()->LoadString(IDS_SERVER_IP),
            GetRS()->LoadDWORD(IDS_SERVER_PORT), this);
        if (IDOK == cConDlg.DoModal())
        {
            m_type = T_CLIENT;
            UpdateStatus();

#define X_OFFSET 10
        SIZE s[] = { /* define the position within the screen */
            0, 0, /* (0,0), */
        }
40
{1,0},
(2,0),
(0,1),
(1,1),
(2,1),
(0,2),
(1,2),
(2,2),
};
int w,h;
CRect wr;
// tile all the camera windows
for(int i=0;i<MAX_CAM;i++)
{
    m_PlayWnd[i]=new PlayWnd(this,
        GetRSO->LoadDWORD(IDS.FRAME_WIDTH),
        GetRSO->LoadDWORD(IDS.FRAME_HEIGHT),
        GetRSO->LoadDWORD(IDS.FRAME_SIZE_IDX));

    m_PlayWnd[i]->GetWindowRect(&wr);
w=wr.Width();
h=wr.Height();
m_PlayWnd[i]->MoveWindow(X_OFFSET+s[i].cx,w,s[i].cy*h,w,h);
}
// finally move our main window
GetWindowRect(&wr);
MoveWindow(X_OFFSET+s[i].cx,w,s[i].cy*h,wr.Width(),wr.Height());
// send start message to the server
CStartMessage m(GetRSO->LoadDWORD(IDS_VIDEO_FRAME_RATE),
    GetRSO->LoadDWORD(IDS.FRAME_WIDTH),
    GetRSO->LoadDWORD(IDS.FRAME_HEIGHT),
    GetRSO->LoadDWORD(IDS.CODEC_HANDLER));
m_ObjSocket.SendObject(&m);
return;
}
while(AfxMessageBox(sMsg,MB_YESNO|MB_ICONQUESTION)==IDYES);

void CRemoteWatchDlg::SetFocusWindow(PlayWnd *w,BOOL flag)
{
    for(int i=0;i<MAX_CAM;i++)
    {
        if(v==m_PlayWnd[i])
            m_PlayWnd[i]->SetFrameFocus(FALSE);
        else
        {
            if(flag)
                m_PlayWnd[i]->SetFrameFocus(TRUE);
            // xmit command to focus on this (i-th) camera
    CFocusMessage m(1<i);
    m_ObjSocket.SendObject(&m);
        }
        else
        {
            m_PlayWnd[i]->SetFrameFocus(FALSE);
            // xmit command to start multiplexing
        CFocusMessage m(-1);
    m_ObjSocket.SendObject(&m);
        }
    }
}
void CRemoteWatchDlg::OnDisconnect()
void CRemoteWatchDlg::UpdateStatus()
{
    CString sMsg;
    if (!_ObjSocket.IsConnected())
    {
        CMString sFormat(IDS_NOTCONNECTED);
        sMsg.Format(sFormat, AfxGetAppName());

        CMString slMsg(IDS_STATUS_NOTCONNECTED);
        _list.AddString(slMsg);
        m_connectb.EnableWindow(TRUE);
        EnableSysMenuItem(IDM_VIDEOSELECT, FALSE);
        EnableSysMenuItem(IDM_CODEC_CONFIG, FALSE);
        EnableSysMenuItem(IDM_DISCONNECT, FALSE);

        // kill all the display windows and capture window
        for (int i = 0; i < MAX_CAM; i++)
        {
            if (IS_VALID_PTR(m.PlayWnd[i]))
                m.PlayWnd[i]->DestroyWindow();
            INIT_PTR(m.PlayWnd[i]);
        }

        if (IS_VALID_PTR(m.CaptWnd))
            m.CaptWnd->DestroyWindow();
        INIT_PTR(m.CaptWnd);

        // set priority level to normal
        AfxGetThread()->SetThreadPriority(THREAD_PRIORITY_NORMAL);

        // prepare for next connection
        if (_type == T_SERVER && GetRSO()->LoadWORD(IDS_SERVER_RECONNECT))
            PostMessage(WM_COMMAND, IDC_SERVER);
    }
    else
    {
        if (_type == T_SERVER)
        {
            CMString sFormat(IDS_CONNECTED_TO_CLIENT);
            sMsg.Format(sFormat, AfxGetAppName());

            CMString sFormatMsg(IDS_CONNECTED_TO_CLIENT);
            CString peern;
            m_ObjSocket.GetPeerName(peern);
            CString slMsg;
            // format the message
            slMsg.Format(sFormatMsg, peern);
            _list.AddString(slMsg);

            EnableSysMenuItem(IDM_VIDEOSELECT, TRUE);
            EnableSysMenuItem(IDM_CODEC_CONFIG, TRUE);
            // Set server's thread priority level
            AfxGetThread()->SetThreadPriority(
                GetRSO()->LoadWORD(IDS_SERVER_THREAD_PRIORITY_LEVEL));
        }
        else
            // OutputDebugString("OnDisconnect() : exit");
    }
}
void CRemoteWatchDlg::OnIdle()
{
    if(!m_CaptWnd)    // if capture window does not exist => exit...
        return;
    if(!m_Cev.Lock(m_idle_sleep))    // lock for some time...
        return;
    OutputDebugString("OnIdle(): Sending Frame...");
    char *buf=new char [MAXFRAME_SIZE];
    if(IS_VALID_PTR(buf))
    {
        m_CaptWnd->GetInFrame(buf.cam_no);
        CFrameMessage f((unsigned char *)buf.cam_no);
        m_ObjSocket.SendObject(&f);
        DELETE_PTR(buf);
    }
}

BOOL CRemoteWatchDlg::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
    if(IS_VALID_PTR(m_PlayWnd))
        m_PlayWnd->SetFocus();
    return CDialog::OnMouseWheel(nFlags, zDelta, pt);
}

void CRemoteWatchDlg::OnCodecConfigure()
{
    if(!m_ObjSocket.m_codec.ConfigureCurCodec())
        return;
    CMSString chf(IDS_CODECHANDLE_FMT);
    CMSString csf(IDS_CODECSETTING_FMT);
    DWORD handler=m_ObjSocket.m_codec.GetCurCodecHandler();
    CString hndlr("",5);  memset((void)*(LPCTSTR)hndlr.thandler,4);
    hndlr.SetAt(4,0);
    hndlr.MakeLower();
    // find the settings for this codec handler
    for(int i=1;i<256;i++)
    {
        CString ch;
        ch.Format(chf,i);
        DWORD ch=GetAS()>(LPCTSTR)ch);
        if(ch==0)
            break;
        CString chndlr("",5);  memset((void*)((LPCTSTR)chndlr.&ch,4));
        chndlr.SetAt(4,0);
        chndlr.MakeLower();
        // compare the handlers...
        if(chndlr!=hndlr)
            continue;
        CString cs;
        cs.Format(csf,i);
        // store the codec config to registry...
GetRS()->StoreBinary(cS,
(LPBYTE)m_ObjSocket.m_codec.GetCurCodecCfg(),
 m_ObjSocket.m_codec.GetCurCodecCfgSize());

break;
}

// send to remote server...
OutputDebugString("OnCodecConfigure(): Sending Codec Cfg...");
CCodecCfgMessage ccfg(m_ObjSocket.m_codec.GetCurCodecCfg(),
 m_ObjSocket.m_codec.GetCurCodecCfgSize());
m_ObjSocket.SendObject(&ccfg);

void CRemoteWatchDlg::OnVideoInput()
{
    if(IS_VALID_PTR(m_CaptWnd))
        m_CaptWnd->VideoSource();
}

void CRemoteWatchDlg::OnCancel()
{
    OnDisconnect();
    DestroyWindow();
}

void CRemoteWatchDlg::OnOK()
{
    OnDisconnect();
    DestroyWindow();
}
CClientConnect definition file

 ifndef _CLIENTCONNECT_H_
define _CLIENTCONNECT_H_

 if _MSC_VER > 1000
 pragma once
 endif // _MSC_VER > 1000
 // ClientConnect.h : header file
 //
 #include "Csockobj.h"

 // CClientConnect dialog
class CClientConnect : public CDialog {
  CObjectSocket *m_ObjSocket;
  CString m_sAddr;
  int m_sPort;
  int m_codecidx;

  // Construction
  public:
  CClientConnect(CObjectSocket *sock, LPCTSTR sAddr,
      int sPort, CWnd* pParent = NULL); // standard constructor

  // Dialog Data
  #ifdef AFX_DATA(CClientConnect)
    enum { IDD = IDD_CLIENT_CONNECT };
    CStatic m_status;
  #endif //AFX_DATA

  // Overrides
  // ClassWizard generated virtual function overrides
  #ifdefAFX_VIRTUAL(CClientConnect)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
  #endif //AFX_VIRTUAL

  // Implementation
  protected:

    // Generated message map functions
    #ifdefAFX_MSG(CClientConnect)
    virtual BOOL OnInitDialog();
    #endif //AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#endif

CClientConnect implementation file

 #include "stdafx.h"
 #include "RemoteWatch.h"
 #include "ClientConnect.h"

 #ifdef _DEBUG
 #define new DEBUG_NEW
 #undef THIS_FILE
 static char THIS_FILE[] = __FILE__;
 #endif

 // CClientConnect dialog

 Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CClientConnect::CClientConnect(CObjectSocket *sock, LPCTSTR sAddr,
    int sPort, CWnd* pParent /*=NULL*/)
    : CDialog(CClientConnect::IDD, pParent)
{
    _ObjSocket=sock;
    _sAddr=sAddr;
    _sPort=sPort;
    //{{AFX_DATA_INIT(CClientConnect)
    //    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void CClientConnect::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CClientConnect)
    DDX_Control(pDX, IDC.STATUS, m.status);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CClientConnect, CDialog)
    //{{AFX_MSG_MAP(CClientConnect)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CClientConnect message handlers
BOOL CClientConnect::OnInitDialog()
{
    CDialog::OnInitDialog();
    CMString fmt(IDS_CLIENT_CONNECT);
    CString txt;
    txt.Format(fmt , _sAddr);
    m.status.SetWindowText(txt);
    CenterWindow();
    ShowWindow(SW_SHOW);
    RedrawWindow();
    if (_ObjSocket->Connect(_sAddr, _sPort))
    OnOK();
    else
    OnCancel();
    return TRUE;
}
CConnect definition file

#include "resource.h"

// CListenSocket
class CConnect;
class CListenSocket: public CSocket
{
  CConnect * m_pConnect;
public:
  CListenSocket();
  ~CListenSocket();
  bool DidCreate(CConnect * pCon, int nSocket);
  void OnAccept(int nErrorCode);
};

CConnect dialog

class CConnect: public CDialog
{
  CString m_smsg;
  int m_nSocket;
  int m_timeout;
public:
  CListenSocket m_ListenSock;
  CConnect(int nSocket, CWnd* pParent = NULL); // standard constructor
  void GoodAccept();
  void BadAccept();
private:
  DECLARE_MESSAGE_MAP()
};

Connect implementation file

#include "stdafx.h"
#include "resource.h"
#include "connect.h"
#include "RemoteWatch.h"

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
```cpp
#define _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////////////////////////////////////////////////////
// CListenSocket
CListenSocket::CListenSocket()
    : CSocket()
{
}

CListenSocket::~CListenSocket()
{
    Close();
}

BOOL CListenSocket::DidCreate(CConnect *pCon, int nSocket)
{
    ASSERT(pCon);
    m_pConnect = pCon;
    pCon->UpdateWindow();
    CMString sMsg(IDS_SERVER_PREPARING);
    pCon->m_status.SetWindowText(sMsg);
    if (Create(nSocket) && Listen())
    {
        CString sAddr;
        CMString sFormat(IDS_SERVER_WAITINGFORMAT);
        ((CRemoteWatchApp*)AfxGetApp())->GetMyIPAddress(sAddr);
        sMsg.Format(sFormat, sAddr);
        pCon->m_status.SetWindowText(sMsg);
        return TRUE;
    }
    sMsg.LoadString(IDS_SERVER_LISTENSOCKFAIL);
    AfxMessageBox(sMsg);
    return FALSE;
}

void CListenSocket::OnAccept(int nErrorCode)
{
    if (!nErrorCode)
        m_pConnect->GoodAccept();
    else
        m_pConnect->BadAccept();
}

///////////////////////////////////////////////////////////////////////////////
// CConnect dialog
void CConnect::GoodAccept()
{
    OnOK();
}

void CConnect::BadAccept()
{
    CMString sMsg(IDS_SERVER_BADACCEPT);
    AfxMessageBox(sMsg);
    OnCancel();
}

CConnect::CConnect(int nSocket, CWnd * pParent /*=NULL*/)
    : CDialog(CConnect::IDD, pParent)
```
void CConnect::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CConnect)
    // NOTE: the ClassWizard will add DDx and DDX calls here
    DDX_Control(pDX, IDC_STATUS, m_status);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CConnect, CDialog)
    //{{AFX_MSG_MAP(CConnect)
    ON_WM_TIMER()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CConnect::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetWindowText(AfxGetAppName());
    // load the timeout value from the registry...
    m_timeout=GetRS()->LoadDWORD(IDS.SERVER.TIMEOUT.VALUE);
    CenterWindow();
    ShowWindow(SW_SHOWNORMAL);
    RedrawWindow();
    if(!m_ListenSock.Create(this, m_nSocket))
        OnCancel();
    else
    {
        m_status.GetWindowText(m_msys);
        SetTimer(1,1000,NULL);
        return TRUE;
    }
}

void CConnect::OnTimer(UINT nIDEvent)
{
    if(m_timeout<=0)
    {
        m_timeout=GetRS()->LoadDWORD(IDS.SERVER.TIMEOUT.VALUE);
        m_ListenSock.Close();
    }
}
VersionInfo vi;
CMString sMsg(IDS_SERVER_RECONNECTING);
m_status.SetWindowText(sMsg);
if(!m_ListenSock.DidCreate(this.m_nSocket))
  OnCancel();
else
{
  CString msg;
  msg.Format("Error \[\d\] ", m_smsg, m_timeout);
  m_status.SetWindowText(msg);
  m_timeout--;
}
VersionInfo definition file

#ifndef _VERSIONINFO_H_
define _VERSIONINFO_H_
#endif

#ifdef _MSC_VER > 1000
#pragma once
#endif

class VersionInfo
{
public:
    VersionInfo();
    VersionInfo(LPCTSTR imageName);
    VersionInfo(LPCSTR imageName);
    VersionInfo();

    CString ProductName() const;
    CString ProductVersion() const;
    CString CompanyName() const;
    CString Copyright() const;
    CString Comments() const;
    CString FileDescription() const;
    CString FileVersion() const;
    CString InternalName() const;
    CString LegalTrademarks() const;
    CString PrivateBuild() const;
    CString SpecialBuild() const;

private:
    void InitVer(LPCTSTR imageName);
    CString GetValue(LPCTSTR key) const;

    LPVOID m_ResourceData;
    CString m_LangID;
};
#endif

VersionInfo implementation file

#include "stdafx.h"
#include "RemoteWatch.h"
#include "VersionInfo.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#endif

refund

.isHidden

VersionInfo::VersionInfo()
{
    // m_pszExeName is the real image name in MFC
    CString name = AfxGetApp()->m_pszExeName;
    name += _T(".EXE");
    InitVer(name);
}

VersionInfo::VersionInfo(LPCTSTR imageName)
{
    InitVer(imageName);
}
VersionInfo::VersionInfo()
{
    delete [] m_ResourceData;
}

