KAHRAMANMARAŞ SÜTÇÜ İMAM ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKRONİK MÜHENDİSLİĞİ ANABİLİM DALI

BORLAND C++ BUILDER VE TCP/IP İLE GERÇEK ZAMANLI KAMERA GÖRÜNTÜ AKTARIMI

KADİR AKTEPE

YÜKSEK LİSANS TEZİ

KAHRAMANMARAŞ
Eylül - 2005

KAHRAMANMARAŞ SÜTÇÜ İMAM ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKRONİK MÜHENDİSLİĞİ ANABİLİM DALI

BORLAND C++ BUILDER VE TCP/IP İLE GERÇEK ZAMANLI KAMERA GÖRÜNTÜ AKTARIMI

KADİR AKTEPE

YÜKSEK LİSANS TEZİ

Kod No:

Bu Tez 23/09/2005 Tarihinde Aşağıdaki Jüri Üyeleri Tarafından Oy Birliği ile Kabul Edilmiştir.

…….…..……..	……..…….……	….....
Yrd. Doç. Dr. Şaban ERGÜN DANIŞMAN	**Prof. Dr. M. Kemal KIYMIK ÜYE**	**Yrd. Doç. Dr. A. Serdar YILMAZ ÜYE**

Yukarıdaki imzaların adı geçen öğretim üyelerine ait olduğunu onaylarım.

Yrd. Doç. Dr. İrfan Ersin AKINCI
Enstitü Müdürü Vekili

Bu çalışma herhangi bir kuruluş tarafından desteklenmemektedir.
Proje No:

İÇİNDEKİLER

SAYFA

KAHRAMANMARAŞ SÜTÇÜ İMAM ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKRONİK MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

ÖZET

BORLAND C++ BUİLDER VE TCP/IP İLE GERÇEK ZAMANLI KAMERA GÖRÜNTÜ AKTARIMI

KADİR AKTEPE

DANIŞMAN: Yrd. Doç. Dr. Şaban ERGÜN

Yıl: 2005 Sayfa: 79

Jüri : Yrd. Doç. Dr. Şaban ERGÜN
: Prof. Dr. M. Kemal KIYMIK
: Yrd. Doç. Dr. A. Serdar YILMAZ

Bu çalışmada USB web kamerada elde edilen görüntünün server olarak kullanılan bir PC' ye aktarımı sağlanmış; istenilmesi halinde uzak nokta olarak tanımlayabileceğimiz client konumunda bulunan bilgisayarda da aynı görüntünün elde edilerek izlenmesi sağlanmıştır.

Çalışma esnasında hızı ve esnekliği göz önünde bulundurularak Borland C++ Builder yazılım geliştirme aracı olarak kullanılmıştır. Yine uygulama kolaylığı sağlayan Windows API' larından da faydalanılmıştır. Programlama esnasında iletişim hızını artırmak için JPEG resim sıkıştırma formatı kullanılmıştır. Server ile client arasındaki haberleşme TCP/IP protokolü kullanılarak sağlanmıştır.

Tez çalışmasında ilk olarak bir server uygulaması geliştirilmiş, bu server ile USB webcam' dan gerçek zamanlı görüntü elde edilmiştir. İkinci aşamada ise bir client uygulaması geliştirilmiş ve bu uygulama ile de ağ veya internet ortamı kullanılarak TCP/IP ile server' da elde edilen görüntüye erişim sağlanmıştır.

Anahtar kelimeler: Windows API, USB, TCP/IP , JPEG, Görüntü Aktarımı.

UNIVERSITY OF KAHRAMANMARAŞ SÜTÇÜ İMAM
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

MSc THESIS

ABSTRACT

THE REAL TIME CAMERA IMAGE TRANSFER WİTH BORLAND C++ BUİLDER AND TCP/IP

KADİR AKTEPE

Supervisor: Assist. Prof. Dr. Şaban ERGÜN

Year: 2005 Pages: 79

Jury : Assist. Prof. Dr. Şaban ERGÜN
: Prof. Dr. M. Kemal KIYMIK
: Assist. Prof. Dr. A. Serdar YILMAZ

In this study an image, obtained by an USB camera, transferred to a PC which is used as a server; if intended this image can be watched in a computer that is defined as client which is located at a long distance.

During the programming stage; because of the speedily and ability for various application of Borland C++ Builder software had been used. At the same time Windows APIs are also used. In the programming, JPEG picture zipping format is used for the communication speed. TCP/IP Protocol is used between the server and client in order to supply the communication.

In the study of Thesis the application of server was improved firstly, with this server a real time image has been obtained. In the second stage a client application has been improved; and also in this application with the usage of internet backbone there has been an access supplied through the image which is obtained by TCP/IP Protocol.

Key words: Windows API, USB, TCP/IP, JPEG, Image Transfer.

ÖNSÖZ

Bilgi çağı olan günümüzde bu alanda meydana gelen gelişmeler baş döndürücü bir hızla devam etmektedir. Bu bağlamda internet kullanımı ve teknolojisi de daha geniş kitlelere ulaşacak şekilde yaygınlaşmakta ve aynı ölçüde hızı da artmaktadır. Bu hızlı ve yaygın bilgisayar ağının verimli kullanımına yönelik yeni projeler sürekli olarak geliştirilmektedir.

İnternet'in ilk kullanıldığı yıllarda text tabanlı transferler yapılmakta iken daha sonra dosya transfer protokolü (FTP) geliştirildi. Hızın artması ile http protokolü ile daha renkli bir internetin ilk adımları atılmaya başladı.

Dosaya transfer ve HTTP'nin ardından gerçek zamanlı projeler hayata geçirilmeye başlanmış olup, bu alt yapıya dayalı olarak insanların bazı hizmetleri ayaklarına getirme alışkanlıkları lüks durumdan ihtiyaç haline gelmiştir. Bu kapsamda projemde bir kameranın görüntüsünün USB portu ile bağlı olduğu bilgisayarın ekranında gösterilmesi, kameranın bağlı olduğu bilgisayarın server konumunda bulunarak, elde edilen bu görüntüleri istenmesi halinde client konumunda bulunan diğer PC ve laptoplara iletmesi sağlanmıştır. İletim ortamı olarak LAN ve İnternet, haberleşme protokolü olarak ise TCP/IP kullanılmıştır.

Çalışmalarım sırasında bilgi ve tecrübeleriyle bana yardımcı olan danışman hocam Sayın Yrd. Doç. Dr. Şaban ERGÜN'ne sonsuz teşekkür ve şükranlarımı sunarım.

Çalışmamda programlama konusunda bilgi ve tecrübesi ile yardımlarını esirgemeyen Bilgisayar Mühendisi Sayın Kemal ÖNYURT'a ve yine proje derlemem sırasında katkıda bulunan Sayın Mustafa AKSU'ya teşekkürlerimi sunarım.

Tez çalışmalarım sürecinde yanımda olup, beni yalnız bırakmayan eşim Hilal AKTEPE'ye ve yine çalışmalar sırasında yanımda olup, dokümanlarımı dağıtan kızım Emine Şimal AKTEPE'ye sevgilerimi sunarım.

Eylül 2005

KAHRAMANMARAŞ **KADİR AKTEPE**

ÇİZELGELER DİZİNİ

SAYFA

ŞEKİLLER DİZİNİ

SAYFA

SİMGELER VE KISALTMALAR DİZİNİ

3G : Third Generation
ADB : Apple Desktop Bus
API : Application Programming Interface
ARIMA : Auto Regressive Integrated Moving Average
ARPANET : Advanced Research Projects Agency NETwork
BDE : Borland Database Engine
BMP : Bitmap
CCD : Charge-Coupled Device
CMOS : Complementary Metal Oxide Semiconductor
COFF : Common Object File Format
CPU : Central Proses Unit
DF : Don't Fragment
DHCP : Dynamic Host Configuration Protocol
DLL : Dynanmic Link Library
DNS : Domain Name System
E-MAIL : Electronic Mail
EOB : End Of Buffer
EOD : End Of Data
FIFO : First In First Out
FQN : Fully Qualified Name
FTP : File Transfer Protocol
GIF : Graphics Interchange Format
GUI : Graphical User Interface
HTTP : Hypertext Transfer Protocol
IP : Internet Protocol
ISO : International Organization for Standardization
JPEG : Joint Photographics Experts Group
LAN : Local Area Network
LZW : Lempel Zev Welch
MF : More Fragment
MIT : Massachusets İnstitute of Technology
MMM : Multi Media Messaging
MZ : Mark Zibikovski
NE : New Executable
NIR : Near Infrared Color Composite
NT : New Technology
OMF : Object Module Format
OOP : Object Oriented Programming
OSI : Open Systems Interconnection
PC : Personal Computer
PDA : Personal Digital Assistant
PDF : Portable Document Format
PE : Portable Executable
PNG : Portable Netwok Graphic
PPP : Point-to-Point Protocol