// Attributes
CString VersionInfo::ProductName() const
{
    return GetValue(_T("ProductName"));
}

CString VersionInfo::ProductVersion() const
{
    return GetValue(_T("ProductVersion"));
}

CString VersionInfo::CompanyName() const
{
    return GetValue(_T("CompanyName"));
}

CString VersionInfo::Copyright() const
{
    return GetValue(_T("LegalCopyright"));
}

CString VersionInfo::Comments() const
{
    return GetValue(_T("Comments"));
}

CString VersionInfo::FileDescription() const
{
    return GetValue(_T("FileDescription"));
}

CString VersionInfo::FileVersion() const
{
    return GetValue(_T("FileVersion"));
}

CString VersionInfo::InternalName() const
{
    return GetValue(_T("InternalName"));
}

CString VersionInfo::LegalTrademarks() const
{
    return GetValue(_T("LegalTrademarks"));
}

CString VersionInfo::PrivateBuild() const
{
    return GetValue(_T("PrivateBuild"));
}

CString VersionInfo::SpecialBuild() const
{
    return GetValue(_T("SpecialBuild"));
}

// Construction helper
void VersionInfo::InitVer(LPCTSTR imageName)
{
    DWORD resourceSize, zeroHandle = 0;
}
LPTSTR image = const_cast<LPTSTR>(imageName); // WinAPI wants non-const arguments

m_ResourceData = NULL;
resourceSize = ::GetFileVersionInfoSize(image, &zeroHandle);
if (zeroHandle == 0) {
    m_ResourceData = new char [resourceSize];
    if (::GetFileVersionInfo(image, NULL, resourceSize, m_ResourceData)) {
        // Pick up the language ID used by this version resource.
        // Does not work with multi-language resources.
        UINT dataSize;
        LPVOID valuePointer;
        if (::VerQueryValue(m_ResourceData, _T("\VarFileInfo\Translation"),
                             &valuePointer, &dataSize)) {
            // Must know the size of the scalar type used.
            // I bet there are more portable ways of doing this...
            _int16* lang = reinterpret_cast<_int16*>(valuePointer);
            m_LangID.Format("%04x%04x", lang[0], lang[1]);
        }
    }
}

// Attribute helper
CString VersionInfo::GetValue(LPCTSTR key) const
{
    CString str;
    LPVOID valuePointer;
    UINT dataSize;
    ASSERT(m_ResourceData != NULL);

    str.Format("\StringFileInfo\%s\%s", m_LangID, key);
    // We need a non-const pointer to the string data.
    ::VerQueryValue(m_ResourceData, str.GetBuffer(0), &valuePointer, &dataSize);
    str.ReleaseBuffer();
    str.Format("%s", reinterpret_cast<LPTSTR>(valuePointer));
    return str;
}
StdAfx definition file

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently

#if !defined(AFX_STDAFX_H__820CD4CB_5C13_11D4_8040_0090272BCF9E__INCLUDED_)
#define AFX_STDAFX_H__820CD4CB_5C13_11D4_8040_0090272BCF9E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN // Exclude rarely-used stuff from Windows headers

#include <afxwin.h> // MFC core and standard components
#include <afxext.h> // MFC extensions
#include <afxdtctl.h> // MFC support for Internet Explorer 4 Common Controls
#if !defined(_AFX_NO_AFXCMN_SUPPORT)
#include <afxcmn.h> // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h> // MFC socket extensions
#include <afxinet.h>
#include <afxtr.h>
#include <assert.h>

#define IS_VALID_PTR(a) (((a)!=(void*)0xc0c0c0c0c0c0c0c0c0c0c0c0))
#define INIT_PTR(a) (a=NULL)
#define DELETE_PTR(a) {if(IS_VALID_PTR(a)) delete a; INIT_PTR(a);};

class CString: public CString
{
public:
    CString(int id): CString() { LoadString(id); }
};

#ifdef OutputDebugString
#undef OutputDebugString
define OutputDebugString
#endif
#define OutputDebugString

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // 'defined(AFX_STDAFX_H__820CD4CB_5C13_11D4_8040_0090272BCF9E__INCLUDED_)

StdAfx implementation file

// stdafx.cpp : source file that includes just the standard includes
// RemoteWatch.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
Resource definition file

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by RemoteWatch.rc
//
#define IDM_ABOUTBOX 0x0010
#define IDM_VIDEoselect 0x0020
#define IDM_CODECCONFIG 0x0030
#define IDM_DISCONNECT 0x0050
#define IDD_ABOUTBOX 100
#define IDD_ABOUTBOX 101
#define IDD_REMOTEWATCH_DIALOG 102
#define IDD_SERVER_WAITINGFORMAT 102
#define IDD_SERVER_PREPARING 103
#define IDD_SERVER_LISTENSOCKFAIL 104
#define IDD_SERVER_BDACCEP 105
#define IDD_OSEAR_MEMORY 106
#define IDD_OSEAR_OBJECT 107
#define IDD_OSEAR_FILE 108
#define IDD_OSEAR_SOCKET 109
#define IDD_OSEAR_DROP 110
#define IDD_SERVER_CONNECTIONFAILED 111
#define IDD_CLIENT_SERVERNOTFOUND 112
#define IDD_NOTCONNECTED 113
#define IDD_CONNECTEDTODCLIENT 114
#define IDD_CONNECTEDTOSERVER 115
#define IDD_SERVER_NOSERIALPORT 116
#define IDD_APP_REG_KEY 117
#define IDD_SERVER_NOCODEC 118
#define IDD_IP_URL 119
#define IDD_SERVER_RECONNECTING 120
#define IDD_IP_FILE 121
#define IDD_CODEC_SETTING 122
#define IDD_SERVER_TIMEOUT_VALUE 123
#define IDD_SERVER_COM_PORT 124
#define IDD_CODEC_HANDLER 125
#define IDD_SERVER_IP 126
#define IDD_SERVER_PORT 127
#define IDR_MAINFRAME 128
#define IDS_VIDEO_FRAME_RATE 128
#define IDR_CONNECT 129
#define IDS_SERVER_IDLE_TIME 129
#define IDR_VIDEOSELECT 130
#define IDR_SETUP 131
#define IDR_CODECCONFIG 131
#define IDR_PAGE1 132
#define IDS_SERVER_THREAD_PRIORITY_LEVEL 132
#define IDR_SERVER_ACCESS 133
#define IDR_SERVER_RECONNECT 133
#define IDR_SERVER_VIDEO 134
#define IDR_DISCONNECT 134
#define IDS_SOCKET_INIT_FAILED 135
#define IDS_FATAL_SOCKET_INIT_FAILED 135
#define IDS_FRAME_WIDTH 136
#define IDR_CLIENT_CONNECT 136
#define IDR_FRAME_HEIGHT 137
#define IDR_CURSOR_LEFT 137
#define IDR_CODEC_FILE 138
#define IDR_CURSOR_MIDDLE 138
#define IDR_CODEC_MODE 139
#define IDR_CURSOR_RIGHT 139
#define IDS_FATAL_RAS_OPEN 140
#define IDR_CURSOR_TOP 140
#define IDR_CODEC_CONFIG_MENU 141
#define IDR_CODEC_SW 141
#define IDR_CLIENT_CONNECT 142
#define IDC_CURSOR_W 142
#define IDS_SERVER_AUTOSTART 143
#define IDC_CURSOR_NW 143
#define IDS_CODEC_HANDLER_FMT 144
#define IDC_CURSOR_S 144
#define IDS_CODEC_FILE_FMT 145
#define IDC_CURSOR_MOVE 145
#define IDS_CODEC_SETTING_FMT 146
#define IDS_CAMERA_SEQUENCE 147
#define IDS_CAMERA_NAME_FMT 148
#define IDS_CLIENT_CONNECT_BUTTON 149
#define IDS_SERVER_CONNECT_BUTTON 150
#define IDS_STATUS_NOTCONNECTED 151
#define IDS_STATUS_CONNECTED_TO_SERVER 152
#define IDS_CONNECTED_TO_CLIENT 153
#define IDS_FRAME_SIZE_IDX 154
#define IDC_SERVER 1000
#define IDC_CLIENT 1001
#define IDC_DISCONNECT 1002
#define IDC_DISPLAY 1003
#define IDC_STATUS 1004
#define IDC_IPADDRESS 1005
#define IDC_CAMUP 1006
#define IDC_CAMDOWN 1007
#define IDC_CAMRIGHT 1008
#define IDC_CAMLEFT 1009
#define IDC_CAMSTOP 1010
#define IDC_EDIT1 1011
#define IDC_CAMERA 1012
#define IDC_EDIT4 1012
#define IDC_EDIT2 1013
#define IDC_OPTIONS 1013
#define IDC_SERVER_LIST 1013
#define IDC_PANEDITOR 1013
#define IDC_CAMERA_ACTION 1014
#define IDC_CLIENT_LIST 1014
#define IDC_EDIT3 1015
#define IDC_APPLY 1015
#define IDC_TILTEDITOR 1015
#define IDC_CAMERATILT 1016
#define IDC_TITLE 1017
#define IDC_CODEC_CONFIG 1017
#define IDC_FRAME 1018
#define IDC_SERVER_FRAME 1018
#define IDC_VIDEO_INPUT 1018
#define IDC_LIST1 1019
#define IDC_CLIENT_FRAME 1019
#define IDC_LIST 1019
#define IDC_COMBO1 1020
#define IDC_CHECK1 1021
#define IDC_CHECK2 1022
#define IDC_SERVERS_TITLE 1023
#define IDC_CLIENTS_TITLE 1024
#define IDC_IPADDRESS1 1025
#define IDC_PROGRAMINFO 1026
#define IDC_CONNECT 1028
#define IDC_PET_STATIC 1029
#define IDC_TILT_STATIC 1030

// Next default values for new objects

//
#ifndef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 146
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1031
#endif
#endif

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
#define _APS_NEXT_SYMED_VALUE
#endif
#endif
Network

CSocketobj definition file

```c
#define _CSOCKOBJ_H
#define _CSOCKOBJ_H

#include "SCodec.h"
#include "serio.h"
#define SHOW_ERRORS FALSE

// Some Forward Declarations
class CObjectSocket;
class CNetObject;

class CSocketArchive : public CArchive{
public:
    CSocketArchive(CFile *pFile, UINT nMode, int nBufSize=4096):
        CArchive(pFile, nMode, nBufSize) {}
    void ClearIndex();
};

class CSocketSlave : public CSocket{
    CObjectSocket *m_pMaster;
public:
    CSocketSlave(CObjectSocket *pOS);
    void OnReceive(int nErrorCode);
    void OnClose(int nErrorCode);
};

class CObjectSocket{
protected:
    friend class CSocketSlave;
    CSocketSlave *m_pSocket;
    CSocketFile *m_pSocketFile;
    CSocketArchive *m_pArchiveIn,
        * m_pArchiveOut;
    void ClearNewPointers();
    void ReadPendingO;
public:
    BOOL m_connected;
    serio_t m_serio;
    CCodec m_codec;

   ///////////////////////////////////////////
    CObjectSocket();
    ~CObjectSocket();
    BOOL IsConnected();
    BOOL Connect(LPCTSTR lpszHostName, UINT nHostPort);
    BOOL Connect(CSocket *pSocket);
    void GetPeerName(CString &pname);
    void Disconnect();
    BOOL SendObject(CNetObject *pObj);

    virtual void OnForcedDrop();
    virtual void ConnectionClosed();
    virtual void ErrorMemory();
    virtual void ErrorArchive(int nErrorCode);
    virtual void ErrorFile(int nErrorCode);
    virtual void ErrorSocket(int nErrorCode);
};
class CNetObject : public CObject{
```
public:
  DECLARE_SERIAL(CKetObject)
  CKetObject() { };
  virtual void DoAction(CObjectSocket * pOS);
  void Serialize(CArchive & ar);
};
#endif

CSockobj implementation file

#include "stdafx.h"
#include "resource.h"
#include "csockobj.h"
#include "RemoteWatch.h"

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CSocketArchive
void CSocketArchive::ClearIndex()
{
  if(IsStoring())
  {
    OutputDebugString("ClearIndex() : storing");
    if(m_nMapCount > 1)
    {
      m_pStoreMap->RemoveAll();
      m_pStoreMap->SetAt(NULL, (void*)(DWORD)0);
      m_nMapCount = 1;
    }
  }
  else
  {
    OutputDebugString("ClearIndex() : loading");
    while(m_nMapCount > 1)
    {
      m_nMapCount--;
      m_pLoadArray->RemoveAt(m_nMapCount);
    }
  }
}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CSocketSlave
void CSocketSlave::OnReceive(int nErrorCode)
{
  ASSERT(m_pMaster);
  if(nErrorCode == 0)
  {
    DWORD dwBytes;
    IDctt(FIGREAD, &dwBytes);
    //Don't read if nothing is there
    if (dwBytes > 0)
    {
      OutputDebugString("OnReceive() >> ReadPending()");
      m_pMaster->ReadPending();
      return;
    }
    m_pMaster->ErrorSocket(nErrorCode);
  }
}

CSocketSlave::CSocketSlave(CObjectSocket * pOS)
{
  ASSERT(pOS);
  m_pMaster = pOS;
}
void CSocketSlave::OnClose(int nErrorCode)
{
    OutputDebugString("OnClose()\n");
    ASSERT(m_pMaster);
    m_pMaster->ConnectionClosed();
}

// CSocketObjects
CObjectSocket::CObjectSocket()
{
    m_connected=FALSE;
    //init socket pointers...
    INIT_PTR(m_pSocket);
    INIT_PTR(m_pSocketFile);
    INIT_PTR(m_pArchiveIn);
    INIT_PTR(m_pArchiveOut);
    m_serial_port=INVALID_HANDLE_VALUE;
}

CObjectSocket::CObjectSocket()  
{
    Disconnect();
}

BOOL CObjectSocket::Connect(LPCTSTR lpszHostName, UINT nHostPort)
{
    ASSERT(lpszHostName);
    try
    {
        m_pSocket = new CSocketSlave(this);
        if(!m_pSocket->Create())
        {
            DELETE_PTR(m_pSocket);
            return FALSE;
        }
        if(!m_pSocket->Connect(lpszHostName, nHostPort))
        {
            DELETE_PTR(m_pSocket);
            return FALSE;
        }
        m_pSocketFile = new CSocketFile(m_pSocket);
        m_pArchiveIn = new CSocketArchive(m_pSocketFile,CArchive::load);
        m_pArchiveOut = new CSocketArchive(m_pSocketFile,CArchive::store);
        int cidx=m_codec.GetCodecIndex(GetRS()->LoadDWORD(IDS_CODEC_HANDLER),
                                        GetRS()->LoadDWORD(IDS_CODEC_WIDTH),
                                        GetRS()->LoadDWORD(IDS_CODEC_HEIGHT));
        if(cidx<0)
        {
            CMSString sMsg(IDS_SERVER_NOCODEC);
            AfxMessageBox(sMsg);
        }
        if(m_codec.InitDecompressor(cidx, 
                                    GetRS()->LoadDWORD(IDS_CODEC_WIDTH),
                                    GetRS()->LoadDWORD(IDS_CODEC_HEIGHT))
            m_codec.Begin();
        else
        {
            CMSString sMsg(IDS_SERVER_NOCODEC);
            AfxMessageBox(sMsg);
        }
        m_connected=TRUE;
        OutputDebugString("Client connected\n");
    }
}
return TRUE;
} catch (CMemoryException * me) {
    me->Delete();
    ClearNewPointers();
    ErrorMemory();
    return FALSE;
}
} catch (CArchiveException * ae) {
    ae->Delete();
    ClearNewPointers();
    return FALSE;
} catch (CFileException * fe) {
    fe->Delete();
    ClearNewPointers();
    return FALSE;
}
}

void CObjectSocket::ClearNewPointers()
{
    OutputDebugString("ClearNewPointers()");
    DELETE_PTR(m_pSocket);
    DELETE_PTR(m_pSocketFile);
    DELETE_PTR(m_pArchiveIn);
    DELETE_PTR(m_pArchiveOut);
    m_codec.End();
    m_codec.Close();
    if(m_serio.port!=INVALID_HANDLE_VALUE)
    {
        serio_close(&m_serio);
        m_serio.port=INVALID_HANDLE_VALUE;
    }
}

BOOL CObjectSocket::Connect(CSocket *pSocket)
{
    ASSERT(pSocket);
    try
    {
        m_pSocket = new CSocketSlave(this);
        if(!pSocket->Accept(m_pSocket))
        {
            DELETE_PTR(m_pSocket);
            return FALSE;
        }
        m_pSocketFile = new CSocketFile(m_pSocket);
        m_pArchiveIn = new CSocketArchive(m_pSocketFile,CArchive::load);
        m_pArchiveOut = new CSocketArchive(m_pSocketFile,CArchive::store);
        CString com.port=GetRS()->LoadString(IDS_SERVER.COM.PORT); 
        if(serio_open(&m_serio, CBR_2400, (char*)(LPCTSTR)com.port) != serio.RES_OK)
        {
            CMString sMsg(IDS_SERVER.NOSERIALPORT);
            AfxMessageBox(sMsg);
        }
        OutputDebugString("Server connected");
        m_connected=TRUE;
        return TRUE;
    }
}
catch (CMemoryException * me) {
    me->Delete();
    ClearNewPointers();
    ErrorMemory();
    return FALSE;
}
catch (CArchiveException * ae) {
    ae->Delete();
    ClearNewPointers();
    return FALSE;
}
catch (CFileException * fe) {
    fe->Delete();
    ClearNewPointers();
    return FALSE;
}

void CObjectSocket::GetPeerName(CString &name) {
    UINT peerp = 0;
    m_pSocket->GetPeerName(name, peerp);
}

BOOL CObjectSocket::IsConnected() {
    return (m_connected &&
        IS_VALID_PTR(m_pSocket) &&
        IS_VALID_PTR(m_pArchiveOut) &&
        IS_VALID_PTR(m_pArchiveIn) &&
        IS_VALID_PTR(m_pSocketFile));
}

void CObjectSocket::Disconnect() {
    if (IsConnected()) {
        m_connected = FALSE;
        OutputDebugString("Disconnect() : Disconnecting...");
        m_pArchiveOut->Abort();
        DELETE_PTR(m_pArchiveOut);
        m_pArchiveIn->Abort();
        DELETE_PTR(m_pArchiveIn);
        DELETE_PTR(m_pSocketFile);
        m_pSocket->ShutDown();
        DWORD dwBytes;
        m_pSocket->IOCtl(FIONREAD, &dwBytes);
        if (dwBytes > 0) {
            BYTE Buffer[50];
            while (m_pSocket->Receive(Buffer, 50) > 0);
        }
        DELETE_PTR(m_pSocket);
        // close codec
        m_codec.End();
        m_codec.Close();
        // close serial port
        if (m_serio.port != INVALID_HANDLE_VALUE)
```cpp
{
    serio_close(m_serio);
    m_serio.port=INVALID_HANDLE_VALUE;
}

m_connected=FALSE;
OutputDebugString("Disconnect() : Done...");

void CObjectSocket::ReadPending()
{
    ASSERT(IsConnected());
    CNetObject *pObj = NULL;
    try
    {
        do
        {
            OutputDebugString("ReadPending() : Reading data");
            *m_pArchiveIn >> pObj;
            m_pArchiveIn->ClearIndex();
pObj->DoAction(this);
        }
        while(!m_pArchiveIn->IsBufferEmpty());
    }
    catch (CArchiveException * ae)
    {
        OutputDebugString("ReadPending() : CArchiveException!");
        ErrorArchive(ae->m_cause);
        ae->Delete();
    }
    catch (CFileException * fe)
    {
        OutputDebugString("ReadPending() : CFileException!");
        ErrorFile(fe->m_cause);
        fe->Delete();
    }
    catch (CMemoryException * me)
    {
        OutputDebugString("ReadPending() : CMemoryException!");
        ErrorMemory();
        me->Delete();
    }

    BOOL CObjectSocket::SendObject(CNetObject * pObj)
    {
        ASSERT(pObj);
        ASSERT(IsConnected());
        try
        {
            *m_pArchiveOut << pObj;
            m_pArchiveOut->Flush();
            m_pArchiveOut->ClearIndex();
            return TRUE;
        }
        catch (CArchiveException * ae)
        {
            OutputDebugString("SendObject() : CArchiveException!");
            ErrorArchive(ae->m_cause);
            ae->Delete();
            return FALSE;
        }
        catch (CFileException * fe)
        {
            OutputDebugString("SendObject() : CFileException!");
            ErrorFile(fe->m_cause);
        }
    }
```
fe->Delete();
return FALSE;
}

void CObjectSocket::OnForcedDrop()
{
    OutputDebugString("OnForcedDrop()");
}

void CObjectSocket::ErrorMemory()
{
    Disconnect();
    OnForcedDrop();
#if SHOW_ERRORS
    CMString sMsg(IDS_OSERR_MEMORY);
    AfxMessageBox(sMsg);
#endif
}

void CObjectSocket::ErrorArchive(int nErrorCode)
{
    Disconnect();
    OnForcedDrop();
    switch (nErrorCode)
    {
    case CArchiveException::badIndex:
    case CArchiveException::badClass:
    case CArchiveException::badSchema:
#if SHOW_ERRORS
        CMString sMsg(IDS_OSERR_OBJECT);
        AfxMessageBox(sMsg);
#endif
        break;
    default:
#if SHOW_ERRORS
        CMString sMsg(IDS_OSERR_FILE);
        AfxMessageBox(sMsg);
#endif
        break;
    }
}

void CObjectSocket::ErrorFile(int nErrorCode)
{
    Disconnect();
    OnForcedDrop();
    if (SHOW_ERRORS
        CMString sMsg(IDS_OSERR_FILE);
        AfxMessageBox(sMsg);
    }
}

void CObjectSocket::ErrorSocket(int nErrorCode)
{
    Disconnect();
    OnForcedDrop();
#if SHOW_ERRORS
    CMString sMsg(IDS_OSERR_SOCKET);
    AfxMessageBox(sMsg);
#endif
}
void CObjectSocket::ConnectionClosed()
{
    OutputDebugString("ConnectionClosed()"),
    Disconnect();
    OnForcedDrop();
    if SHOW_ERRORS
        CMString sMsg(IDS_OSERR_DROP);
        AfxMessageBox(sMsg);
    endif

    IMPLEMENT_SERIAL(CNetObject, CObject, 1)

    void CNetObject::Serialize(CArchive& ar)
    {
        CObject::Serialize(ar);
    }

    void CNetObject::DoAction(CObjectSocket * pQS)
    {
        ASSERT(FALSE);
        This member should be overridden in the
        programmer defined CNetObject
Tranobj definition file

#include "TRANOBJ.H"
#define _TRANOBJ_H_

class CStartMessage : public CNetObject
{
    struct HEADER{
        int m_fps;
        int m_width;
        int m_height;
        DWORD m_handler;
        char m_version[12];
    };

    // define PACKET union that is 256 bytes long
    // this is for future compatibility...
    union PACKET{
        HEADER h;
        char bytes[256];
    };

    PACKET m_p;
public:
    DECLARE_SERIAL(CStartMessage);
    CStartMessage() : CNetObject() {};
    CStartMessage(int fps, int width, int height, DWORD handler);
    void Serialize(CArchive & ar);
    void DoAction(CObjectSocket * pOS);
};

class CFocusMessage : public CNetObject
{
    int m_focus;
public:
    DECLARE_SERIAL(CFocusMessage);
    CFocusMessage() : CNetObject() { m_focus=1; };
    CFocusMessage(int focus);
    void Serialize(CArchive & ar);
    void DoAction(CObjectSocket * pOS);
};

class CCamInfoMessage : public CNetObject
{
    struct CAMERAS{
        char m_cam;
        char m_name[63];
    };

    // define PACKET union that is 64 bytes long
    // this is for future compatibility...
    union RPACKET{
        CAMERAS c;
        char bytes[64];
    };

    RPACKET m_p;
public:
    DECLARE_SERIAL(CCamInfoMessage);
    CCamInfoMessage() : CNetObject() {};
    CCamInfoMessage(int cam, CString name);
    void Serialize(CArchive & ar);
    void DoAction(CObjectSocket * pOS);
};
enum{   CAM_STOP =0,   CAM_UP =0x08,   CAM_DWN =0x10,   CAM_LEFT =0x04,   CAM_RIGHT =0x02,   CAM_ZIN =0x20,   CAM_ZOUT =0x40,   PACKET_LEN =7};

class CCamCtrlMessage : public CNetObject
   
   char m_Packet[PACKET_LEN],m_PacketR[PACKET_LEN]+;   public:
   DECLARE_SERIAL(CCamCtrlMessage);

   CCamCtrlMessage() : CNetObject() {}  
   CCamCtrlMessage(char *packet);  
   CCamCtrlMessage(int camera,int cmd=CAM_STOP,int pans=0,int tilts=0);  
   void CalcCheckSum();

   void Serialize(CArchive &ar);  
   void DoAction(CObjectSocket *pOS);
};

class CCCodecCfgMessage : public CNetObject
   
   char *m_cfgs,*m_cfgr;  
   int m_size;

   public:
   DECLARE_SERIAL(CCodecCfgMessage);

   CCCodecCfgMessage() : CNetObject() { INIT_PTR(m_cfgr); }  
   CCCodecCfgMessage(char *cfg,int size);  

   void Serialize(CArchive &ar);  
   void DoAction(CObjectSocket *pOS);
};

class CFrameMessage : public CNetObject
   
   char *m_frame;  
   short m_cam;  
   char m_buf[MAXFRAME_SIZE];

   public:
   DECLARE_SERIAL(CFrameMessage);

   CFrameMessage() : CNetObject() { INIT_PTR(m_frame); }  
   CFrameMessage(unsigned char *,int);  

   void Serialize(CArchive &ar);  
   void DoAction(CObjectSocket *pOS);
};

class CErrorMessage : public CNetObject
   
   CString m_error;  

   public:
   DECLARE_SERIAL(CErrorMessage);

   CErrorMessage() : CNetObject() {}  
   CErrorMessage(LPCTSTR err);  

   void Serialize(CArchive &ar);  
   void DoAction(CObjectSocket *pOS);
};
Tranobj implementation file

#include "stdafx.h"
#include "RemoteWatch.h"
#include "RemoteWatchDlg.h"
#include "cssockobj.h"
#include "tranobj.h"

IMPLEMENT_SERIAL(CStartMessage, CNetMessage, 1)

CStartMessage::CStartMessage(int fps, int width, int height, DWORD handler)
{
    // fill in the startup message that is sent to the server...
    memset(&m_p, 0, sizeof(PACKET));
    m.p.h.m_fps=fps;
    m.p.h.m.width=width;
    m.p.h.m.height=height;
    m.p.h.m_handler=handler;
    // send our version...
    memcpy(&m.p.h.m.version,GetVI()->ProductVersion(),
          (GetVI()->ProductVersion()).GetLength());
}

void CStartMessage::Serialize(CArchive &ar)
{
    CNetMessage::Serialize(ar);
    if (ar.IsStoring())
        ar.Write((void*) &m_p,sizeof(PACKET));
    else
        ar.Read((void*) &m_p,sizeof(PACKET));
}

void CStartMessage::DoAction(CObjectSocket *pQS)
{
    // get the index of the codec...
    int cidx=pQS->m_codec.GetCodecIndex(m.p.h.m_handler,
                                        m.p.h.m.width,
                                        m.p.h.m.height);
    if(cidx>0)
    {
        if(pQS->m_codec.InitCompressor(cidx,
                                        m.p.h.m.width,
                                        m.p.h.m.height))
        {
            pQS->mCodec.Begin();
            // Load Codec state from registry...
            CString chf(IDS_CODEC_HANDLER_FMT);
            CString csf(IDS_CODEC_SETTING_FMT);
            CString hndlr("",5);  mempcpy((void*)(LPCTSTR)hndlr,&m_p.h.m_handler,4);
            hndlr.SetAt(4,0);  hndlr.MakeLower();
            for(int i=1;i<256;i++)
            {
                CString ch;
                ch.Format(chf,i);
                DWORD ch=GetAS()->LoadDWORD((LPCTSTR)ch);
                if(ch==0)
                    break;
                CString chndlr("",5); mempcpy((void*)(LPCTSTR)chndlr,&ch,4);
            }
        }
    }
}
chndlr.SetAt(4,0); chndlr.MakeLower();
// compare handlers...
if(chndlr!=hndlr)
    continue;
CString cS;
cS.Format(cSf,i);
int size;
char *cState=(char*)GetRS()->LoadBinary(cS,size);
if(cState!=NULL)
{
    pOS->m_codec.SetCurCodecCfg(cState,size);
    DELETE_PTR(cState);
}
else
{
    CMSString sMsg(IDS_SERVER_NOCODEC);
    AfxMessageBox(sMsg);
}
CMSString cCS(IDS_CAMERA_SEQUENCE);
DWORD camseq=GetRS()->LoadDWORD((LPCTSTR)cCS);
if(camseq==0)
    camseq=1; // turn 'on' at least 1 camera
// create capture window...
((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->m_CaptWnd=new CaptWnd(
    ((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd,
    &((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m cámara,
    camseq,
    m.p.h.m_fps,
    m.p.h.m_width,
    m.p.h.m_height);
if(IS_VALID_PTR(((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m_CaptWnd))
    ((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m_CaptWnd->CaptureVideo();
// Send the camera info (name) to the client program
CMSString cNF(IDS_CAMERA_NAME_FMT);
for(int i=0;i<MAX_CAM;i++)
{
    if(camseq & (1<<i)) // if camera exist send its number and name
    {
        CMSString cNS; cNS.Format(cNSf,i+1);
        CMSString s; s=GetRS()->LoadString((LPCTSTR)cNS);
        CCamInfoMessage caminfo(i,s);
        pOS->SendObject(tcaminfo);
    }
}
delete this;