PSNR : Peak Signal To Noise Ratio
RAD : Rapid Application Development
RGB : Red Green Blue
RMSE : Root Mean Squared Error
SCSI : Small Computer Standard Interface
SLIP : Serial Line IP
SMP : Switch Mode Proses
SNMP : Simple Network Management Protocol
SOB : Start Of Buffers
TCP : Transmission Control Protokol
TIFF : Tagged Image File Format
USB : Universal Serial Bus
WAN : Wide Area Network
WINS : Windows Internet Naming Service
V86 : Virtual 86
VCL : Visual Component Library

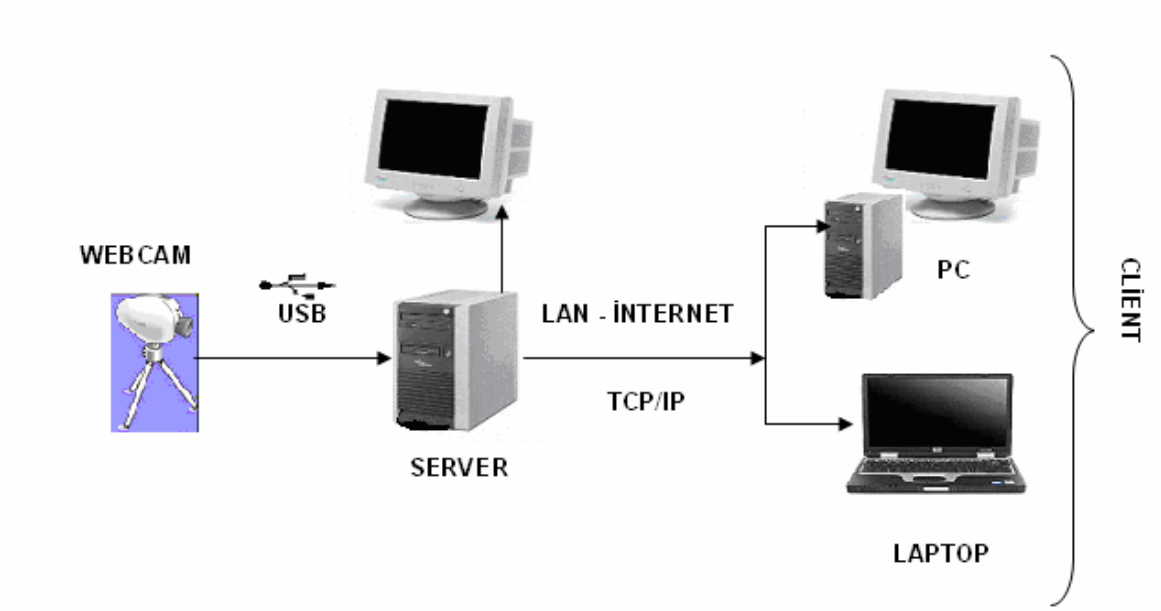
1. GİRİŞ

"Borland C++ Builder ve TCP/IP ile Gerçek Zamanlı Kamera Görüntü Aktarımı" isimli uygulamalı tez çalışmamız iki ana bölümden oluşmaktadır. Uygulamanın birinci bölümü Server, ikinci bölümü ise Client uygulaması olarak planlanmıştır.

Uygulamanın birinci bölümünde elimizde bulunan Web Kamera ile USB Portu üzerinden görüntünün alınması ve alınan bu görüntünün monitör üzerinde izlenmesi hedeflenmektedir. Bu projede yazılım geliştirme aracı olarak Borland C++ Builder kullanılacaktır.

Programın birinci bölümü olan Server kısmının hem yalnız başına kamera izleme amacıyla kullanılabilmesi, hem de istenilmesi halinde Client konumundaki başka bilgisayarlara elde edilen görüntüleri gönderebilecek şekilde çalışması hedeflenmiştir.

Projenin ikinci kısmında ise ayrı bir bilgisayar Client olarak kullanılacak ve yine Borland C++ Builder ortamında geliştirilen bir ekran dizayn edilerek, bu ekran üzerine de Server konumunda bulunan bilgisayardan elde edilen veri aktarılacaktır. Server ile Client arasındaki veri aktarımı için LAN (Local Area Network) veya İnternet kullanılacaktır. Server ile Client arasındaki haberleşme için ise TCP/IP protokolü kullanılacaktır. Proje şematik olarak Şekil 1.1.' de gösterilmiştir.



Şekil 1.1. Uygulama projesinin şematik gösterimi

Proje uygulamasında USB Web Kamera tercih edilmiştir. 1997'nin başından beri çoğu bilgisayarların arkasında birçok kullanıcının ne işe yaradığını bile bilmediği bir ya da iki dikdörtgen şeklinde bağlantı mevcuttu. Bu bağlantıların bilinmemesi aslında gayet normaldi. Zira uzun süre özellikle Türkiye bilgisayar pazarında bu bağlantı noktalarına bağlayabileceğiniz hiçbir cihaz bulunmamaktaydı. "Evrensel seri yol" diye Türkçe de

karşılığını bulan USB, yazıcı, fare, klavye, modem, scanner, joystick, dijital kamera gibi çoğu çevrebirim aygıtı için ortak bir arabirim sunmaktadır. Paralel ve seri arabirimlerin neden olduğu kablo karmaşasının da önüne geçmektedir. USB kamera tercih edilmesi sonucu bu kamera server üzerinden enerji aldığından, daha önceki cihazların yanında görmeye alışılan adaptör kullanılmasına da gerek kalmamaktadır. USB çevrebirim aygıtları bilgisayara, sistemi yeniden başlatmaya veya yapılandırmaya gerek kalmadan bağlanabildiği için ayrı bir kullanım kolaylığı daha sağlamaktadır. Uygulama sırasında Philips PCVC675K Marka USB Web Kamera kullanılmıştır. Günümüzde iki tip kamera sensoru kullanılmakta olup, bu sensorlar kullanılan kameranın tipini belirlemektedirler. Bu sensor çeşitleri CCD ve CMOS' tur. Uygulamada kullanılan Web Kamera CCD sensorludur.

Uygulamamızda, Microsoft Windows XP Professional işletim sistemi, Intel Pentium 4 CPU 3.06 GHz hızı ve 320 Mb RAM kapasitesi bulunan PC kullanılmıştır. İşletim sisteminin Windows olması; aynı anda birden fazla programın çalıştırılabildiği çok işlemli bir ortam yaratması, grafik tabanlı tasarımlara olanak sağlaması, DOS programlarını çalıştıracak şekilde dizayn edilmesi, bir programın, sistemin geneline ve başka programlara zarar vermesini engelleyen bir koruma mekanizması bulunması gibi çoğaltılabilecek faydalar sağlayacaktır.

İşletim sistemleri fonksiyonel bir biçimde yazılmış programlardır. Bazı fonksiyonlar hem işletim sisteminin çalışması sırasında sistem tarafından hem de programcı tarafından kullanılabilmektedir. Windows dünyasında bu fonksiyonlara API (Application Programming Interface) fonksiyonları denilmektedir. Uygulama sırasında Windows API' lerinden de faydalanılacaktır.

Günümüzde çok popüler olan nesneye dayalı programlama dalında Internet ve veritabanı uygulamalarında kendilerinin en iyi yazılım olduğunu iddia eden araçlar bolca mevcuttur. Borland C++ Builder bütün bunların hepsini aynı anda entegre bir biçimde sunabilecek ölçüde bir ürün olduğu için uygulamamız da yazılım geliştirme aracı olarak kullanılacaktır.

Borland C++ Builder diğer hızlı uygulama-geliştirme araçlarının tersine tam anlamıyla esnektir; yani C++Builder ile oluşturamayacağınız uygulama hemen hemen yok gibidir. Yapacağınız uygulama bir veritabanı uygulaması ya da bir ActiveX kontrolü yaratmak olsun kolayca gerçekleştirilebilecektir. Burada esnekliği sağlayan en önemli özellik, C++Builder'ın görsel bileşenlerinin oluşturduğu kullanıcı ara birimin yanısıra, C++Builder'i diğer hızlı uygulama-geliştirme araçlarından farklı kılıp hayat veren tabanındaki C++ dilidir. C++ gerçek anlamda nesneye dayalı bir dildir; yani inheritance, polymorphism ve encapsulation özelliklerini tam olarak barındırır. Bu özelliklerden inheritance sayesinde, hem mevcut bileşenleri geliştirebilir hem de sil baştan yeni bileşenler yaratabilirsiniz. Görsel bileşenler kütüphanesinde (VCL -Visual Component Library) yer alan 120 civarındaki öğe sayesinde, Windows işletim sistemi altında görmeye alıştığımız her türlü uygulamayı hazırlamak çok kolay bir hale gelmektedir. Hatta gerektiğinde C, Assembly ve Pascal ile yazılmış olan kaynak kodları kullanabilmeye izin vermesi, Windows API' sine direkt komutlar gönderebilmesi C++' ın gücüne destek

vermektedir. Hazırda bulunan DLL (Dynanmic Link Library), ActiveX ve bunun gibi diğer Windows nesnelerini kendi programlarınızda kullanabilir ve isterseniz kendiniz de bunlardan oluşturabilirsiniz.