(reinterpret/:// // CFocusMessage
IMPLEMENT_SERIAL(CFocusMessage, CNetObject, 1)
CFocusMessage::CFocusMessage(int focus)
{
    m_focus=focus;
}
void CFocusMessage::Serialize(CArchive &ar)
{
    CNetObject::Serialize(ar);
    if (ar.IsStoring())
    {
        ar << m_focus;
    }
}
```cpp
else {
    ar >> m_focus;
}
}

void CFocusMessage::DoAction(CObjectSocket *pOS) {
    if (m_focus == -1) {
        // load default camera sequence
        CString cCS(IDS_CAMERASEQUENCE);
        m_focus = GetRS()->LoadDWORD((LPCTSTR)cCS);
        if (m_focus == 0)
            m_focus = 1; // turn 'on' at least 1 camera
    } else { // else, capture only this camera
        ((CRemoteWatchDlg*)AfxGetApp()->m.pMainWnd)->m_CaptWnd->SetCamSequence(m_focus);
        delete this;
    }
}

#pragma pack(1)
IMPLEMENT_SERIAL(CFocusMessage, CNetObject, 1)
```

```cpp
void CFocusMessage::DoAction(CObjectSocket *pOS) {
    if ((m_focus >= 0) && (m_focus < MAX_CAM)) {
        ((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m_PlayWnd[m_focus + 1] ->
        SetCameraAddress(m_focus + 1); // addressing starts from 1
        ((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m_PlayWnd[m_focus + 1] ->
        SetWindowText(m.p.c.m.name);
    } delete this;
}
```

```cpp
void CFocusMessage::DoAction(CObjectSocket *pOS) {
    if ((m.p.c.m_cam >= 0) && (m.p.c.m_cam < MAX_CAM)) {
        ((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m_PlayWnd[m.p.c.m_cam] ->
        SetCameraAddress(m.p.c.m_cam + 1); // addressing starts from 1
        ((CRemoteWatchDlg*)AfxGetApp())->m_pMainWnd->m_PlayWnd[m.p.c.m_cam] ->
        SetWindowText(m.p.c.m_name);
    } delete this;
}
```
memcpy(m_Packet.packet,PACKET_LEN);
}

CCamCtrlMessage::CCamCtrlMessage(int camera,int cmd,int pan_speed,int tilt_speed)
{
    m_Packet[0]=(char)0xff;
    m_Packet[1]=(char)camera;
    m_Packet[2]=(char)0x88;
    m_Packet[3]=(char)cmd;
    m_Packet[4]=(char)(pan_speed>255?255:pan_speed);
    m_Packet[5]=(char)(tilt_speed>255?255:tilt_speed);
    CalcCheckSum();
}

void CCamCtrlMessage::CalcCheckSum()
{
    int s=0;
    for(int i=1;i<PACKET_LEN-1;i++)
        s+=m_Packet[i];
    m_Packet[PACKET_LEN-1]=s&0xff;
}

void CCamCtrlMessage::Serialize(CArchive &ar)
{
    CNetObject::Serialize(ar);
    if (ar.IsStoring())
    {
        CalcCheckSum();
        ar.Write((void*)m_Packet,PACKET_LEN);
    }
    else
    {
        ar.Read((void*)m_PacketR,PACKET_LEN);
    }
}

void CCamCtrlMessage::DoAction(CObjectSocket *pOS)
{
    if(&pOS->m_serio.port!=INVALID_HANDLE_VALUE)
        serio.write(&pOS->m_serio,(unsigned char*)m_PacketR,PACKET_LEN);
        delete this;
};

////////////////////////////////////////////////////////////////////////
// CCodecCfgMessage
IMPLEMENT_SERIAL(CCodecCfgMessage, CNetObject, 1)

CCodecCfgMessage::CCodecCfgMessage(char *cfg,int size)
{
    m_cfgs=cfg;
    m_size=size;
}

void CCodecCfgMessage::Serialize(CArchive &ar)
{
    CNetObject::Serialize(ar);
    if (ar.IsStoring())
    {
        ar << m_size;
        ar.Write((void*)m_cfgs,m_size);
    }
    else
    {
        ar >> m_size;
        m_cfgs=new char [m_size];
void CCodecCfgMessage::DoAction(CObjectSocket *pOS)
{
    if(IS_VALID_PTR(m_cfgr))
    {
        pOS->m_codec.SetCurCodecCfg(m_cfgr.m_size);
        DELETE_PTR(m_cfgr);
    }
    delete this;
}

void CFrameMessage::DoAction(CObjectSocket *pOS)
{
    if(!pOS->IsConnected())
    {
        return;
        ar << m_cam;
        unsigned short s=(unsigned short)pOS->m_codec.GetOutputSize();
        ar << s;
        ar.Write((void*)pOS->m_codec.GetData(),s);
        CString txt;
        txt.Format("Sent %ld bytes",s);
        ((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->m_status.SetWindowText(txt);
    }
    else
    {
        unsiged short s;
        ar >> m_cam;
        ar >> s;
        ar.Read((void*)m_buf,s);
        CString txt;
        txt.Format("Rcvd %ld bytes",s);
        ((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->m_status.SetWindowText(txt);
        if(pOS->m_codec.Decompress(m_buf))
        {m_frame=pOS->m_codec.GetData();
        }
        else
        {INIT_PTR(m_frame);
        }
    }
}

void CFrameMessage::DoAction(CObjectSocket *pOS)
{
```cpp
if (IS_VALID_PTR(m_frame))
    if (n.cam>=0 && n.cam<MAX_CAM)
        {
            ((RemoteWatchDlg*)AfxGetApp()->m_pMainWnd->m_PlayWnd[n.cam]->
                SetVideoFrame((unsigned char*)m_frame);
            delete this;
        }

    // CErrorMessage
    IMPLEMENT_SERIAL(CErrorMessage, CNetObject, 1)

CErrorMessage::CErrorMessage(LPCTSTR err)
    {
        m_error=err;
    }

void CErrorMessage::Serialize(CArchive &ar)
    {
        CNetObject::Serialize(ar);
        if (ar.IsStoring())
            ar << m_error;
        else
            ar >> m_error;
    }

void CErrorMessage::DoAction(CObjectSocket *pOS)
    {
        AfxMessageBox(m_error, MB_OK|MB_ICONSTOP);
        delete this;
    }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Program Settings

RegSettings definition file

```cpp
#ifndef _REGSETTINGS_H_
define _REGSETTINGS_H_

#include "RegKeyFolder.h"
#include "RegKeyValue.h"

class CRegSettings
{
    CRegKeyFolder m_keyf;
public:
    CRegSettings();
    virtual ~CRegSettings();

    BOOL Open(int resourceId, LPCTSTR c="");
    void Close();

    CString LoadString(int resourceId);
    LPPBYTE LoadBinary(int resourceId, int* size);
    DWORD LoadDWORD(int resourceId);

    CString LoadString(LPCTSTR name);
    LPPBYTE LoadBinary(LPCTSTR name, int* size);
    DWORD LoadDWORD(LPCTSTR name);

    void StoreString(int resourceId, LPPCTSTR str);
    void StoreBinary(int resourceId, LPPBYTE d, int size);
    void StoreDWORD(int resourceId, DWORD v);

    void StoreString(LPCTSTR name, LPPCTSTR str);
    void StoreBinary(LPCTSTR name, LPPBYTE d, int size);
    void StoreDWORD(LPCTSTR name, DWORD v);

};
#endif
```

RegSettings implementation file

```cpp
#include "stdafx.h"
#include "RemoteWatch.h"
#include "RegSettings.h"

#ifndef _DEBUG
#define _DEBUG
#define THIS_FILE
static char THIS_FILE[]="_.FILE_."
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////

CRegSettings::CRegSettings() : m_keyf(HKEY_CURRENT_USER)
{
}
CRegSettings::~CRegSettings()
{
    Close();
}
BOOL CRegSettings::Open(int resourceId, LPCTSTR c)
{
    return m_keyf.OpenSubFolder(resourceId, c);
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
void CRegSettings::Close()
{
  m_keyf.Close();
}

CString CRegSettings::LoadString(int resourceId)
{
  static CString str(""");
  CRegKeyValue *val=m_keyf.LoadValueByNaae(resourceId);
  if(IS_VALID_PTR(val))
  {
    str=val->GetValue();
    DELETE_PTR(val);
  }
  return str;
}

LPBYTE CRegSettings::LoadBinary(int resourceId, int & size)
{
  LPBYTE bin=NULL;
  size=0;
  CRegKeyValue *val=m_keyf.LoadValueByNaae(resourceId);
  if(IS_VALID_PTR(val))
  {
    size=val->GetSize();
    bin=new BYTE[size];
    if(IS_VALID_PTR(bin))
    {
      memcpy(bin,val->GetValue(),size);
      DELETE_PTR(val);
    }
  }
  return bin;
}

DWORD CRegSettings::LoadDWORD(int resourceId)
{
  DWORD dw=0;
  CRegKeyValue *val=m_keyf.LoadValueByNaae(resourceId);
  if(IS_VALID_PTR(val))
  {
    dw=*( (DWORD • ) val->GetValue());
    DELETE_PTR(val);
  }
  return dw;
}

CString CRegSettings::LoadString(LPCSTR name)
{
  static CString str(""");
  CRegKeyValue *val=m_keyf.LoadValueByNaae(name);
  if(IS_VALID_PTR(val))
  {
    str=val->GetValue();
    DELETE_PTR(val);
  }
  return str;
}

LPBYTE CRegSettings::LoadBinary(LPCSTR name, int & size)
{
  LPBYTE bin=NULL;
  size=0;
  CRegKeyValue *val=m_keyf.LoadValueByNaae(name);
  if(IS_VALID_PTR(val))
  {
    size=val->GetSize();
    bin=new BYTE[size];
    if(IS_VALID_PTR(bin))
    {
      memcpy(bin,val->GetValue(),size);
      DELETE_PTR(val);
    }
  }
}
return bin;
}

DWORD CRegSettings::LoadDWORD(LPCSTR name)
{
    DWORD dw=0;
    CRegKeyValue *val=m_keyf.LoadValueByName(name);
    if(IS_VALID_PTR(val))
    {
        dw=((DWORD*)val->GetValue());
        DELETE_PTR(val);
    }
    return dw;
}

void CRegSettings::StoreString(int resourceId,LPCSTR str)
{
    CRegKeyValue val(resourceId,str);
    m_keyf.StoreValue(val);
}

void CRegSettings::StoreBinary(int resourceId,LPBYTE d,int size)
{
    CRegKeyValue val(resourceId,d,size);
    m_keyf.StoreValue(val);
}

void CRegSettings::StoreDWORD(int resourceId,DWORD v)
{
    CRegKeyValue val(resourceId,v);
    m_keyf.StoreValue(val);
}

void CRegSettings::StoreString(LPCSTR name,LPCSTR str)
{
    CRegKeyValue val(name,str);
    m_keyf.StoreValue(val);
}

void CRegSettings::StoreBinary(LPCSTR name,LPBYTE d,int size)
{
    CRegKeyValue val(name,d,size);
    m_keyf.StoreValue(val);
}

void CRegSettings::StoreDWORD(LPCSTR name,DWORD v)
{
    CRegKeyValue val(name,v);
    m_keyf.StoreValue(val);
}
CRegKeyFolder definition file

ifndef _REGKEYFOLDER_H_
define _REGKEYFOLDER_H_

#include "RegKeyValue.h"

class CRegKeyFolder
{
    HKEY m_Key;
    HKEY m_Parent;
    CString m_KeyName;
    DWORD m_Disp;
public:
    CRegKeyFolder(HKEY parent);
    virtual ~CRegKeyFolder();
    operator HKEY() {return m_Key;}

    BOOL CreateSubFolder(LPCTSTR name);
    BOOL CreateSubFolder(int name);
    BOOL OpenSubFolder(LPCTSTR name);
    BOOL OpenSubFolder(int name);
    BOOL OpenSubFolder(int name, LPCTSTR name2);
    CRegKeyFolder *OpenSubFolderByIndex(int idx);

    BOOL DeleteSubFolder(LPCTSTR name);
    BOOL DeleteSubFolder(int name);

    void Close();

    LPCTSTR GetName();

    CRegKeyValue *LoadValueByIndex(int idx);
    CRegKeyValue *LoadValueByName(LPCTSTR name);
    CRegKeyValue *LoadValueByName(int name);

    BOOL StoreValue(CRegKeyValue value);
    BOOL DeleteValue(LPCTSTR name);
};
#endif

RegKeyFolder implementation file

ifndef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]="RegKeyFolder.h"
define new _DEBU
#endif

#include "stdafx.h"
#include "RemoteWatch.h"
#include "RegKeyFolder.h"

ifndef _DEBUG

CRegKeyFolder::CRegKeyFolder(HKEY parent)
{
    m_Parent=parent;
    INIT_PTR(m_Key);
}

CRegKeyFolder::~CRegKeyFolder()
BOOL CRegKeyFolder::CreateSubFolder(LPCTSTR name)
{
    if(!IS_VALID_PTR(m_Parent))
        return FALSE;
    if(RegCreateKeyEx(m_Parent.name, 0,
        NULL,
        REG_OPTION_NON_VOLATILE,
        KEY_ALL_ACCESS,
        NULL,
        &m.Key.m_Display)!=ERROR_SUCCESS)
        return FALSE;
    mKeyName=name;
    return TRUE;
}

BOOL CRegKeyFolder::CreateSubFolder(int name)
{
    CMString sname(name);
    return CreateSubFolder(sname);
}

BOOL CRegKeyFolder::OpenSubFolder(LPCTSTR name)
{
    if(!IS_VALID_PTR(m_Parent))
        return FALSE;
    if(RegOpenKeyEx(m_Parent.name, 0,
        KEY_ALL_ACCESS,
        &m.Key)!=ERROR_SUCCESS)
        return FALSE;
    mKeyName=name;
    return TRUE;
}

BOOL CRegKeyFolder::OpenSubFolder(int name)
{
    CMString sname(name);
    return OpenSubFolder(sname);
}

BOOL CRegKeyFolder::OpenSubFolder(int name, LPCTSTR namex)
{
    CMString sname(name);
    return OpenSubFolder(sname+namex);
}

CRegKeyFolder *CRegKeyFolder::OpenSubFolderByIndex(int idx)
{
    if(!IS_VALID_PTR(m_Key))
        return NULL;
    CRegKeyFolder *sf=new CRegKeyFolder(m_Key);
    if(!IS_VALID_PTR(sf))
        return NULL;
    DWORD knamesize=256;
    CString kname(" ",knamesize);
    DWORD kclasssize=256;
    CString kclass(" ",kclasssize);
    FILETIME ft;
    if(RegEnumKeyEx(m_Key,
        idx,
        (char*)((LPCTSTR)kname,
        &knamesize,
        &kclass,
        &kclasssize,
        &ft)
        return NULL;
    }
NULL,
 (char*)(LPCTSTR)kclass,
 &kclasssize,
 &ft) !=ERROR_SUCCESS)
 return NULL;
 if(sf->OpenSubFolder(kname))
 return sf;
 else
 DELETE_PTR(sf);
 return NULL;

BOOL CRegKeyFolder::DeleteSubFolder(LPCTSTR name)
{
 if(!IS_VALID_PTR(m_Key))
 return FALSE;
 return (RegDeleteKey(m_Key.name)==ERROR_SUCCESS);
}

BOOL CRegKeyFolder::DeleteSubFolder(int name)
{
 CMString sname(name);
 return DeleteSubFolder(sname);
}

void CRegKeyFolder::Close()
{
 if(IS_VALID_PTR(m_Key))
 {
 RegCloseKey(m_Key);
 INIT_PTR(m_Key);
 }
}

LPCTSTR CRegKeyFolder::GetName()
{
 return m.KeyName;
}

CRegKeyValue *CRegKeyFolder::LoadValueByIndex(int idx)
{
 if(!IS_VALID_PTR(m_Key))
 return NULL;
 CRegKeyValue *val=new CRegKeyValue;
 if(!IS_VALID_PTR(val))
 return NULL;
 if(val->LoadValueByIndex(m_Key, idx))
 return val;
 DELETE_PTR(val);
 return NULL;
}

CRegKeyValue *CRegKeyFolder::LoadValueByName(LPCTSTR name)
{
 if(!IS_VALID_PTR(m_Key))
 return NULL;
 CRegKeyValue *val=new CRegKeyValue;
 if(!IS_VALID_PTR(val))
 return NULL;
 if(val->LoadValueByName(m_Key, name))
 return val;
 DELETE_PTR(val);
 return NULL;
}
CRegKeyValue = CRegKeyFolder::LoadValueByName(int name)
{
    if(!IS_VALID_PTR(m_Key))
        return NULL;
    CRegKeyValue *val = new CRegKeyValue;
    if(!IS_VALID_PTR(val))
        return NULL;
    if(val->LoadValueByName(m_Key,name))
        return val;
    DELETE_PTR(val);
    return NULL;
}

BOOL CRegKeyFolder::StoreValue(CRegKeyValue &val)
{
    if(!IS_VALID_PTR(m_Key))
        return FALSE;
    return val.StoreValue(m_Key);
}

BOOL CRegKeyFolder::DeleteValue(LPCTSTR name)
{
    if(!IS_VALID_PTR(m_Key))
        return FALSE;
    return (RegDeleteValue(m_Key,name)==ERROR_SUCCESS);
};
CRegKeyValue definition file

#ifndef _CREGKEYVALUE_H_
#define _CREGKEYVALUE_H_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CRegKeyValue
{
    LPBYTE m_value;
    DWORD m_size;
    CString m_name;
    DWORD m_type;

public:
    CRegKeyValue(DWORD size=1024); // define value buffer size
    CRegKeyValue(LPCTSTR name, LPBYTE value, DWORD size); // value's name is passed as a first parameter
    CRegKeyValue(LPCTSTR name, DWORD value); // initialize string value
    CRegKeyValue(LPCTSTR name, DWORD value); // initialize D Word value
    CRegKeyValue(LPCTSTR name, DWORD value, DWORD size); // initialize binary data value
    CRegKeyValue(int name, LPCTSTR value); // initialize string value
    CRegKeyValue(int name, DWORD value); // initialize D Word value
    CRegKeyValue(int name, LPCTSTR value, DWORD size); // value's name index in a string table is passed as a first parameter

    virtual ~CRegKeyValue();

    BOOL LoadValueByName(HKEY parent, LPCTSTR value);
    BOOL LoadValueByName(HKEY parent, DWORD value);
    BOOL LoadValueByIndex(HKEY parent, int idx);
    BOOL StoreValue(HKEY parent);
    LPTSTR GetName();
    DWORD GetType();
    LPBYTE GetValue();
    DWORD GetSize();

    void SetName(LPCTSTR name);
    void SetName(int name);

    void SetType(DWORD type);
    void SetValue(LPCTSTR value);
    void SetValue(DWORD value);
    void SetValue(LPBYTE value, DWORD size);

protected:
    BOOL AllocateBuffer(DWORD size);
    void DeallocateBuffer();
};
#endif

RegKeyValue implementation file

#include "stdafx.h"
#include "RemoteWatch.h"
#include "RegKeyValue.h"

#ifndef _DEBUG
#undef _DEBUG
#endif

static char THIS_FILE[]="..FILE..";
#define new DEBUG_NEW

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
// Construction/Destruction

CRegKeyValue::CRegKeyValue(DWORD size)
{
    m_name="";
    INIT_PTR(m_value);
    AllocateBuffer(size);
}

CRegKeyValue::CRegKeyValue(LPCTSTR name,LPCTSTR value)
{
    INIT_PTR(m_value);
    SetName(name);
    SetValue(value);
}

CRegKeyValue::CRegKeyValue(LPCTSTR name,DWORD value)
{
    INIT_PTR(m_value);
    SetName(name);
    SetValue(value);
}

CRegKeyValue::CRegKeyValue(LPCTSTR name,LPBYTE value, DWORD size)
{
    INIT_PTR(m_value);
    SetName(name);
    SetValue(value,size);
}

CRegKeyValue::CRegKeyValue(int name,LPCTSTR value)
{
    INIT_PTR(m_value);
    SetName(name);
    SetValue(value);
}

CRegKeyValue::CRegKeyValue(int name,DWORD value)
{
    INIT_PTR(m_value);
    SetName(name);
    SetValue(value);
}

CRegKeyValue::CRegKeyValue(int name,LPBYTE value, DWORD size)
{
    INIT_PTR(m_value);
    SetName(name);
    SetValue(value,size);
}

CRegKeyValue::~CRegKeyValue()
{
    DeallocateBuffer();
}

BOOL CRegKeyValue::AllocateBuffer(DWORD size)
{
    DeallocateBuffer();
    m_value=new BYTE [size];
    if(!IS_VALID_PTR(m_value))
        return FALSE;
    memset(m_value,0,size);
    m_size=size;
return TRUE;
}

void CRegKeyValue::DeallocateBuffer()
{
    if(IS_VALID_PTR(m_value))
    {
        delete [] m_value;
        INIT_PTR(m_value);
    }
}

BOOL CRegKeyValue::LoadValueByName(HKEY parent, LPCSTR valuename)
{
    SetName(valuename);
    if(RegQueryValueEx(parent,
        (LPCSTR)m_name,
        NULL,
        (LPDWORD)&m_type,
        (LPBYTE)m_value,
        (LPDWORD)&m_size)!=ERROR_SUCCESS)
        return FALSE;
    return TRUE;
}

BOOL CRegKeyValue::LoadValueByName(HKEY parent, int valuename)
{
    SetName(valuename);
    if(RegQueryValueEx(parent,
        (LPCSTR)m_name,
        NULL,
        (LPDWORD)&m_type,
        (LPBYTE)m_value,
        (LPDWORD)&m_size)!=ERROR_SUCCESS)
        return FALSE;
    return TRUE;
}

BOOL CRegKeyValue::LoadValueByIndex(HKEY parent, int idx)
{
    DWORD namesize=256;
    CString name("\0",namesize);
    if(RegEnumValue(parent,
        (DWORD)idx,
        (LPTSTR)&name,
        (LPDWORD)&namesize,
        NULL,
        (LPDWORD)&m_type,
        (LPBYTE)m_value,
        (LPDWORD)&m_size)!=ERROR_SUCCESS)
        return FALSE;
    m_name=name;
    return TRUE;
}

BOOL CRegKeyValue::StoreValue(HKEY parent)
{
    return (RegSetValueEx(parent,
        (LPTSTR)m_name,
        0,
        (DWORD)m_type,
        (LPBYTE)m_value,
        (DWORD)m_size)==ERROR_SUCCESS);
}

LPCTSTR CRegKeyValue::GetName()
DWORD CRegKeyValue::GetType()
{
    return m_type;
}

LPBYTE CRegKeyValue::GetValue()
{
    return (LPBYTE)m_value;
}

DWORD CRegKeyValue::GetSize()
{
    return m_size;
}

void CRegKeyValue::SetName(LPCTSTR name)
{
    m_name = name;
}

void CRegKeyValue::SetName(int name)
{
    m_name.LoadString(name);
}

void CRegKeyValue::SetType(DWORD type)
{
    m_type = type;
}

void CRegKeyValue::SetValue(LPCTSTR value)
{
    CString val = value;
    if (AllocateBuffer(val.GetLength() + 1))
    {
        memcpy(m_value, val.m_size);
        m_type = REG_SZ;
    }
}

void CRegKeyValue::SetValue(DWORD value)
{
    if (AllocateBuffer(sizeof(DWORD)))
    {
        *(DWORD*)m_value = value;
        m_type = REG_DWORD;
    }
}

void CRegKeyValue::SetValue(LPBYTE value, DWORD size)
{
    if (AllocateBuffer(size))
    {
        memcpy(m_value, value, m_size);
        m_type = REG_BINARY;
    }
}
Video Input and Processing

CBT848 definition file

////////////////////////////////////////////////////////////////////////
// BT848, BT849, BT878, BT879 register access class
////////////////////////////////////////////////////////////////////////
#ifndef _BT848_H_
#define _BT848_H_

// Register addresses
const int DTSTATUS = 0x00;
const int IFORM = 0x04;
const int TDEC = 0x08;
const int E_CROP = 0x0C;
const int O_CROP = 0x8C;
const int E_VDELAY_LO = 0x10;
const int O_VDELAY_LO = 0x90;
const int E_VACTIVE_LO = 0x14;
const int O_VACTIVE_LO = 0x94;
const int E_DELAY_LO = 0x18;
const int O_DELAY_LO = 0x98;
const int E_HACTIVE_LO = 0x1C;
const int O_HACTIVE_LO = 0x9C;
const int E_VSSCALE_HI = 0x20;
const int O_VSCALE_HI = 0xA0;
const int E_VSCALE_LO = 0x24;
const int O_VSCALE_LO = 0xA4;
const int BRIGHT = 0x28;
const int E_CONTROL = 0x2C;
const int O_CONTROL = 0x8C;
const int CONTRAST_LO = 0x30;
const int SAT_U_LO = 0x34;
const int SAT_Y_LO = 0x38;
const int HUE = 0x3C;
const int E_SCOOP = 0x40;
const int O_SCOOP = 0x40;
const int UFORM = 0x48;
const int E_VSCALE_HI = 0x4C;
const int O_VSCALE_HI = 0xCC;
const int E_VSCALE_LO = 0x50;
const int O_VSCALE_LO = 0xD0;
const int TEST = 0x54;
const int ADELAY = 0x58;
const int BDELAY = 0x64;
const int ADC = 0x68;
const int E_VTC = 0x6C;
const int O_VTC = 0xEC;
const int SRESET = 0x7C;
const int COLOR_FMT = 0x80;
const int COLOR_CTL = 0x88;
const int CAP_CTL = 0xDC;
const int VBI_PACK_SIZE = 0xE0;
const int VBI_PACK_DEL = 0xE4;
const int PLL_F_LO = 0xF0;
const int PLL_F_HI = 0xF4;
const int PLL_XC = 0xF8;
const int DVSIF = 0xFC;
const int FCAP = 0xE8;
const int INT_STAT = 0x100;
const int INT_MASK = 0x104;
const int GPIO_DMA_CTL = 0x10C;
const int I2C = 0x110;
const int RISC_STRT_ADD = 0x114;
const int GPIO_OUT_EN = 0x118;
const int GPIO_REG_IMP = 0x11C;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
const int RISC_COUNT = 0x120;
const int GPIO_DATA = 0x200;

// Data formats
const int RGB32 = 0x00;
const int RGB24 = 0x11;
const int RGB16 = 0x22;
const int RGB15 = 0x33;
const int YUV2 = 0x44;
const int BTYUV = 0x55;
const int Y8 = 0x66;
const int RGB8 = 0x77;
const int YUV2P = 0x88;
const int BTYUVP = 0x99;
const int RAW8 = 0xee;

// RISC Instructions
const int RESYEVENTSTART = 0x8C00800C;
const int RESYPACKEDDATA = 0x8C008006;
const int RESYPLANEARDATA = 0x8C00800E;
const int SYNCDDSTART = 0x8C000004;
const int SYNCVEVENTSTART = 0x8C00000C;
const int JUMP = 0x7C000000;
const int WRITE768e = 0x1C30300;
const int WRITE768o = 0x1C3C0300;
const int WRITE123Y = 0x9C000300;
const int WRITE123UC = 0x01800180;
const int SKIPY1 = 0xACE10300;
const int SKIPY2 = 0xACD20300;
const int SKIPY3 = 0xAC30300;
const int SKIPY4 = 0xACB40300;
const int SKIPUV = 0x01800180;

#include "TVicHW32s.h"

// CBT848 class
class CBT848
{
    unsigned int m_BTAddress;
    TVicHW32Descriptor *HW32;
public:
    CBT848();
    virtual ~CBT848();

    void InitBT();
    void ResetBT();
    bool BTFound();
    void BTSelECTMUX(int i);

private:
    bool ReadBT(int, unsigned int*);
    bool WriteBT(int, unsigned int);
    unsigned int FindAddress(unsigned short, unsigned short);
};

#endif

CBT848 implementation file

#pragma warning(disable: 4100 4201 4514)

#include "stdafx.h"

#include "BT848.h"
const unsigned short BROOKTREE = 0x109E;

const unsigned short BT848 = 0x0350;
const unsigned short BT849 = 0x0351;
const unsigned short BT878 = 0x036E;
const unsigned short BT879 = 0x036F;

CBT848::CBT848()
{
    HW32=NULL;
    _BTAddress = FindAddress(BROOKTREE,BT848);
    if(_BTAddress == 0)
    {
        _BTAddress = FindAddress(BROOKTREE,BT849);
        if(_BTAddress == 0)
        {
            _BTAddress = FindAddress(BROOKTREE,BT878);
            if(_BTAddress == 0)
                _BTAddress = FindAddress(BROOKTREE,BT879);
        }
    }
    if(BTFound() == true)
    {
        HW32=OpenVicHW32(NULL);
        if(HW32->fOpenDrive == FALSE)
            _BTAddress = 0;
    }
}

CBT848::CBT848()
{
    if(BTFound() == true)
    {
        CloseVicHW32(HW32);
        _BTAddress = 0;
    }
}

bool CBT848::BTFound()
{
    if(_BTAddress != 0)
        return true;
    return false;
}

void CBT848::InitBT()
{
    WriteBT(GPIO_OUT_EU, 0xffff);
}

void CBT848::ResetBT()
{
    WriteBT(E_VSCALE_HI, 0x60);
    WriteBT(O_VSCALE_HI, 0x60);
    WriteBT(E_SCLoop, 0x40);
    WriteBT(O_SCLoop, 0x40);
    WriteBT(E_CONTROL, 0x00);
    WriteBT(O_CONTROL, 0x00);
    WriteBT(SAT_U_LO, 0xFE);
    WriteBT(SAT_V_LO, 0xB4);
    WriteBT(CONTRAST_LO, 0xD8);
    WriteBT(DVSIF, 0x00);
void CBT848::BTSelectMUX( int m )
{
    WriteBT(GPIO_DATA, m&7);
}
```cpp
bool CBT848::ReadBT(int nRegister, unsigned int nValue)
{
    if (BTFound() == false)
        return false;

    if (GetActiveHW(HW32) == FALSE)
        return false;

    unsigned int *pPhysical = (unsigned int*)MapPhysToLinear(HW32, m_BTAddress + nRegister, 0x04);
    *nValue = *pPhysical;
    return true;
}

bool CBT848::WriteBT(int nRegister, unsigned int nValue)
{
    if (BTFound() == false)
        return false;

    if (GetActiveHW(HW32) == FALSE)
        return false;

    unsigned int *pPhysical = (unsigned int*)MapPhysToLinear(HW32, m_BTAddress + nRegister, 0x04);
    *pPhysical = nValue;
    return true;
}

unsigned int CBT848::FindAddress(unsigned short nVendorID, unsigned short nDeviceID)
{
    // Check if we are running under NT
    OSVERSIONINFO sOSVersion;

    sOSVersion.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);
    GetVersionEx(&sOSVersion);

    // Cannot access the hardware registers
    if (sOSVersion.dwPlatformId == VER_PLATFORM_WIN32_NT)
        return 0;

    unsigned int nAddress = 0;
    __asm
    {
        push ax;
        push bx;
        push cx;
        push dx;
        push si;

        mov cx, nDeviceID; // DeviceID
        mov dx, nVendorID; // VendorID
        mov si, 0; // Index
        mov ah, 0xb1; // PCI_FUNCTION_ID
        mov al, 0xA2; // Find PCI Device
        int Oxia; // BIOS Int 1ah call
        // BH = Bus number
        // BL = Device number
        jc ErrorBT; // When CARRY FLAG is set -> error
        cmp ah, 0x00;
        jnz ErrorBT; // When AH is 0x00 -> error

        mov di, 0x10; // Register PCI_BASE_ADDRESS_0
        mov ah, 0xb1; // PCI_FUNCTION_ID
        mov al, 0x0a; // PCI Konfiguration DWORD
    }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
int Ox1a;           // BIOS Int 1Ah call

jc ErrorBT;         // Wenn das CARRY FLAG gesetzt ist, ist ein Fehler aufgetreten
czp ah,00;
jnz ErrorBT;        // Wenn AH null ist, ist ein Fehler aufgetreten
mov nAddress,ecx;

ErrorBT:
    pop si;
    pop dx;
    pop cx;
    pop bx;
    pop ax;
}
return nAddress & 0xffffffff;
CaptWnd definition file