Uygulama sırasında resim formatlarından faydalanılacaktır. JPEG formatı, grafik veri ve resimleri dijital iletim ortamlarında iletmek için dizayn edilen ve genellikle tam renk gerçek resimleri tutmak için kullanılan bir formattır. Buna ilave olarak JPEG formatı fotoğraflar ve çok renkli resimlerin network üzerinden aktarımı için idealdir.

Proje uygulamasında server ile client arsındaki haberleşme için TCP/IP Protokolü kullanılacaktır. Bilgisayar ve network' ün internet üzerinde bilgi ve mesaj paylaşımı yapmalarını sağlayan düşünce aslında basittir. Her mesaj ve bilgi, paket (packet) denilen ufak parçalara ayrılır, bu paketler doğru yere ulaştırılır ve paketler yerine ulaştıktan sonra yeniden birleştirilerek alıcı bilgisayarların kullanabileceği ve kullanıcıya sunabileceği şekilde orijinal haline gelir. Bu işleri yapmak, internet üzerindeki en önemli iki iletişim protokolü olan Transmissiol Control Protocol(TCP) ve İnternet Protocol(IP)'ün görevidir. TCP paketlere ayırma ve yeniden birleştirme işini yaparken, IP paketlerin doğru hedefe gitmelerini sağlamakla yükümlüdür.

TCP/IP protokolünün, IBM, 3Com, DEC, Sun, HP ve benzeri firmaların büyük bir kısmı tarafından benimsenmesi, her türlü bilgisayar ortamında rahatlıkla çalışması(PC, Server, Mainframe gibi), Unix ortamına çok iyi entegrasyon sağlaması, istemci – server mimarisi, Ethernet, X.25 ve Token Ring gibi birçok yerel ve genel ağ protokollerini desteklemesi gibi bir çok özelliğe sahip olması tercih sebeplerinden bazılarıdır.

2. ÖNCEKİ ÇALIŞMALAR

Çalışma konumuzla ilgili olan internet ve TCP/IP hakkında hızlı bir gelişme söz konusu olup, bu sürece ait tarihi gelişmeler ve yine tez konumuzla ilgili olarak kamera görüntü işleme üzerine bugüne kadar yapılan çalışmalar aşağıdaki bölümlerde ifade edilmeye çalışılmıştır.

2.1. İnternet ve TCP/IP'nin Tarihçesi

İnternet binlerce bilgisayar ağı, milyonlarca kullanıcının bilgisayar kaynaklarını ve bilgi paylaşımını sağlayan uluslar arası bir bilgisayar haberleşme ağıdır. Kelime anlamı olarak da "Uluslararası Çalışma Ağı" anlamına gelir. Internet sayesinde kullanıcılar, bilgisayarda bir dosya formatında depolanabilen herhangi bir veriyi paylaşabilirler. İnternete bağlı tüm ağların ve bilgisayarların birbirleriyle haberleşirken aynı dili konuşabilmesi veya çevirmen kullanması gerekir. Bu dil farklı ağlarda bulunan farklı tipteki makinelerin iletişimini ve bilgiyi paylaşmasını sağlayan programlardır.

Internet; verilere, resimlere, seslere, yazılımlara, metne, mültimedyaya ve kişilere erişim sağlarken iletişim ve veri aktarımı için de araçlar sunar (Anonim, 2000).

Şekil 2.1. Şematik İnternet Ağı (Anonim, 2000).

1960'larda iletim kontrol protokolü (TCP- Transmission Control Protokol) ve internet protokolü (IP-Internet Protocol) ağa bağlı iki cihaz arasında yüksek hızlı iletişim kurabilmek amacıyla geliştirildi. Bu ağ protokolleri cihazlar arasındaki bağlantılardan bazıları kopsa bile iletişimin sürekliliğini hedefliyordu. RAND Corporation, MIT(Massachusets İnstitute of Technology) ve University of California bu teknolojiyi Amerika Birleşik Devletleri Savunma Bakanlığı için geliştirdi. ABD Savunma Bakanlığı bir nükleer savaş sonrasında iletişimin sürekliliğini sağlayabilecek bir bilgisayar ağına ihtiyaç duyuyordu.

1969'da ABD Savunma Bakanlığı deneysel bir bilgisayar ağı olan ARPANET (Advanced Research Projects Agency NETwork)'i kullanmaya başladı. Bu ilk ağ tabanlı protokol teknolojisiydi. ARPANET ilk önce dört bilgisayarı birbirine bağladı.

TCP/IP ilk olarak 1970'li yılların başında Amerikan Savunma Bakanlığı'na bağlı İleri Araştırma Ajansı'nın (ARPA) savunma amaçlı projelere destek vermek üzere paket anahtarlamalı ağ deneylerinde kullanılmaya başlanmıştır. ARPA, geliştirdiği paket anahtarlamalı deneysel çalışmaları ilk olarak WAN (Wide Area Network)'lara uygulamaya başlamış, daha sonra geliştirilen bu sistemi LAN (Local Area Network) sistemlerine göre de adapte etmiştir.

1980'li yılların başlarında Berkeley'in UNIX 4.2 versiyonu da tam olarak TCP/IP protokolünü içermeye başladı. Sonuç olarak bu protokol kümesinin kullanılma alanının artışı diğer network sistemlerinde de kullanılmasına sebep oldu. 1983 yıllarında ise TCP/IP protokolü artık bütün yerel ve genel ağlarla birlikte askeri çalışmalarında standardını oluşturmuştur.

Yapılan çalışmalar sonucu TCP/IP protokolü 1990 yıllarının sonunda her türlü işletim sistemi ve ağ ortamına uyumlu olması ve OSI (Open Systems Interconnection) standartları ile etkileşimli olması nedeniyle çok fazla kullanılır hale gelmiştir. Bu nedenle Internet bağlantıları için TCP/IP ve bileşenleri kaçınılmaz hale gelmiştir.

2.2. Görüntü İşleme İle İlgili Çalışmalar

Akyol (1999), " Sayısal Görüntü İşlemede Görüntü Karşılaştırma" konulu tezinde görüntü karşılaştırmasının yapılabilmesi için sayısal görüntü işlemeden ne şekilde yararlanılabileceğini bu konuda olabildiğince çeşitli kaynaklardan yararlanarak araştırdığını ifade etmiştir. Bu çalışmasında görüntü karşılaştırmak için kullanıldığı söylenebilecek belirgin, tek ve basit bir teknik olmadığı gözlemlenmiştir. Görüntüleri analiz ederek noktasal tanımını yapmanın, görüntüleri karşılaştırmanın başlangıcı olabileceği araştırılmış, bunun için kullanılabilecek Houg Transform ve bununla ilişkili kontur yönelimli segmantasyonun prosedürleri incelenmiştir. Bu prosedürlerle elde edilen sonuçlar Ad Oculos Programında gözlenmiştir. Ayrıca ilk görüntünün piksel piksel aynı yada farklı olduğu noktaların tarifini yaparak ve görüntü üzerinde matematiksel ve mantıksal işlemler uygulayarak görüntüleri karşılaştırma üzerinde çalışılmış, bu şekildeki karşılaştırmayı yapan Pascal Programı yazılmıştır. Piksel piksel karşılaştırmanın sonuçlarını yorumlamak için, RMSE (Root Mean Squared Error) ve PSNR (Peak Signal To Noise Ratio) ölçütleri kullanılmıştır.

Erol (1999), " Registration of Digital Images " adlı tezinde değişik zamanlarda uzaktan çekilen resimlerin analizinde resimlerin çakıştırılması gerektiğini belirtmiş ve bu işlemi gerçekleştirmek için resimlerden birisini referans olarak almış ve diğer resimleri de giriş resimleri olarak kullanılmıştır. Bu çalışmada referans resimden giriş resime dönüşümde kullanılan kontrol noktalarını ve özel bölgeleri ortaya çıkaran yeni geliştirilmiş tam otomatik bir algoritma ve kontrol noktalarının eşlenmesinde şablon eşleme için en yaygın olarak kullanılan benzerlik ölçütleri kullanılmıştır. Örtüşmeyen nokta da olan kontrol nokta çiftleri yeni bir algoritma ile setten çıkarılmış ve kontrol noktaları arasındaki dönüşüm ikinci dereceden denklemlerle modellenmiştir. Denklemin sabitleri ise en küçük hata kareleri yöntemi kullanılarak bulunmuştur. Tez kapsamında aynı manzaraya ait resim setini çakıştırarak işleyip büyük bir resim oluşturan bir Mozaik Program'ı Borland C++ Builder kullanılarak yazılmıştır.

Gümüşer (2001), "Digital Görüntü İşleme" konulu tezinde digital görüntü işleme konusuna giriş niteliğinde bir çalışma yapmıştır. Çalışmanın amacı, görüntü işlemede kullanılan kavramları tanıtmak ve daha ileri düzeydeki uygulamalara bir temel oluşturmaktır. Bunun için önce işlenecek görüntünün bilgisayarda gösterimi ve saklanmasıyla ilgili konular ayrıntılı olarak anlatılmıştır. Daha sonra ise hem insanın kullanımına, hem de bilgisayarın kullanımına yönelik görüntü geliştirme uygulamaları

tartışılmış ve pratikte kullanılan çeşitli yöntemler verilmiştir. Yine genellikle ikilik görüntülerin segmantasyonu aşamasında kullanılan temel morfolojik yöntemler tanıtılmış, çeşitli tekniklerin geliştirilmesi ve uygulanması sonucunda ortaya çıkan durumlar değerlendirilmiştir. Digital görüntü işleme tekniklerinin bilgisayar uygulamalarında hızlı ve verimli çalışma özellikleri göz önüne alınarak, C ve C++ bilgisayar programlama dilleri kullanılmıştır.

Yaman ve ark.(2001), "Dinamik çizelgeleme için görüntü işleme ve ARIMA modelleri yardımıyla veri hazırlama" adlı çalışmada Ankara Hızlı Raylı Sistemde, Kızılay-Ankaray istasyonunda bekleyen yolcuların, sistemde güvenlik amaçlı kullanılan kameralar vasıtasıyla algılanan gri seviye görüntüleri, bilgisayar ortamına aktarılmıştır. Daha sonra, görüntü segmentasyon işlemleri ile nesneler arka plandan ayrılmış ve ayrılan nesnelere ait görüntüler, görüntü güçlendirme metotları ile belirginleştirilmiştir. Bir sonraki aşamada, netleştirilmiş görüntülerin gri-seviye histogramlarından nesnelere ait alansal bilgiler çıkarıldı. Hesaplanan yolcu yoğunluk oranı değerleri ile gözle sayılan yolcu sayıları arasındaki ilişkiler incelenerek Ankara Hızlı Raylı Ulaşım Sistemde tren sefer aralıklarının optimizasyon işlemlerine giriş verileri sağlanacak hale getirildi. Elde edilen bu sayısal değerler, zaman serisi verileri olarak alınıp hafta içi yolcu gelişlerinin ARIMA(ARIMA-Auto Regressive Integrated Moving Average) modelleri yardımıyla modellemesi yapıldı. Sonuçta bu model kullanılarak söz konusu hizmet sisteminde dinamik çizelgeleme sürecine veri hazırlanması sağlandı.

Alparslan ve ark. (2004), TUBITAK-MAM Yer ve Deniz Bilimleri Araştırma Enstitüsü'nde "Uçakla Digital Fotoğraf Çekimi Çalışmaları" adlı araştırmalarında görüntü işleme tekniklerini de kullanmışlardır.Uçakla dijital fotoğraf çekimi teknolojisi diğer uzaktan veri toplama teknolojilerine farklı olarak basit ve etkili bir teknoloji olup hızlı, bağımsız ve uygun maliyetli haritalandırma veya harita güncelleme sağlar. Gerekli olan malzeme ise dijital fotoğraf makinesi, GPS ünitesi ve sayısal görüntülerin kaydedilmesi ve işlenmesi için özel yazılımdan oluşan entegre veri üretme ve kaydetme sistemidir. Fotoğraflar Cessna veya benzeri tercihen yukarı kanatlı küçük bir uçağın altına yerleştirilmiş bilgisayar kontrollü bir dijital fotoğraf makinesi ve paralel olarak işlem gerçekleştiren GPS cihazı ile elde edilebilir. Fotoğraf çekimi esnasında GPS ünitesi ile fotoğrafların koordinatlandırılması böylece harita olarak kullanılması sağlanır. Keşif uçuşu sona erdiğinde temelde sayısal görüntüler, belirli alanların mozaik görüntüsü yanı sıra coğrafik koordinatlandırılmış ve düzeltilmiş harita ürünleri elde edilir. Bu veriler özellikle izleme ve kontrol amaçlı uygulamalarda, afet sonrası hızlı veri toplamada, kentleşmenin ve kaçak yapılaşmanın takibinde, arazi kullanım değişimi tespiti vb. uygulamalarda kullanılabilir.

Ünsal ve ark.(2004), "Landsat 7TM Uydu Verileri Kullanılarak Dedegöl Dağı ve Çevresinin Çizgisellik ve Jeolojik Özelliklerinin Belirlenmesi" adlı çalışmalarında Landsat TM 7 4.3.1 band kombinasyonundan oluşan NIR (Near Infrared Color Composite)) uydu görüntüsü kullanmışlardır.. Bu görüntü üzerinde koordinat dönüşümleri, kontrast düzeltmeleri, mekansal filtrelemeler gibi görüntü işleme teknikleri kullanılmıştır. Çalışmada, öncelikle Landsat TM 7 uydu görüntüsünü gerçek dünya koordinatlarına dönüştürebilmek için 1/25.000 ölçekli topografik haritalardan yer kontrol noktaları belirlenerek koordinat dönüşümleri yapılmıştır. Daha sonra koordinat dönüşüm-

leri yapılan görüntü üzerinde renk zıtlığı artırılmış, mekansal filtrelemeler yapılarak sınrları keskinleştirilmiştir. Görüntü üzerinde Edge Enhance 5x5 ve Sharpen 5x5 filtreleri kullanılmıştır. Tüm görüntü işleme teknikleri ile gözlemsel yorumlamalar (renk, drenaj sistemleri, bitki örtüsü, morfoloji v.b.) kullanılarak çalışma alanındaki jeolojik birimler ayırtlanarak haritalanmıştır.

3. MATERYAL VE METOT

Projemizin uygulama ve geliştirme süresince bazı donanım, yazılım, işletim sistemi, iletim ortamı ve haberleşme protokollerine ihtiyaç duyulmuştur. Donanım olarak PC, Laptop ve USB Web Kamera kullanılmıştır. Yazılım geliştirme aracı olarak ise internet ve veritabanı uygulamalarında çok iyi, hızlı ve esnek olan Borland C++ Builder tercih edilmiştir. İşletim sistemi için Windows, iletim sistemi için ağ ve internet, haberleşme protokolü olarak da TCP/IP kullanılmıştır. Ayrıca resim işleme formatlarından faydalanılmış ve bunlarla ilgili olarak bilgi verilmiştir.

3.1. Materyal

3.1.1. PC (Personel Computer)

PC kişisel bilgisayar anlamına gelen Personel Computer isminden kısaltılmıştır. Piyasaya girdiği ilk senelerde, bilgisayarlar büyük şirketlerin, üniversitelerin ve laboratuarların tekelindeydi. Ancak IBM şirketinin uygulaması sonucu, bilgisayar herkesin satın almaya gücünün yetebileceği bir alet oldu. İlk PC, IBM şirketi tarafından piyasaya sürüldü. Bu bilgisayarın ticari başarı sağlayarak tüm dünyaya yayılması sonucu diğer şirketlerde IBM PC ile neredeyse tıpatıp aynı özellikte makineler ürettiler. Böylece IBM uyumlu bilgisayarlar ortaya çıktı. PC'leri teknik olarak incelerken, onları birbirlerinden ayıran özelliklerdir şunlardır: Merkezi işlem ünitesi (CPU), bilgisayarın tüm işlemlerini kontrol eden birimdir. PC'leri markalarına göre ayırmak yerine işlemcilerine göre ayırmak daha yerinde olur. İlk PC'ler 8086 adı verilen bir CPU taşıyorlardı. Daha sonra bu model geliştirilerek günümüzdeki Pentium (İntel), Athlon(AMD) gibi işlemciler ortaya çıkmıştır. Bu işlemcilerin özellikleri 8 bitlik (8 bit=1 Byte, temel hafıza birimi) bilgilerle işlem yapmasıydı. Burada dikkat edilmesi gereken bir konu daha var. Pentium ve Athlon işlemciler piyasaya girene kadar üretilen 286 CPU'lardan sonra CPU'lar 16 bitlik ve 32 bitlik mimariye sahip olarak üretildi. Ama bunlar sadece görünüşte böyleydiler. Hala 8 bitliktiler ama çarpan teorisi ile günümüze kadar geldiler. Anlatılmak istenen ilk PC makinelerde çalışan programların en son teknoloji ile üretilen makinelerde bile çalışması için CPU'lar hala içlerinde 8 bitlik işlem yapmaktadır. Bu durumda performans devreye girdiği zaman işler karışmaktadır. CPU'lar 1 GHz hızların üstüne çıktı ama hala eski mimari sistemi ile çalışmaktadırlar.