```c
#ifndef _CAPTWND_H_
#define _CAPTWND_H_

#include <fwv.h>
#include <cnreg.h>
#include "Bt843.h"
#define MAXVIDDRIVER 10
#define MAXFRAME_SIZE (640L*480L*3L)
#define BITSPIX 24
#define W(X) ((X).right - (X).left)
#define H(X) ((X).bottom - (X).top)

class CaptWnd : public CWnd
{
    int m_fps;
    int m_gwDeviceIndex;
    long m_framesize;

public:
    int m_mux;
    int m_prev_mux;
    CBT848 m_cbt;
    int m_cam_seq;
    int m_InCamera;
    CWnd *m_fWnd;
    char *m_InFrame, *m_OutFrame;
    CEvent *m_cev;

public:
    CaptWnd(CWnd *fWnd, CEvent *cev, int camseq, int fps, int width, int height);
    ~CaptWnd();

    int IsDriverOpen();
    void SetFPS(int fps);
    int GetFPS();
    void SetCamSequence(int camseq);
    long GetFrameSize();
    void GetInFrame(char *buf);
    void GetOutFrame(char *buf, int *cam);
    void SetOutFrame(char *buf);
    void CaptureVideo();
    void AbortCapture();
    void Preview();
    void Overlay();
    void VideoSource();
    void VideoFormat();

protected:
    virtual void PostNcDestroy();
    void CenterCaptureWindow(CWnd *fWnd);
    void SetFrameSize();

    static RESULT CALLBACK ErrorCallbackProc(HWND hWnd, int nErrID, LPTSTR lpErrorText);
    static RESULT CALLBACK VideoCallbackProc(HWND hWnd, LPVIDEODRIVE hdr);
};
#endif
```

CaptWnd implementation file

```c
#include "stdafx.h"
#include "CaptWnd.h"
```
ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CaptWnd *pCW;

CaptWnd::CaptWnd(CWnd *fWnd, CEvent *cev, int camseq, int fps, int width, int height)
{
    assert(fWnd);
    assert(cev);
    assert(fps);

    m_mux = 7;
    m_prev_mux = 0;
    m_fWnd = fWnd;
    m_gwDeviceIndex = -1;
    INIT_PTR(m_InFrame);
    INIT_PTR(m_OutFrame);
    m_InCamera = 0;
    m_fps = fps;
    m_cev = cev;
    m_cam_seq = camseq;

    // Set the global CWnd pointer for static functions
    pCW = this;
    HWnd ghWndCap = capCreateCaptureWindow("CaptWnd",
                                           WS_CHILD,
                                           0, 0, width, height,
                                           m_fWnd->GetSafeHwnd(), 0);

    if(!ghWndCap)
        return;

    // Subclass the window
    SubclassWindow(ghWndCap);

    // Register the status and error callbacks before driver connect
    capSetCallbackOnError(GetSafeHwnd(), ErrorCallbackProc);
    capSetCallbackOnVideoStream(GetSafeHwnd(), VideoCallbackProc);

    char achDeviceName[80];
    char achDeviceVersion[100];

    // Open the first available driver...
    for(WORD wIndex = 0; wIndex < MAXVIDDRIVERS; wIndex++)
    {
        if( capGetDriverDescription(wIndex,
                                       (LPTSTR)achDeviceName, sizeof(achDeviceName)/sizeof(char),
                                       (LPTSTR)achDeviceVersion, sizeof(achDeviceVersion)/sizeof(char)))
        {
            // Only if no other driver is already connected
            if(capDriverConnect(GetSafeHwnd(), wIndex))
            {
                m_gwDeviceIndex = wIndex;
                break;
            }
        }
    }

    CenterCaptureWindow(m_fWnd);
    SetFrameSize();
    m_cbt.InitBT();
    m_cbt.BTSelectMUX(m_InCamera);
}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CaptWnd::CaptWnd()
{
    if(IS_VALID_PTR(m_InFrame)) delete [] m_InFrame;
    if(IS_VALID_PTR(m_OutFrame)) delete [] m_OutFrame;
}

void CaptWnd::PostNcDestroy()
{
    AbortCapture();
    if(IsDriverOpen())
    {
        capPreview(GetSafeHwnd(),FALSE);
        // Unregister the status and error callbacks before driver connect
        capSetCallbackOnError(GetSafeHwnd(),NULL);
        capSetCallbackOnVideoStream(GetSafeHwnd(),NULL);
        capDriverDisconnect(GetSafeHwnd());
        m_gwDeviceIndex=-1;
    }
    delete this;
}

int CaptWnd::IsDriverOpen()
{
    return!(m_gwDeviceIndex==-1)&&GetSafeHwnd());
}

void CaptWnd::SetFrameSize()
{
    RECT wr;
    BITMAPINFO bi;
    if(!IsDriverOpen())
    return;
    capGetVideoFormat(GetSafeHwnd(),&bi,sizeof(BITMAPINFO));
    GetClientRect(&wr);
    bi.bmiHeader.biSize = sizeof( BITMAPINFOHEADER );
    bi.bmiHeader.biWidth = W(wr);
    bi.bmiHeader.biHeight = H(wr);
    bi.bmiHeader.biPlanes = 1;
    bi.bmiHeader.biBitCount = BITSPIX;
    bi.bmiHeader.biCompression = 0;
    bi.bmiHeader.biSizeImage = bi.bmiHeader.biWidth *
        bi.bmiHeader.biHeight * bi.bmiHeader.biBitCount / 8;
    bi.bmiHeader.biClrUsed = 0;
    bi.bmiHeader.biClrImportant = 0;
    m_framesize = bi.bmiHeader.biSizeImage;
    // Allocate frame buffers
    m_InFrame = new char [m_framesize];
    memset( m_InFrame, 0, m_framesize );
    m_OutFrame = new char [m_framesize];
    memset( m_OutFrame, 0, m_framesize );
    capSetVideoFormat( GetSafeHwnd(), &bi, sizeof( BITMAPINFO ) );
}

LRRESULT CALLBACK CaptWnd::ErrorCallbackProc(HWND hWnd, int nErrID, LPTSTR lpErrorText)
{
    if( nErrID == 0 )
        return TRUE;
    char buf[256];

    Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
assert(pCW->m_fWnd);
capDriverDisconnect(pCW->GetSafeHwnd());

// Show the error ID and text
wsprintf(buf, TEXT("Error# %d\r\n\nError: %s\r\n\nMB_OK | MB_ICONEXCLAMATION\r\n"), nErrID);
(pCW->m_fWnd)->MessageBox(buf,
MB_OK | MB_ICONEXCLAMATION);

RESULT CALLBACK CaptWnd::VideoCallbackProc(HWND hWnd, LPVIDEODATA lpVHdr)
{
    assert(pCW->m_cev);
    assert(pCW->m_InFrame);
    assert(pCW->m_OutFrame);

    for(int i=0; i<8; i++) // go through all the cameras
    {
        pCW->m_mux=(pCW->m_mux+i)%8;
        if(pCW->m_cams_seq & (1<<i))
            break;
    }
    pCW->m_out.BTSelectMUX(pCW->m_mux);
    memcpy(pCW->m_InFrame, lpVHdr->lpData, lpVHdr->dwBufferLength);
    pCW->m_InCamera=pCW->m_prev_mux;
    pCW->m_prev_mux=pCW->m_mux;
    // use prev_mux value...
    pCW->m_cev->PulseEvent();
    return (RESULT) TRUE;
}

void CaptWnd::CenterCaptureWindow(CWnd *fWnd)
{
    RECT MainRect, CapRect;
    WORD wCapXPos, wCapYPos;
    if(!IsDriverOpen())
        return;

    fWnd->GetClientRect(&MainRect);
    GetClientRect(&CapRect);
    wCapXPos = max(0, (W(MainRect) - W(CapRect)) / 2);
    wCapYPos = max(0, (H(MainRect) - H(CapRect)) / 2);

    MoveWindow(wCapXPos, wCapYPos, W(CapRect), H(CapRect), FALSE);
}

long CaptWnd::GetFrameSize()
{
    return m_framesize;
}

void CaptWnd::GetInFrame(char *buf)
{
    memcpy(buf, m_InFrame, m_framesize);
}

void CaptWnd::GetInFrame(char *buf, int *cam)
{
    memcpy(buf, m_InFrame, m_framesize);
    *cam=m_InCamera;
}
void CaptWnd::SetOutFrame(char *buf)
{
    memcpy(m_OutFrame, buf, m_framesize);
}

void CaptWnd::SetFPS(int fps)
{
    m_fps = fps;
}

int CaptWnd::GetFPS()
{
    return m_fps;
}

void CaptWnd::SetCamSequence(int camseq)
{
    m_cam_seq = camseq;
}

void CaptWnd::CaptureVideo()
{
    CAPTUREPARAMS s;
    if (!IsDriverOpen())
        return;

    // disable overlay and preview channels
    capOverlay(GetSafeHwnd(), FALSE);
    capPreview(GetSafeHwnd(), FALSE);

    capCaptureGetSetup(GetSafeHwnd(), &s, sizeof(CAPTUREPARAMS));
    s.dwRequestMicroSecPerFrame = 10000000 / m_fps;
    s.fCaptureAudio = FALSE;  // Do not capture audio
    s.fLimitEnabled = FALSE;   // Capture time limit
    s.wPercentDropForError = 0;  // Frame drop percentage
    s.fYield = TRUE;          // Let another process run
    s.fAbortLeftMouse = FALSE;  // Frame drop percentage
    s.fAbortRightMouse = FALSE;  // Frame drop percentage
    s.vKeyAbort = 0;           // Frame drop percentage

    capCaptureSetSetup(GetSafeHwnd(), &s, sizeof(CAPTUREPARAMS));
    capCaptureSequenceNoFile(GetSafeHwnd());
    ShowWindow(SW_HIDE);
}

void CaptWnd::AbortCapture()
{
    if (!IsDriverOpen())
        return;

    ShowWindow(SW_HIDE);
    capCaptureAbort(GetSafeHwnd());
    Sleep(200);
    capCaptureAbort(GetSafeHwnd());
}

void CaptWnd::Preview()
{
    CAPSTATUS gCapStatus;
    if (!IsDriverOpen())
        return;
capGetStatus( GetSafeHwnd(), &gCapStatus, sizeof(CAPSTATUS) );
capPreview( GetSafeHwnd(), 'gCapStatus.fLiveWindow );

void CaptWnd::Overlay()
{
    CAPSTATUS gCapStatus;
    if( 'IsDriverOpen() )
        return;

capGetStatus( GetSafeHwnd(), &gCapStatus, sizeof(CAPSTATUS) );
capOverlay( GetSafeHwnd(), 'gCapStatus.fOverlayWindow );
}

void CaptWnd::VideoSource()
{
    CAPDRIVERCAPS gCapDriverCaps;
    if( 'IsDriverOpen() )
        return;

capDriverGetCaps( GetSafeHwnd(), &gCapDriverCaps, sizeof(CAPDRIVERCAPS) );
    if( gCapDriverCaps.fHasDlgVideoSource )
    {
        AbortCapture();
        // Only if the driver has a "Video Source" dialog box
        capDlgVideoSource( GetSafeHwnd() );
        CaptureVideo();
    }
}

void CaptWnd::VideoFormat()
{
    CAPDRIVERCAPS gCapDriverCaps;
    CAPSTATUS gCapStatus;
    if( 'IsDriverOpen() )
        return;

capDriverGetCaps( GetSafeHwnd(), &gCapDriverCaps, sizeof(CAPDRIVERCAPS) );
    if( gCapDriverCaps.fHasDlgVideoFormat )
    {
        AbortCapture();
        // Only if the driver has a "Video Format" dialog box
        if( capDlgVideoFormat( GetSafeHwnd() ) )
        {
            // If successful,
            // Get the new image dimension and center capture window
            capGetStatus( GetSafeHwnd(), &gCapStatus, sizeof(CAPSTATUS) );
            SetWindowPos( NULL, 0, 0, gCapStatus.uiImageWidth,
                          gCapStatus.uiImageHeight, SWP_NOZORDER | SWP_NOMOVE );
            CenterCaptureWindow( m_fWnd );
        }
        CaptureVideo();
    }
}
Video Output

PlayWnd definition file

```c
#define _PLAYWND_H
#endif
#define BPS 24
#define ABS(a) ( (a < 0) ? -a : a )

class PlayWnd : public CWnd
{
    BITMAPINFO m_bi;
    CWnd *m_fWnd;
    char m_buf[256], m_buf1[256];
    CBitmap *m_disp;
    HBITMAP m_hbitmap;
    unsigned char *m_bits,*m_bits1;
    int m_w,m_h,m_framesize;
    int m_cam, m_sid;
    CPoint m_OrgPoint;
    int m_bstate;

public:

    PlayWnd(CWnd *fWnd, int w, int h, int sid);
    ~PlayWnd();

    void SetVideoFrame(unsigned char *d);
    void SetCameraAddress(int cam);
    void SetFrameFocus(BOOL flag);

protected:

    void GetCaptionRect(CRect &prc);
    void GetBtnRect(CRect &br);
    void SetFrameSize();
    void SetSize();
    virtual void PostNcDestroy();
    afx_msg BOOL OnEraseBkgnd(CDC *pDC);
    afx_msg void OnMouseMove(UINT, CPoint);
    afx_msg void OnLButtonDown(UINT, CPoint);
    afx_msg void OnLButtonUp(UINT, CPoint);
    afx_msg BOOL OnMouseWheel(UINT nFlags, short zDelta, CPoint pt);
    afx_msg void OnRButtonDown(UINT, CPoint);
    afx_msg BOOL OnNcActivate(BOOL bActive);
    afx_msg void OnNcLButtonDown(UINT nHitTest, CPoint point);
    afx_msg void OnNcPaint();
    DECLARE_MESSAGE_MAP()
};
#endif
```

PlayWnd implementation file

```c
#include "stdafx.h"
#include "PlayWnd.h"
#include "resource.h"
#include "RemoteWatchDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
#endif

static char THIS_FILE[] = __FILE__;
#endif

DECLARE_MESSAGE_MAP(PlayWnd, CWnd)
```
ON_WM_ERASEBKGND()
ON_WM_MOUSEMOVE()
ON_WM_LBUTTONDOWN()
ON_WM_LBUTTONDOWN()
ON_WM_MOUSEWHEEL()
ON_WM_KEYDOWN()
ON_WM_NCBUTTONDOWN()
ON_WM_NCMOUSEMOVE()
END_MESSAGE_MAP()

#define WS_STYLE       WS_POPUP | WS_DLGFRAME | WS_CAPTION | WS_THICKFRAME

PlayWnd::PlayWnd(CWnd &fWnd, int w, int h, int sid)
{
  m_bstate=0;
  m_sid=sid;
  m_fWnd = fWnd;
  m_w=w;
  m_h=h;
  m_framesize=m_w=m_h*(BPS/8);
  m_disp = new CBitmap();
  m_bits=new unsigned char [w=*h*(BPS/8)];
  m_cam=1;

  CString strWndClass = AfxRegisterWndClass(0,
      AfxGetApp() - LoadStandardCursor(IDC_ARROW),
      (HBRUSH)NULL,
      NULL);

  DWORD exStyle = 0;
  // initial window size
  CRect wr(0, 0, m_w, m_h);
  CreateEx(exStyle, strWndClass, 
      "", WSTYLE, wr, fWnd, 0);
  SetFrameSize();
}

PlayWnd::PlayWnd()
{
  m_disp->Detach();
  delete m_disp;
  DeleteObject(m_hBitmap);
  if(m_bits)
    delete [] m_bits;
}

void PlayWnd::PostNcDestroy()
{
  delete this;
}

void PlayWnd::SetFrameSize()
{
  HBMP HBitmap = NULL;

  m_b.bmiHeader.biSize     = (DWORD)sizeof(BITMAPINFOHEADER);
  m_b.bmiHeader.biPlanes   = 1;
  m_b.bmiHeader.biBitCount = BPS;
  m_b.bmiHeader.biWidth    = m_w;
  m_b.bmiHeader.biHeight   = m_h;
  m_b.bmiHeader.biCompression = BI_RGB;

  PlayWnd(); PlayWnd(CWnd &fWnd, int w, int h, int sid)
m_bmiHeader.biXPelsPerMeter = 0;
m_bmiHeader.biYPelsPerMeter = 0;
m_bmiHeader.biClrUsed = 0;
m_bmiHeader.biClrImportant = 0;
m_bmiHeader.biSizeImage = m_bmiHeader.biWidth*m_bmiHeader.biHeight*(BPS/8);

// comment next line out - for vertically flipped images
// m_bmiHeader.biHeight=-m_bmiHeader.biHeight;
if (hBitmap = CreateDIBSection(GetDC(HWND)this), m_bmi,
DIB_RGB_COLORS,
(void**)&m_bits,
NULL,
0 ) != NULL )
{
    memset(m_bits, 0, m_bmiHeader.biSizeImage);
}

m_disp->Attach(hBitmap);
m_hBitmap = hBitmap;

SetSize();

// Window size multiplier and divider table
int WSizex[] = { 1, 2,
    1, 1,
    3, 2,
    2, 1 };

void PlayWnd::SetSize()
{
    CRect ns(0, 0, (m_w*WSizex[m_sidi2])/WSizex[m_sidi2+1],
    (m_h*WSizex[m_sidx2])/WSizex[m_sidx2+1]);
    AdjustWindowRectEx(&ns, WSIZEX, FALSE, 0);
    ClientToScreen(&ns);
    MoveWindow(&ns, TRUE);
    RedrawWindow();
}

void PlayWnd::Draw(CDC *dc)
{
    CDC dcMem;
    CRect cr;
    GetClientRect(&cr);
    dcMem.CreateCompatibleDC(dc);
    dcMem.SelectObject(mDisp);
    dc->SetStretchBltMode(COLORONCOLOR);
    dc->StretchBlt( 0, 0, cr.Width(), cr.Height(), &dcMem,
    0, 0, m_w, m_h, SRCCOPY );
}

void PlayWnd::SetVideoFrame(unsigned char *d)
{
    // copy the frame data
    memcpy(m_bits,d,m_framesize);
    // show the window
    if (!IsWindowVisible())
    ShowWindow(SW_SHOWN);
    RedrawWindow();
}

void PlayWnd::SetCameraAddress(int cam)
{
    m_cam = cam;
}
BOOL PlayWnd::OnEraseBkgnd(CDC* pDC)
{
    Draw(pDC);
    return 0;
}

void PlayWnd::OnMouseMove(UINT nFlags, CPoint point)
{
    if(GetCapture() != this)
        return;
    CPoint dr(point - m_OrgPoint);
    CClientDC dc(this);
    int lr = (dr.x == 0) ? 0 : (dr.x > 0) ? CAM_RIGHT : CAM_LEFT;
    int ud = (dr.y == 0) ? 0 : (dr.y > 0) ? CAM_DWN : CAM_UP;
    // change the cursors accordingly
    if(lr == CAM_UP & lr == 0)
        SetCursor(AfxGetApp()->LoadCursor(IDC_CURSOR_UP));
    if(lr == CAM_DWN & lr == 0)
        SetCursor(AfxGetApp()->LoadCursor(IDC_CURSOR_DOWN));
    if(lr == CAM_LEFT & lr == 0)
        SetCursor(AfxGetApp()->LoadCursor(IDC_CURSOR_LEFT));
    if(lr == CAM_RIGHT & lr == 0)
        SetCursor(AfxGetApp()->LoadCursor(IDC_CURSOR_RIGHT));
    // send the message to the server
    CCamCtrlMessage msg(m_cam, ud, abs(dr.x) / 5, abs(dr.y) / 5);
    ((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->m_ObjSocket.SendObject(&msg);
}

void PlayWnd::OnLButtonDown(UINT nFlags, CPoint point)
{
    m_OrgPoint = point;
    SetCapture();
    SetCursor(AfxGetApp()->LoadCursor(IDC_CURSOR_MOVE));
}

void PlayWnd::OnLButtonUp(UINT nFlags, CPoint point)
{
    if(GetCapture() != this)
    {
        ReleaseCapture();
        // send the stop message to the server
        CCamCtrlMessage msg(m_cam);
        ((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->m_ObjSocket.SendObject(&msg);
        SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
    }
}

BOOL PlayWnd::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
    // send the message to the server
    CCamCtrlMessage msg(m_cam, (zDelta > 0) ? CAM_ZIN : CAM_ZOUT);
    ((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->m_ObjSocket.SendObject(&msg);
    return CWnd::OnMouseWheel(nFlags, zDelta, pt);
}
void PlayWnd::OnRButtonDown(UINT nFlag, CPoint pt)
{
    CMenu m;
    CPoint pos;
    m.CreatePopupMenu();
    m.AppendMenu(MF_STRING, 1, "1x zoom");
    m.AppendMenu(MF_STRING, 2, "2x zoom");
    m.AppendMenu(MF_STRING, 3, "2x zoom");
    m.AppendMenu(MF_STRING, 4, "2x zoom");
    m.CheckMenuItems(m.sidx+1, MF_CHECKED);
    GetCursorPos(&pos);
    int res=m.TrackPopupMenu(TPM_LEFTALIGN | TPM_LEFTBUTTON, pos.x, pos.y, this);
    if((res>0) && (res<5))
    {
        m.sidx=res-1;
        SetSize();
    }
}

BOOL PlayWnd::OnNCActivate(BOOL bActive)
{
    BOOL res=CWnd::OnNCActivate(bActive);
    // redraw the non client area
    SendMessage(WM_NCPAINT);
    return res;
}

void PlayWnd::OnSCLButtonDown(UINT nHitTest, CPoint point)
{
    if(nHitTest!=HTCAPTION)
    {
        CWnd::OnSCLButtonDown(nHitTest, point);
        return;
    }
    CRect wr;
    GetWindowRect(&wr);
    point.x-=wr.left;
    point.y-=wr.top;
    CRect br;
    GetBtnRect(&br);
    if(br.PtInRect(point))
    {
        if(m.bstate)
        {
            //((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->SetFocusWindow(this, TRUE);
        }
        else
        {
            //((CRemoteWatchDlg*)AfxGetApp()->m_pMainWnd)->SetFocusWindow(this, FALSE);
        }
    }
    CWnd::OnNCButtonDown(nHitTest, point);
}

void PlayWnd::SetFrameFocus(BOOL flag)
{
    if(flag)
    {
        m_bstate=DFCS_CHECKED;
    }
    else
    {
        m_bstate=0;
        SendMessage(WM_NCPAINT);
    }
}

void PlayWnd::GetBtnRect(CRect *br)
{
    CSize bs(GetSystemMetrics(SM_CXSIZE),
    Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
GetSystemMetrics(SM_CYSIZE));

CRect cr;
GetCaptionRect(&cr);

br->left=cr.right-bs.cx;
br->top=cr.top+1;
br->right=br->left+bs.cx-1;
br->bottom=cr.top+bs.cy-1;
}

void PlayWnd::OnNcPaint()
{
    CWnd::OnNcPaint();
    CRect br;
    GetBtnRect(&br);
    CDC *dc=GetWindowDC();
dc->DrawFrameControl(br,DFC_BUTTON,DFCS_BUTTONRADIO)m_bstate);
}

void PlayWnd::GetCaptionRect(CRect* prc)
{
    CSize sizeFrame(GetSystemMetrics(SM_CXFRAME),
                     GetSystemMetrics(SM_CYFRAME));
    CSize sizeBorder(GetSystemMetrics(SM_CXBORDER),
                     GetSystemMetrics(SM_CYBORDER));
    CSize sizeButton(GetSystemMetrics(SM_CXSIZE),
                     GetSystemMetrics(SM_CYSIZE));

    // Get the window rectangle and style information.
    CRect rcWindow;
    DWORD dwStyle = GetStyle();
    GetWindowRect(rcWindow);
    // Set the rectangle initially based on the frame, window
    // rect, and height of a caption-bar button.
    if (dwStyle & WS_THICKFRAME)
    {
        prc->left = sizeFrame.cx;
        prc->top = sizeFrame.cy;
        prc->right = rcWindow.Width() - sizeFrame.cx;
    }
    else
    {
        prc->left = 0;
        prc->top = 0;
        prc->right = rcWindow.Width();
    }
    prc->bottom = prc->top + sizeButton.cy;
}
Video Compression

SCodec definition file

```c
#ifndef _SCODEC_H
#define _SCODEC_H

#include "vfw.h"

class CSCodec
{
    struct CCInfo {
    ICINFO    *pCompInfo;
    int        mComp;
    COMPVARS   *pCV;
    BITMAPINFOHEADER   *bih;
    char        tbuff[128];
    HIC         hic;
    FOURCC      fccSelect;
    ICINFO      *piiCurrent;

    char    szCurrentCompression[256];
    }

    CCInfo m_cci;
    enum{NONE, 0, COMPRESS, 0, DECOMPRESS};

    int     m_width, m_height, m_quality;
    int     m_action;
    char    *m_data, *m_codeccfg;
    long    m_cmmphsize, m_osize;
    HIC     m_hic;
    BITMAPINFOHEADER  *m_bihinput, *m_bihoutput;
    DWORD   m_cckid, m_compf, m_framenum;

public:
    CSCodec();
    virtual 'CSCodec();
    virtual int EnumCodecs(int width, int height);
    virtual CString GetCodecDescription(int idx);
    virtual DWORD GetCodecHandler(int idx);
    virtual int GetCodecIndex(DWORD handler, int width, int height);
    virtual BOOL ConfigureCurCodec();
    virtual BOOL SetCurCodecCfg(char *cfg, int size);
    virtual char *GetCurCodecCfg();
    virtual int GetCurCodecCfgSize();
    virtual DWORD GetCurCodecHandler();

    virtual void Close();
    virtual BOOL InitCompressor(int idx, int width, int height, int quality=0);
    virtual BOOL Compress(char *bytes);
    virtual BOOL InitDecompressor(int idx, int width, int height);
    virtual BOOL Decompress(char *bytes);

    virtual void Begin();
    virtual void End();

    virtual char *GetOutputData();
    virtual long GetOutputSize();

protected:
    void InitBitmapInfoHeader(BITMAPINFOHEADER *bih, int w, int h, int bits, int comp);
};
#endif
```
SCodec implementation file

#include "stdafx.h"
#include "SCodec.h"
#include "RemoteWatch.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]="_FILE_;
#define new DEBUG_NEW
#endif

#include "RemoteWatch.h"

STDMETHODIMP CSCodec::CSCodec()
{
    m.action=Q.NONE;
    m.cci.hic= NULL;
    m.cci.pCurrent= NULL;
    m.cci.pCompInfo = NULL;
    m.cci.nComp = C;
    INIT_PTR(m.cci.fccSelect);
    INIT_PTR(m.odata);
    INIT_PTR(m.hic);
    INIT_PTR(m.bilnput);
    INIT_PTR(m.biOutput);
    m.codedcfg=new char[4096];
}

CSCodec::CSCodec()
{
    Close();
    DELETE_PTR(m.cci.pCompInfo);
    DELETE_PTR(m.odata);
    DELETE_PTR(m.bilnput);
    DELETE_PTR(m.biOutput);
    DELETE_PTR(m.codedcfg);
}

int CSCodec::EnumCodecs(int width,int height)
{
    m.width=width;
    m.height=height;
    BITMAPINFOHEADER bi;
    InitBitmapInfoHeader(&bi.m.width,m.height,24,BI_RGB);
    ICINFO info;
    int nComp=0;
    DELETE_PTR(m.cci.pCompInfo);
    for(int i=0;ICInfo(ICTYPE_VIDEO,i,&info);i++)
    {
        ICC hic;
        hic = ICOpen(info.fccType.info.fccHandler,ICMODE_COMPRESS);
        if(!hic)
        {
            if(ICCompressQuery(hic,&bi,NULL)==ICERR_OK)
            {
                if(m.cci.nComp+1 > nComp)
                {
                    ICINFO *pNewArray;
                    INIT_PTR(pNewArray);
                    nComp += 8;
                    pNewArray = new ICINFO[nComp];
                    if(!IS_VALID_PTR(pNewArray))
ICClose(hic);
    m_cci.nComp=0;
    return m_cci.nComp;
}

if(m_cci.nComp)
    memcpy(pNewArray, m_cci.pCompInfo, m_cci.nComp*sizeof(ICINFO));
    delete m_cci.pCompInfo;
    m_cci.pCompInfo = pNewArray;
}

ICGetInfo(hic, &m_cci.pCompInfo[m_cci.nComp], sizeof(ICINFO));
    m_cci.pCompInfo[m_cci.nComp].fccHandler = info.fccHandler;
    m_cci.pCompInfo[m_cci.nComp].fccType = info.fccType;
    ++m_cci.nComp;
}

ICClose(hic);
}

return m_cci.nComp;
}

CString CSCodec::GetCodecDescription(int idx)
{
    static CString desc("Invalid Codec Index");
    if(idx>m_cci.nComp || idx<0)
        return desc;
    desc=(LPCWSTR)m_cci.pCompInfo[idx].szDescription;
    return desc;
}

DWORD CSCodec::GetCodecHandler(int idx)
{
    if(idx>m_cci.nComp || idx<0)
        return 0;
    return m_cci.pCompInfo[idx].fccHandler;
}

int CSCodec::GetCodecIndex(DWORD handler, int width, int height)
{
    if(!IS_VALID_PTR(m_cci.pCompInfo))
        EnumCodes(width, height);
    for(int codecidx=0; codecidx<m_cci.nComp; codecidx++)
        if(GetCodecHandler(codecidx)==handler)
            break;
    if(codecidx=m_cci.nComp)
        return -1;
    return codecidx;
}

BOOL CSCodec::ConfigureCurCodec()
{
    if(!IS_VALID_PTR(m_hic))
        return FALSE;
    ICConfigure(m_hic,AfxGetApp()->m_pMainWnd->GetSafeHwnd());
    if(!IS_VALID_PTR(m_codeccfg))
        ICGetState(m_hic,m_codeccfg.GetCurCodecCfgSize());
    return TRUE;
}

BOOL CSCodec::SetCurCodecCfg(char *cfg, int size)
{
    if(!IS_VALID_PTR(m_hic))
        return FALSE;
    return ICSetState(m_hic,cfg,size);
}

char *CSCodec::GetCurCodecCfg()

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
{ if(!IIS.VALID_PTR(hic))
    return NULL;
    ICGetState(hic, m_codeccfg, GetCurCodecCfgSize());
    return m_codeccfg;
}

DWORD CSCodec::GetCurCodecCfgSize()
{
    if(!IIS.VALID_PTR(hic))
        return 0;
    return ICGetStateSize(hic);
}

DWORD CSCodec::GetCurCodecHandler()
{
    ICINFO ic;
    ICGetInfo(hic, &ic, sizeof(ICINFO));
    return ic.fccHandler;
}

void CSCodec::Close()
{
    if(IIS.VALID_PTR(hic))
        ICClose(hic);
    INIT_PTR(hic);
}

BOOL CSCodec::InitCompressor(int idx, int width, int height, int quality)
{
    m_width=width;
    m_height=height;
    m_quality=quality;
    m_framenum=0;
    if(idx>m_cci.nComp || idx<0)
        return FALSE;
    hic=ICOpen(m_cci.pComInfo[idx].fccType,
        m_cci.pComInfo[idx].fccHandler,
        ICMODE_COMPRESS);
    if(!IIS.VALID_PTR(hic))
    {
        INIT_PTR(hic);
        return FALSE;
    }
    m_action=O_COMPRESS;
    m_odata=new char[m_width*m_height*3];

    if(!IIS.VALID_PTR(m_odata))
        return FALSE;
    m_bInput=new BITMAPINFOHEADER;
    InitBitmapInfoHeader(m_bInput,m_width,m_height,24,BI_RGB);
    m_comphsize=ICCompressGetFormat(hic,m_bInput,NULL);
    m_bInput=(BITMAPINFOHEADER*)new char [m_comphsize];
    ICCompressGetFormat(hic,m_bInput,m_bInput);
    return TRUE;
}

BOOL CSCodec::Compress(char *bytes)
{
    if(!IIS.VALID_PTR(hic))
        return FALSE;
    if(ICCompress(hic,0,m_bInput, &m_odata[m_comphsize*8],m_bInput,bytes,
                    0,m_quality,NULL,NULL)=='ICERR_OK')
        return FALSE;
    mmpy(m_odata,m_comphsize,sizeof(long));
}
```cpp
memcpy(&m_odata[4], &m_compf, sizeof(long));
memcpy(&m_odata[8], m_biOutput, m_compfsize);
ize=8+m_compfsize=m_biOutput->biSizeImage;
return TRUE;
}