Şekil 3.1. PC ve Laptop (Anonim, 2005e).

Bilgisayar kullanımının artmasıyla, insanlar her an bilgisayarda işlem yapma ihtiyacı duymaya başladılar. Teknolojinin de gelişmesiyle birlikte Laptop(Diz üstü bilgisayar)

denilen cihazlar hayatımıza girdi. İlk başlarda büyük ve ağır olan bu bilgisayarlar, zamanla küçülüp hafiflemektedirler (Anonim, 2005e).

3.1.2. USB Web Kamera

Bu konu USB ve görüntü algılayıcıları olarak iki başlık altında anlatılacaktır:

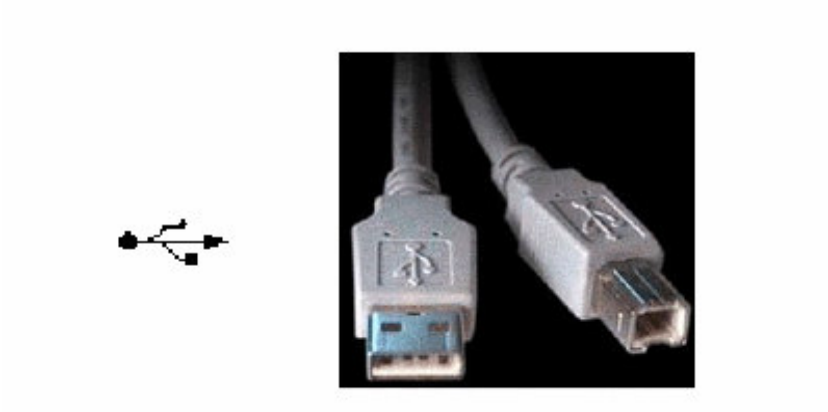
3.1.2.1 . USB (Universal Serial Bus- Evrensel Seri Veri Yolu)

USB, bilgisayarınıza çeşitli aygıtları takmanıza olanak tanıyan bir çevre birimi veri yolu standardıdır. Macintosh bilgisayarların çoğu, çevre birimi aygıtları bağlamak için SCSI (Small Computer Standard Interface), ADB (Apple Desktop Bus) ve seri bağlayıcılar kullanır. USB, SCSI ve ADB gibi standartların yerini alacaktır.

USB standardını başlangıçta, Compaq, Digital Equipment, IBM, Intel, Microsoft, NEC ve Northern Telecom geliştirmiştir. Başlangıçtaki yedi şirketin 1995 yılında oluşturduğu USB Uygulayıcıları Forumu'nun (USBIF; www.usb.org), ortak misyonu USB çevrebirimlerinin gelişimini sağlamak ve tüketici tarafından benimsenmesini artırmaktır ve 500'den fazla üyesi bulunmaktadır.

1997'nin başından beri çoğu bilgisayarların arkasında birçok kullanıcının ne işe yaradığını bile bilmediği bir ya da iki dikdörtgen şeklinde bağlantı yuvası bulunmaktaydı. Buna şaşmamak gerek, zira uzunca bir süre, özellikle Türkiye bilgisayar pazarında bu bağlantı noktalarına bağlayabileceğiniz hiçbir çevre birimi bulamıyordunuz.

Bu bağlantı noktalarının ardında 1995 yılından beri bilgisayar dünyasının devleri tarafından geliştirilen yeni bir ara birim gizli: Universal Serial Bus, kısaca USB, Türkçe ifadesiyle "evrensel seri yol". Bu arabirim; yazıcı, fare, klavye, modem, scanner, joystick, dijital kamera gibi çoğu çevrebirim aygıtı için ortak bir arabirim sunarken, paralel ve seri arabirimlerin neden olduğu kablo karmaşasının da önüne geçmektedir. USB arabiriminin saniyede ki 1 Mbyte'lık hızı, çoğu çevrebirim aygıtı için yeterli olmaktadır.



Şekil3.2. USB'nin sembolik gösterimi ve konektörleri. Sol taraftaki konektör bilgisayara, sağ taraftaki ise çevre birimlerine takılır. Her iki konektörde 4 uçludur. İkisi voltaj, ikisi de data içindir(Anonim, 2005b).

USB'nin temel amacı, standart bağlayıcıları kullanarak ve eklenti kartlarından kurtularak, geliştiriciler ve tüketiciler için maliyetleri düşürmektir. Bunun nedeni, veriler gibi gücün de (güç dediğimiz şey, elektriktir) USB kabloları aracılığıyla dağıtılması ile bazı düşük güçlü aygıtların ayrıca kullanılan adaptörlerden kurtulmasını sağlamaktır.

USB uyumlu göbekleri (hub'ları) kullanarak , bir PC'ye 127 aygıtın takılması sağlanabilmektedir. USB, masaüstünde, kablo bağlantısını 5 metreden az tutarak, orta dereceli bir veri aktarımını amaçlamaktadır.

USB çevrebirim aygıtlarını bilgisayarınıza, sistemi yeniden başlatmaya veya yapılandırmaya gerek kalmadan bağlayabilirsiniz. Bu aygıtlar aynı zamanda birbirlerine zincirleme şeklinde de bağlanabiliyor. PC'nizde ikiden fazla USB aygıtını çalıştırmak isterseniz bir USB dağıtıcısına, yani USB Hub'a ihtiyacınız olacaktır. Bu tür bir bağlantı yapısı Şekil 3.3.'de görülmektedir.

Şekil 3.3. USB Hub Ve Bağlantısı (Anonim, 2005b).

Çoğu klavye ve monitör üreticileri ürünlerini bu tip USB Hub'ları ile donatmaktadırlar. Bu sayede bilgisayarınız çalışırken USB aygıtlarını istediğiniz gibi bağlayabilirsiniz. Windows otomatik olarak hangi USB aygıtının söz konusu olduğunu algılıyor ve bu aygıta uygun sürücüleri kuruyor.

Çoğu USB aygıtın "sabit" veya bağlı USB kablosu vardır. Bilgisayarınıza bir USB aygıt bağlamak için, aygıtın USB kablosunu bilgisayarınızdaki USB kapısına takabilirsiniz. Ne kadar sıklıkta olursa olsun istediğiniz zaman bilgisayarı kapatmak, yeniden başlatmak veya uyutmak zorunda kalmadan USB aygıtı bağlayabilir yada bağlantısını kesebilirsiniz.

USB en hızlı büyüyen üç alanda çok önemli rol oynuyor: dijital görüntüleme, PC uzak iletişimi (PC telephony), ve çoklu ortam oyunları. USB' nin varlığı, bu alanlarda PC'lerin ve yan donanımların güvenilir olarak bir arada çalışmaları anlamına gelmektedir. USB, giriş aygıtları için yenilikler kapsını açıyor. Örnek olarak yeni nesil "force-feedback"

dijital joystikleri gösterebiliriz. Bunların yanı sıra yazıcılardan tarayıcılara, yüksek hızda iletişime (Ethernet, DSL, ISDN veya uydu iletişimi gibi) uyumlu bütün yan donanımlar için de yepyeni imkanlar sunuyor.

3.1.2.2. Görüntü Algılayıcıları

Son yıllarda sayısal fotoğraf makinesi ve sayısal kameralı mobil telefonların yaygınlaşması ile beraber, optik görüntüleme algılayıcılarına olan talep aşırı bir biçimde artmağa başlamıştır. Önümüzdeki bir kaç yıl içinde optik görüntü algılayıcılarının otomotiv alanında, yol ve sürücü sensoru olarak kullanımında büyük bir patlama olması ve optik band görüntü algılayıcıları için cep telefonu pazarının birkaç misli boyutta yeni bir pazarı doğurması beklenmektedir. Optik band' ta görüntü algılayıcılarının potansiyel bir uygulama alanı da özellikle tıbbi tanı ve tedavi amacı ile vücut-içi görüntü alma amacı ile kullanılacak olan mikro-sistemlerdir (Leblebici ve ark., 2004).

Görüntü algılayıcıları Optik Band, İnfrared ve Mikro görüntü algılayıcıları olarak üç guruba ayrılır. Bu kısımda Optik Band görüntü algılayıcıları hakkında bilgi verilecektir. Optik (Görünür) band' ta çalışan görüntü algılayıcıları, insan gözünün algıladığı dalga boylarında çalışmaları nedeni ile en yaygın kullanım alanı olan görüntü algılayıcı sınıfını oluşturmaktadır.

Son yıllarda sayısal fotoğraf makinesi talebinde görülen hızlı artışın yanı sıra Çoklu Ortam Mesajlaşma (Multi Media Messaging, MMM) özellikli, PC ve kameralı cep telefonlarının yaygınlaşması ile beraber, optik görüntüleme algılayıcılarına olan talep aşırı bir biçimde artmağa başlamıştır.

Tarihsel olarak geçen yüzyılın ilk yarısında Plumbicon, Vidicon gibi elektron tüpü teknolojileri kullanılarak gerçekleştirilen optik band görüntü algılayıcıları, son yirmi yıldır yarı iletken teknolojisi kullanılarak gerçekleştirilmektedir.

Günümüzde optik band görüntü algılayıcıları için iki farklı yarı iletken teknolojisi kullanılmaktadır. Bunlar CCD (Charge Coupled Device) ve CMOS (Complementary Metal Oxide Semiconductor). Yaklaşık yirmi yıllık bir geçmişi olan CCD teknolojisi, günümüzde artık olgunluğa ulaşmış ve en yüksek çözünürlüklü, en düşük gürültülü görüntü algılayıcılarını üretmek için kullanılan baskın teknolojidir.

Henüz birkaç yıllık geçmişi olan CMOS görüntü algılayıcı teknolojisi ise hızlı bir gelişme göstermektedir. CMOS görüntü algılayıcılarının bir kaç yıl içinde performans seviyesi bakımından CCD algılayıcıları yakalaması beklenmektedir. CMOS görüntü algılayıcı teknolojisinin CCD teknolojisine göre en önemli avantajı, görüntü algılayıcı ile sayısal görüntü işleme birimini aynı kırmık üzerinde tümleştirebilme olanağını tanımasıdır. Karma (analog/sayısal) CMOS teknolojisi kullanarak üretilmiş, tek kırmık üzerinde işlemcisi ile bütünleşik görüntü algılayıcı birimlerinin sağlayacağı boyut ve maliyet avantajlarının, görüntü algılayıcılar için yepyeni uygulama alanları açması beklenmektedir.

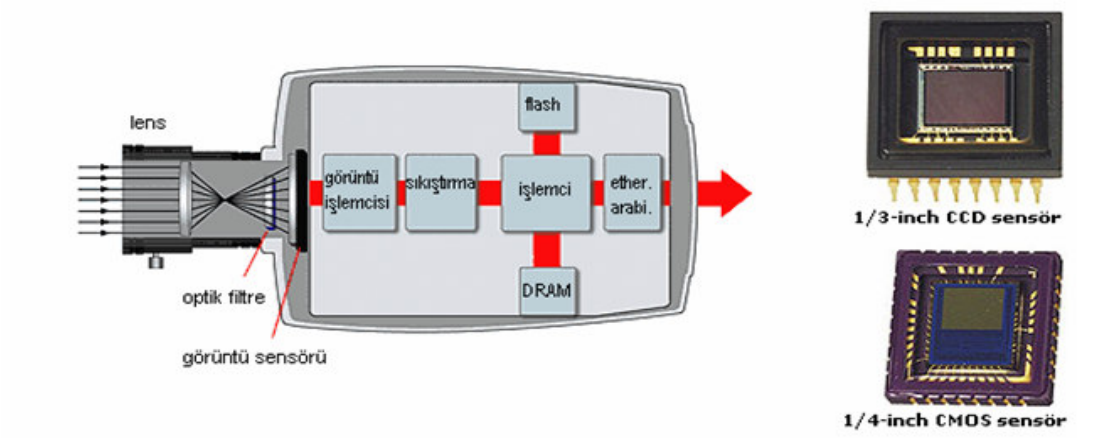
CCD ve CMOS görüntü algılayıcılarının üretimi tamamen tüm devre üretimi teknolojileri kullanılarak gerçekleştirilmektedir. Geleceğin baskın görüntü algılayıcı

teknolojisi olarak görebileceğimiz CMOS görüntü algılayıcıları, günümüzde 0,35 mikron CMOS teknolojisi ile üretilmektedir. Birkaç yıl içinde, yüksek çözünürlüklü ve düşük gürültülü CMOS görüntü algılayıcıları için (sayısal tüm devre üretiminde olgun bir teknoloji sayılan) 0.18 mikron teknolojisi kullanımı öngörülmektedir.

CMOS görüntü algılayıcı ve tümleşik algılayıcı/işlemci tüm devreleri konusundaki dünya lideri firmalara baktığımızda bunların bir bölümünün büyük kapasiteli tüm devre üreticileri olduğunu görmekteyiz. Ancak, bu alanda yenilikçi ürünleri çıkaran bazı önde gelen firmaların da sadece görüntü algılayıcı ve tüm devre tasarımı üzerine uzmanlaştıkları, bu firmaların üretimlerini iş ortaklığı yaptıkları bazı büyük tüm devre üreticilerinin fabrikalarında gerçekleştirdikleri görülmektedir.

Özellikle görüntü algılama ve görüntü işleme işlevlerini aynı kırmık üzerinde, standart CMOS teknolojisi ile üretilebilecek biçimde bütünleştiren yenilikçi ürünler, büyük sermaye yatırımına gerek duymadan ve sadece görüntü algılayıcı ve işaret işleme tasarımı teknolojilerine yatırım yaparak gerçekleştirilebilir(Leblebici ve ark., 2004).

Şekil 3.4.'de bir nesnenin, optik lenslerden başlayarak bilgisayarın Ethernet girişine kadar geçirmiş olduğu aşama şematik olarak ifade edilmektedir.



Şekil.3.4. Web kameranın blok diyagramı görüntüsü ile CCD ve CMOS sensorlar (Anonim, 2005a).

Şekil 3.5. Bazı Marka WebCam Çeşitleri(Anonim, 2004).

Çizelge 3.1. Bazı PC kamera üreticileri (Anonim, 2004).

- 3COM PC Kamera
- A4 Tech PC Kamera
- Akita PC Kamera
- Apache PC Kamera
- Codegen PC Kamera
- Creative PC Kamera
- D-Link PC Kamera
- Genius PC Kamera
- Genx PC Kamera
- Inf PC Kamera
- Intel PC Kamera
- Jaguar PC Kamera
- King PC Kamera
- Labtec PC Kamera
- LifeView PC Kamera
- Logitech PC Kamera
- Micro PC Kamera
- Minton PC Kamera
- Mustek PC Kamera
- Orite PC Kamera
- Philips PC Kamera
- Quattro PC Kamera
- Trust PC Kamera
- Unibrain Kamera
- Zoltrix PC Kamera

3.1.3. Windows İşletim Sistemi

Windows'un ilk versiyonu 1985 yılında çıkmıştır. Bu 1.0 versiyonu çok kısıtlı ölçüde kullanılmıştır. Daha sonra 1987 yılında 2.0, 1988 yılında da 3.0 versiyonları piyasaya sürülmüştür. O yıllarda Microsoft'un DOS işletim sistemi aktif olarak kullanılıyordu. Windows, DOS altında çalışan bir utility program gibiydi. Windows API programlamanın temelleri ağırlıklı olarak 3.0 versiyonu ile oluşturulmuştur. Fakat Windows'un en fazla kullanılan versiyonu 80'li yılların sonlarına doğru piyasaya sürülmüş olan 3.1 versiyonudur. Windows 3.1 de bir işletim sistemi değildi. Fakat 286 ve 386 işlemcilerde korumalı moda geçerek çalışıyordu. Windows 3.1, temel olarak DOS'u kullanan fakat DOS'un olanaklarını korumalı mod yoluyla arttıran, işletim sistemi denilmese bile işletim sistemi olmaya aday bir programdı. Windows 3.1, 16 bitlik bir çalışma sunuyordu.

Microsoft, 1992-93 yıllarında Windows NT (New Technology) adı altında ilk 32 bit gerçek Windows işletim sistemini piyasaya sürmüştür. Windows NT, DOS uyumu zayıf olan bir işletim sistemiydi. Oysa o yıllarda DOS programları oldukça yaygın bir biçimde kullanılıyordu. DOS programlarını çalıştıramayan yani DOS uyumluluğu yüksek olmayan işletim sistemlerinin tutulması zor gözüküyordu. Bu nedenle Microsoft, Windows'un DOS uyumu yüksek yeni bir 32 bitlik versiyonunu 95'te piyasaya sürmüştür. Windows 95 DOS'u neredeyse tamamen destekleyen 32 bit bir işletim sistemidir. Windows 95'ten sonra 98 yılında Windows 98 ve 2000 yılında Windows ME ve 2000 sistemleri çıkmıştır. Bunu yaygın son sürümü olan XP izlemiştir.

Windows sistemleri 16 bit ve 32 bit sistemler olmak üzere 2'ye ayrılır. Bugün 16 bit Windows sistemleri tamamen kullanımdan kalkmıştır. Win16 sistemleri Windows 3.x sistemleridir. Win32 sistemleri ise Windows NT, Windows 95, Windows 98, Windows ME, Windows 2000 ve Windows XP sistemleridir.