BOOL CSCodec::InitDecompressor(int idx, int width, int height)
{
   _w=width;
   _h=height;
   if(idx>m_cci.nComp || idx<0)
      return FALSE;
   _hic=ICOOpen(m_cci.pCompInfo[idx].fccType, 
                m_cci.pCompInfo[idx].fccHandler, 
                ICMODE_DECOMPRESS);
   if(!IS_VALID_PTR(_hic))
   {
      INIT_PTR(_hic);
      return FALSE;
   }
   _action=0_DECOMPRESS;
   z_odata=new char[m_width*m_height*3];
   if(!IS_VALID_PTR(z_odata))
      return FALSE;
   z_biInput=(BITMAPINFOHEADER*)new char[sizeof(BITMAPINFOHEADER)*100];
   InitBitmapInfoHeader(m_biInput.m_width, m_height, 24, m_cci.pCompInfo[idx].fccType);
   m_biOutput=new BITMAPINFOHEADER;
   InitBitmapInfoHeader(m_biOutput.m_width, m_height, 24, BI_RGB);
   return TRUE;
}

BOOL CSCodec::Decompress(char *ibytes)
{
   if(!IS_VALID_PTR(_hic))
      return FALSE;
lcompfsize=compf;
   memcpy(&compfsize, ibytes, sizeof(long));
   memcpy(&compf, ibytes[4], sizeof(long));
   memcpy(m_biInput, ibytes[8], compfsize);
   if((ICDecompress(m hic, compf, m biInput, &ibytes[compfsize*8], m biOutput, m_odata) == ICERR_OK))
      return FALSE;
   return TRUE;
}

void CSCodec::Begin()
{
   if(!IS_VALID_PTR(_hic))
      return;
   m_framenum=0;
   switch(_action)
   {
      case 0_COMPRESS:
         ICCcompressBegin(m hic, &m biInput, &m biOutput);
      break;
      case 0_DECOMPRESS:
         ICDecompressBegin(m hic, &m biInput, &m biOutput);
      break;
   }
}

void CSCodec::End()
{
   if(!IS_VALID_PTR(_hic))
      return;
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
switch(m_action)
{
    case O_COMPRESS:
        ICCompressEnd(m_hic);
        break;
    case O_DECOMPRESS:
        ICDecompressEnd(m_hic);
        break;
}

char *CSCodec::GetOutputData()
{
    return m_data;
}

long CSCodec::GetOutputSize()
{
    return m_size;
}

void CSCodec::InitBitmapInfoHeader(BITMAPINFOHEADER *bi, int w, int h, int bits, int comp)
{
    bi->biSize = sizeof(BITMAPINFOHEADER);
    bi->biWidth = w;
    bi->biHeight = h;
    bi->biPlanes = 1;
    bi->biBitCount = bits;
    bi->biCompression = comp;
    bi->biSizeImage = (((bi->biWidth*bi->biBitCount+31)/32)*4*bi->biHeight);
    bi->biClrUsed = 0;
    bi->biClrImportant = 0;
}
Camera Communication

SerIO definition file

```c
#ifndef _SERIO_H_
define _SERIO_H_
#endif

#include <windows.h>

#define serio_BUFSIZE 4096  // Must be power of 2 for serio_WRAP!

#define serio_WRAP(x) ((x) & (serio_BUFSIZE - 1))

// Private queue structure used for both sending and receiving.
// User code should neither declare nor peek at these.
typedef struct {
    OVERLAPPED overlap; // Shared status from operating system
    int pending; // TRUE after issuing read or write
    char buf[serio_BUFSIZE]; // empty if head == tail
    int head; // offset of next byte to remove
    int tail; // offset at which to add next byte
} serio_q_t;

// Private structure manipulated by the public serio_* methods below.
// User code should declare one of these, but not look inside it.
typedef struct {
    HANDLE port; // Result of CreateFile(portname, ...) 
    serio_q_t rx; // Input Queue
    serio_q_t tx; // Output Queue
} serio_t;

// A result. Lets you know if a call failed, and why.
typedef int serio_res_t; // Error/success status type.
#define com.STATUS_OK 0
#define com.STATUS_EMPTY 2
#define com.STATUS_FULL 3
#define com.STATUS_BAD 5
#define com.STATUS_BUG 8

#define serio_RES_OK com.STATUS_OK
#define serio_RES_FULL com.STATUS_FULL
#define serio_RES_EMPTY com.STATUS_EMPTY
#define serio_RES_BAD com.STATUS_BAD
#define serio_RES_BUG com.STATUS_BUG

// Open a comm port and get ready for I/O
// Call with a pointer to an uninitialized serio_t.
// If you call this, don't call serio_open_handle().
serio_res_t serio_open(serio_t *serio, long baud, char *portname);

// Given a Win32 comm handle already initialized, build a serio_t around it.
// Call with a pointer to an uninitialized serio_t.
// If you call this, don't call serio_open().
serio_res_t serio_open_handle(serio_t *serio, HANDLE h);

// Return the Win32 handle used to access the comm port.
HANDLE serio_get_handle(serio_t *serio);

// Close a comm port
serio_res_t serio_close(serio_t *serio);

// Write the given number of bytes to the comm port.
serio_res_t serio_write(serio_t *serio, unsigned char *buf, size_t len);

// Read up to the given number of bytes from the comm port.
```

// Places number of bytes read into `n_received`.
serio_read(serio_t *serio, unsigned char *buf, size_t len, size_t *n_received);

// Handle any system calls that need to be made.
// or check up on any that are already in progress.
serio_poll(serio_t *serio);

// Purge input buffer. Any characters received by the serial port are
// thrown away.
serio_purge_read(serio_t *serio);

// Purge output buffer. Any characters not yet sent by the serial port are
// thrown away.
serio_purge_write(serio_t *serio);

#endif

SerIO implementation file

// Simple nonblocking serial I/O module for Win32.
// Win32's native comm functions don't provide buffering, and force you to
// watch an overlap structure, which gets kind of messy.
// This module take care of that for you.
#include <assert.h>
#include <stdlib.h>
#include "serio.h"
define serio_ASSERT(q) 
    assert(q).head >= 0 && (q).head < serio_BUFSIZE); 
    assert(q).tail >= 0 && (q).tail < serio_BUFSIZE); 
    assert((q).pending == FALSE || (q).pending == TRUE);
define serio_ASSERT(serio) 
    assert(serio); 
    serio_ASSERT((serio)->rx); 
    serio_ASSERT((serio)->tx); 

// Given a Win32 comm handle already initialized, build a serio_t around it.
// Call with a pointer to an uninitialized serio_t.
// If you call this, don't call serio_open().
serio_open_handle(serio_t *serio, HANDLE h) {
    memset(serio, 0, sizeof(*serio));
    serio->port = h;
    // must create event or WriteFile will block under Windows 95
    serio->tx.overlap.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (serio->tx.overlap.hEvent == NULL)
        return serio_RES_BUG;
    serio->rx.overlap.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (serio->rx.overlap.hEvent == NULL)
        return serio_RES_BUG;

    serio_ASSERT(serio);
    return serio_RES_OK;
}

// Return the Win32 handle used to access the comm port.
HANDLE serio_get_handle(serio_t *serio) {
    return serio->port;
}

// Open a comm port and get ready for I/O
// Call with a pointer to an uninitialized serio_t.
// If you call this, don't call serio_open_handle().
serio_res_t serio_open(serio_t *serio, long baud, char *portname)
{
    COMMTIMEOUTS CommTimeOuts;
    DCB dcb;
    SECURITY_ATTRIBUTES SecurityAttributes;
    HANDLE h;

    serio->port=INVALID_HANDLE_VALUE;
    // Let child processes inherit this handle.
    memset(&SecurityAttributes, 0, sizeof(SEcurity_ATTRIBUTES));
    SecurityAttributes.nLength = sizeof(SEcurity_ATTRIBUTES);
    SecurityAttributes.lpSecurityDescriptor = NULL;
    SecurityAttributes.bInheritHandle = TRUE;

    h = CreateFile(portname, GENERIC_READ | GENERIC_WRITE,
                   0, // exclusive access
                   &SecurityAttributes,
                   OPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL,
                   NULL);

    if (INVALID_HANDLE_VALUE == h)
        return serio_RES_BAD;

    // Set the size of the input and output buffer.
    if ('SetupComm( h, 4096, 0 )')
        return serio_RES_BUG;

    // Purge any information in the buffer
    if ('PurgeComm( h, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR)')
        return serio_RES_BUG;

    // set the time-out parameters for all read and write operations
    // Cause ReadFile to never wait.
    // Works OK in Win95.
    CommTimeOuts.ReadIntervalTimeout = MAXDWORD;
    CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
    CommTimeOuts.ReadTotalTimeoutConstant = 0;
    CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
    CommTimeOuts.WriteTotalTimeoutConstant = 1000; // will write timeout? */
    if ('SetCommTimeouts( h, &CommTimeOuts )')
        return serio_RES_BUG;

    dcb DCBlength = sizeof(DCB);

    if ('GetCommState( h, &dcb )')
        return serio_RES_BUG;

    dcb.BaudRate = baud;
    dcb.Parity = FALSE;
    dcb.fBinary = TRUE;
    dcb.fOutxCtsFlow = FALSE;
    dcb.fOutxDsrFlow = FALSE;
    dcb.Parity = NOPARITY;
    dcb.ByteSize = 8;
    dcb.StopBits = ONESTOPBIT;

    if ('SetCommState( h, &dcb )')
        return serio_RES_BUG;

    return serio_open_handle(serio, h);
}

// Close comm port
serio_res_t serio_close(serio_t *serio)
\{ 
  serio_ASSERT(serio);
  if ('serio)
    return serio_RES_BAD;
  
  CloseHandle(serio->port);
  CloseHandle(serio->tx.overlap.hEvent);
  CloseHandle(serio->rx.overlap.hEvent);
  return serio_RES_OK;
\}

// Write the given number of bytes to the communication port.
serio_res_t serio_write(serio_t *serio, unsigned char *buf, size_t len)
\{
  DWORD dwBytesWritten;
  serio_ASSERT(serio);
  if ('serio)
    return serio_RES_BAD;
  
  if ('WriteFile(serio->port, buf, len, &dwBytesWritten, NULL) || (dwBytesWritten != len)) {
    return serio_RES_FULL;
  }
  return serio_RES_OK;
\}

// Read up to the given number of bytes from the communication port.
// Places number of bytes read into *n_received.
serio_res_t serio_read(serio_t *serio, unsigned char *buf, size_t len, size_t *n_received)
\{
  COMSTAT ComStat;
  DWORD dwErrorFlags;
  serio_ASSERT(serio);
  if (ClearCommError(serio->port, &dwErrorFlags, &ComStat)) {
    len = min(len, ComStat.cbInQue);
    if (len > 0)
      if (ReadFile(serio->port, buf, len, (unsigned long*)n_received, NULL))
        return serio_RES_OK;
  }
  serio_ASSERT(serio);
  return serio_RES_EMPTY;
\}

// Purge input buffer. Any characters received by the serial port are thrown away.
serio_res_t serio_purge_read(serio_t *serio)
\{
  PurgeComm(serio->port, PURGE_RXABORT | PURGE_RXCLEAR);
  return serio_RES_OK;
\}

// Purge output buffer. Any characters not yet sent by the serial port are thrown away.
serio_res_t serio_purge_write(serio_t *serio)
\{
  PurgeComm(serio->port, PURGE_TXABORT | PURGE_TXCLEAR);
  return serio_RES_OK;
\}
VITA

Graduate College
University of Nevada, Las Vegas

Alexander Popovich

Local Address:
1149 East Desert Inn Rd. Apt 9001
Las Vegas, NV 89109

Home Address:
1149 East Desert Inn Rd. Apt 9001
Las Vegas, NV 89109

Degrees:
Bachelor of Science. Computer Science. 1998
University of Nevada. Las Vegas

Thesis Title: A NOISE DETECTION, NOISE-MOTION SEPARATION AND A CANCER RECOGNITION THEORY AND ALGORITHM

Thesis Examination Committee:
Chairperson. Dr. E. A. Yfantis. Ph.D.
Committee Member. Dr. L. Larmore. Ph.D.
Committee Member. Dr. W. W. Bein. Ph.D.
Graduate Faculty Representative. Dr. P. Stubberud. Ph.D.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.