Win32 sistemleri koruma mekanizmasının katılığına göre ve çekirdek yapısına göre 95 grubu sistemler ve NT grubu sistemler olmak üzere 2'ye ayrılır.

95 grubu sistemler Windows 95, 98 ve ME sistemleridir. Bu sistemlerin DOS uyumu çok yüksektir yani DOS programlarını tamamen çalıştırabilirler. Bu sistemlerde katı bir koruma uygulanmamıştır. Oysa NT grubu sistemlerin DOS uyumu zayıftır ve bu sistemlerde katı bir koruma uygulanmıştır. Genel olarak NT grubu sistemlerin 95 grubu sistemlere göre daha kararlı olduğu söylenebilir.

API programlama modeli Win32 sistemlerin hepsinde büyük ölçüde aynıdır. Fakat bazı sistem özellikleri 95 grubu sistemlerde olmadığı halde NT grubu sistemlerde söz konusu olmaktadır.

WIN32 sistemlerinin temel özellikleri aşağıda sıralanmıştır:

3.1.3.1. Çok İşlemlilik

Win32 sistemleri aynı anda birden fazla programın çalıştırılabildiği çok işlemli sistemlerdir. Çok işlemli sistemlerde prosesler parçalı olarak zaman paylaşımlı bir biçimde çalıştırılmaktadır. Yani bir proses bir süre çalıştırılır, sonra çalışmasına ara verilir, başka bir proses çalıştırılır. Sonra sıra ilk prosese geldiğinde çalışma kalınan yerden devam eder. Bir prosesin parçalı çalıştırılma süresine "quanta süresi" denilmektedir. Win32 sistemlerinde quanta süresi tipik olarak 20ms'dir.

Çok işlemli sistemlerde programın iki noktası arasında geçen gerçek zaman, sistemin o anki durumuna göre farklı olabilir. Kabaca sistemde ne kadar çok proses çalışıyor durumda ise programımızın çalışması yavaşlamaktadır.

Bazı makinelerde birden fazla işlemci bulunabilir. Bu mimariye SMP (Switch Mode Proses) denilmektedir. Bu durumda prosesleri işlemciler paylaşarak daha hızlı çalıştırırlar. Windows 95, 98 ve ME versiyonları çok işlemcili sistemleri desteklememektedir. Ama NT, 2000 ve XP; çok işlemcili bilgisayarlarda çalışabilmektedir.

3.1.3.2. Grafik Tabanlı Çalışma

Windows sistemleri çekirdekle entegre edilmiş grafik bir ara birime sahiptir. Yani işletim sistemi bir pencere sistemi göz önünde bulundurularak tasarlanmıştır. Oysa örneğin UNIX/LINUX sistemlerinde grafik çalışma pencere yöneticileri denilen çekirdekle entegre edilmemiş programlar tarafından sağlanmaktadır. Genel olarak Windows sistemlerinin grafik işlemleri UNIX/LINUX sistemlerinin grafik işlemlerinden daha hızlıdır.

3.1.3.3 GUI ve Console Uygulamaları

GUI (Graphical User Interface) uygulaması, pencereye sahip olan gerçek Windows uygulamasıdır. GUI uygulamalarında mesaj tabanlı bir programlama modeli kullanılmaktadır. Console uygulamaları ise siyah ekranda çalışan klasik programlardır. Yani console uygulamaları için yine DOS'ta ve UNIX/LINUX sistemlerindeki programlama modeli kullanılır. Console uygulamaları görünüşü ve ara birimi önemli olmayan fakat hızlı çalışması istenen programlarda tercih edilir.

3.1.3.4. DOS Uyumu

Win32 sistemleri Intel mimarisinde DOS programlarını çalıştıracak biçimde tasarlanmışlardır. Windows 95, 98 ve ME sistemleri hemen hemen tüm DOS programlarını çalıştırırken Windows NT, 2000 ve XP sistemleri yalnızca bazı programları çalıştırabilmektedir.

Windows 98 sisteminde bir DOS programın çalıştırdığımızı düşünelim. Bu program da zaman paylaşımlı olarak diğer Windows programları ile birlikte çalıştırılacaktır. Fakat çalışma zamanı DOS programına geldiğinde mikroişlemcinin çalışma modu geçici süre korumalı moddan V86 (Virtual 86) moduna geçirilir. V86 modu, Pentium işlemcilerinin 8086 işlemcisi gibi çalıştığı bir moddur. Bilindiği gibi 8086 işlemcisi 1MB belleği kullanabilen 16 bit bir işlemcidir. DOS programları, 8086 mikroişlemcisi ile çalışabilecek yapıya sahiptir. Yani eğer mikroişlemcinin modu geçici bir süre V86'ya geçemeseydi DOS programları ile Windows programları birlikte çalışamazdı. Bu sistemlerde farklı DOS programları adeta aynı anda farklı DOS makinelerinde çalışıyor gibi bir yapı oluşmaktadır.

En çok karıştırılan konu DOS prompt'u kavramı ile ilgilidir. 95 grubu sistemlerde DOS prompt'una geçtiğimizde DOS işletim sistemi, işlemcinin modunu değiştirerek ayrı bir proses biçiminde çalıştırmaktadır. Yani bu sistemlerde DOS prompt'u bir DOS programıdır. Halbuki NT grubu sistemlerinin prompt'unun DOS ile ilgisi yoktur.

DOS prompt'unda bir Windows console programı çalıştırırsak ne olur? Windows içerisindeki DOS .exe formatına bakar; bu bir Windows console programı ise yeni bir console ekranı açmadan işlemcinin modunu değiştirerek programı çalıştırır. Yani özetle 95 grubu sistemlerin DOS prompt'unda Windows programlarını da çalıştırabiliriz.

3.1.3.5. Çalışabilir ve Amaç(Object) Dosya Formatları

DOS'ta kullanılan .exe formatına "MZ" formatı denilmektedir (MZ, bu formatı tasarlayan Mark Zibikovski'den yapılmış bir kısaltmadır). Windows 3.1 sistemlerine geçildiğinde NE (New Executable) denilen başka bir format kullanılmaya başlamıştır. Gerçekten de Win16.exe dosyalarının başı NE harfleriyle başlamaktadır. Daha sonra Win32 sistemlerine geçildiğinde .exe dosya formatı da yenilenerek PE (Portable Executable) formatına geçilmiştir. Bir PE formatı isteğe bağlı olarak küçük bir MZ programına da sahip olabilir.

DOS'ta kullanılan .obj dosya formatına OMF (Object Module Format) deniliyordu. Win16 sistemlerinde de 16 bit OMF formatı kullanılmıştır. Fakat Win32 sistemlerine geçildiğinde .obj dosya formatı değiştirilerek COFF (Common Object File Format) formatına geçilmiştir. COFF formatının orijini UNIX sistemleridir. Win32 formatı UNIX COFF formatından oldukça farklıdır.

3.1.3.6. Mesaj Tabanlı Programlama Modeli

Klasik console tabanlı uygulamalarda fare kullanımı yaygın değildir ve girdi

oluşturan pek az olay vardır. Programcı bu sistemlerde bir olayın gerçekleşip gerçekleşmediğini bir fonksiyonu çağırarak tespit eder. Olay gerçekleşmişse uygun işlemleri yapar. Örneğin hem fareyi hem klavyeyi izleyebilmek için bir döngü içerisinde hiç beklemeden bu olayların gerçekleşip geçekleşmediğine bakmak gerekir. Bu da çok yorucu bir işlem gerektirmektedir.

Mesaj tabanlı programlama modelinde programcı girdi olaylarını kendisi izlemez. Bu olaylar ilk elden işletim sistemi tarafından alınır ve hangi programa ilişkinse o programa mesaj olarak verilir. Mesaj, teknik olarak bir yapıdır. Örneğin yapının bir elemanında girdi olayının ne olduğu; başka bir elemanında, o olaya ilişkin bilgiler bulunmaktadır. Olay oldukça sistem, mesajı programcıya nasıl iletmektedir? Sistemin yaptığı tek şey mesajı bir FIFO (First In First Out) kuyruk sistemine eklemektir. Programcı, bir döngü içerisinde bu kuyruk sistemine bakmalı; eğer bir olay oluşmuşsa o olayın bilgilerini kuyruk sisteminden alarak işleme sokmalıdır. Yani program bütün yaşamını kuyruktan mesajı alan ve onu işleyen bir döngüde geçirir. Bu döngüye mesaj döngüsü denilmektedir.

3.1.3.7. API Fonksiyonları

İşletim sistemleri fonksiyonel bir biçimde yazılmış programlardır. Bazı fonksiyonlar hem işletim sisteminin çalışması sırasında sistem tarafından hem de programcı tarafından kullanılabilmektedir. Böyle fonksiyonlara UNIX dünyasında sistem fonksiyonları; Windows dünyasında ise API (Application Programming Interface) fonksiyonları denilmektedir. API, bir işlemler kümesi ve üzerinde işlem yapılabilen veri yapılarının kurallarını da belirler. Benzer şekilde programcılar kendi istedikleri fonksiyon kümeleri için API'ler yaratabilirler ve bu API fonksiyonları yazılım geliştirme araçlarında ve derleyicilerde yeni oluşturulan program kodlarına eklenerek kullanılabilir.API fonksiyonları, işletim sisteminin bir parçası olarak bulunmaktadır.

Win32 sistemlerinde temelde 4 grup API fonksiyonu vardır:

1) Kernel API Fonksiyonları: Bu fonksiyonlar KERNEL32.DLL dosyası içerisindedir ve sistemin çekirdek kısmı ile ilgili işlemlere yöneliktir.

2) User API Fonksiyonları: Bu fonksiyonlar USER32.DLL dosyası içerisindedir ve temel olarak pencere işlemleri ile ilgili fonksiyonlardır.

3) GDI API Fonksiyonları: Bu fonksiyonlar GDI32.DLL içerisindedir ve grafik işlemleri ile ilgilidir.

4) Diğer API Fonksiyonları: Yukarıdakilerin dışında çeşitli değişik konularda birtakım sistem DLL'leri içerisinde bulunan fonksiyonlardır.

3.1.3.8. Bellek Yönetimi

Win32 sistemleri sanal bellek (virtual memory) kullanabilen sistemlerdir. Sanal bellek RAM'in yetmediği durumlarda programın disk ile RAM arasında yer değiştirmeli bir biçimde çalıştırılmasına yönelik bir mekanizmadır. Programcı programını sanki tek

parça belleğe yüklenecekmiş gibi yazar. Sanal bellek, işletim sisteminin kendisinin arka planda sağladığı bir mekanizmadır.

Win32 sistemlerinde proseslerin bellek alanları birbirinden tamamen izole edilmiştir. Her proses, çalışma sırası kendisine geldiğinde 4GB'lik belleğin düşük anlamlı 2GB'lik alanına yüklenerek çalıştırılır. (2GB user, 2GB system)

Yüksek anlamlı 2GB'lik alan, işletim sisteminin bulunduğu sistem alanıdır. Bu alan prosesler arası geçişten etkilenmez. Görüldüğü gibi bir proses çalışırken diğer prosesler bellekte değildir. Her proses 2GB'ye kadar büyüyebilir. Örneğin bir prosesin 2GB'lik alanında bir adres bölgesi düşünelim. Başka bir prosesin aynı bölgesinde başka değerler bulunacaktır. Yani biz, bir gösterici hatası yaparak başka bir prosesin bellek alanına erişemeyiz. Win32 sistemlerinde tüm göstericiler 4 byte uzunluğundadır. Bu sistemler Flat model kullandıkları için segment kavramına gereksinim duymazlar. 4 byte' lık göstericiler ile 4GB'lik bellek alanının her yerine erişilebilmektedir.

Win32 sistemlerinde doğal olarak heap alanı(Önemli kaynakların depolandığı özel hafıza alanı) da çok geniştir. Örneğin programcı Malloc fonksiyonuyla 100'lerce MegaByte'lık alanı tahsis edebilir. Bu sistemlerde ciddi bir bellek sıkıntısı yaşanmamaktadır.

3.1.3.9. Koruma Mekanizması

Win32 sistemleri Intel işlemcilerinin korumalı modunda çalışırlar. Koruma mekanizması bir programın, sistemin geneline ve başka programlara zarar vermesini engelleyen bir mekanizmadır.

Koruma mekanizması temel olarak mikroişlemci tarafından sağlanan bir mekanizmadır. Yani bir koruma ihlali olduğunda bu daha ilk elden mikroişlemci tarafından tespit edilir ve işletim sistemine bildirilir.

Koruma mekanizması aşağıda belirtilen konularda bir koruma faaliyeti oluşturmaktadır:

1) Bellek Koruması: Bir proses kendi bellek alanı dışına erişmeye çalışırsa sonlandırılır.

2) I/O Port Koruması: Örneğin proses IRQ kesmelerini kapatmak istesin. Bunun için 21H portuna OUT işlemi yaptığı anda proses sonlandırılacaktır.

3) Makine Kodu Koruması: Bazı makine komutları sistemin tamamen çökmesine neden olabilmektedir. İşte koruma mekanizması altında bu makine komutlarının da kullanımı tamamen yasaklanmıştır.

Intel işlemcilerinde bir prosese 0 ile 3 arasında bir öncelik derecesi verilmektedir. Gerek Win32 sistemleri gerekse UNIX/LINUX sistemleri yalnızca 0 ve 3 öncelik derecelerini kullanırlar. 0 öncelik derecesinde çalışan bir proses hiçbir koruma engeline takılmaz, her şeyi yapabilir. 3 öncelikli prosesler ise çalıştırdığımız sıradan programlardır.

Bunlar, yukarıda sözü edilen koruma duvarlarına çarparlar. Önceliği 3 olan bir prosesin (ring 3'te çalışan bir prosesin) bir API fonksiyonunu çağırdığını düşünelim. Bu fonksiyon sistemin kendi içerisindeki birtakım dataların üzerinde güncelleme yapabilir. Bu durumda koruma hatası ortaya çıkmaz mı? Intel işlemcilerinde Kapı (Gate) denilen bir mekanizma vardır. Bir fonksiyon kapı yoluyla çağırılırsa otomatik olarak fonksiyon sonlanana kadar prosesin önceliği 0'a (ring 0'a) çekilir. Görüldüğü gibi sistem fonksiyonlarının çalışması süresinde proses ring 0'da kalmaktadır. Kapılar, işletim sistemi tarafından güvenli fonksiyonlara yerleştirilmiştir.

Bazen çeşitli gerekçelerle birtakım özel işlemlerin yapılması gerekebilir. Örneğin bir program bir kartın üzerindeki işlemcileri programlamak isteyebilir. İşte ring 0'da çalışacak olan programlar, aygıt sürücü (device driver) denilen özel bir formatta yazılmalıdır. Aygıt sürücüler adeta işletim sisteminin bir parçasıymış gibi yüklenerek çalışırlar.

3.1.3.10. Birden Fazla Thread İle Çalışma

Win32 sistemleri aynı zamanda çok thread ile çalışmaya olanak sağlayan sistemlerdir. Thread, bir programın, sanki başka bir programmış gibi çizelgelenen akışıdır. Bir proses, ana bir thread ile çalışmaya başlatılır. Bu C'deki main fonksiyonunu çalıştıran thread'dir. Diğer thread'ler bir thread akışı içerisinde CreateThread API fonksiyonu ile herhangi bir zaman yaratılabilir. CreateThread fonksiyonunda thread'i yaratırken akışın başlatılacağı fonksiyon adresi verilir. Artık o thread de sanki ana thread'miş gibi diğerinden bağımsız olarak çalışır.

Thread'ler prosesin aynı data bölgesini kullanır. Yani tüm Thread'ler statik ömürlü nesneleri ortak olarak kullanmaktadır. Fakat her thread ayrı bir stack'e sahiptir. Yani thread'lerin stack'leri birbirinden ayrılmıştır. Bu durumda örneğin iki thread aynı fonksiyon üzerinde ilerlese fonksiyonun yerel değişkenlerinin farklı kopyalarını kullanır. Yani yerel değişkenler birbirine karışmazlar.

Win32 sistemlerinin proses yönetimi thread'lere göre çizelgeleme yapmaktadır. Yani her quanta geçişinde bir thread bırakılır, başka bir thread çalıştırılır. Thread'ler arası geçiş aynı iki prosesin thread'leri arasında olabileceği gibi farklı proseslerin thread'leri arasında da olabilir. Thread'ler aynı prosesin bellek alanını kullandığına göre aynı prosesin thread'leri arasındaki geçişte belleğin düşük anlamlı 2GB'lik bölümü değişmez. Fakat farklı prosesin thread'leri arasındaki geçişte bellek üzerinde de geçiş yapılmaktadır.

Thread'ler özellikle arka plan işlemlerin gerçekleştirilmesinde kullanılmaktadır. Örneğin bir program çalışırken aynı zamanda saati ekranda gösterecek olsun. Saatin gösterilmesi için akışın hiçbir yerde takılmaması gerekir. Bu tür işlemleri thread'siz işlemlerde yapmak mümkün olsa bile çok zordur. Oysa thread'li sistemlerde programcı bir thread yaratır, saatin gösterilmesi işlemini o thread'e yaptırır. Artık gerçek thread, bir yerde beklese bile problem oluşmaz.

3.1.3.11. Unicode Kullanımı

Unicode, her bir karakterin 2 Byte ile ifade edildiği yeni bir karakter tablolama

sistemidir. Bir metin, unicode olarak yazılırsa metin içerisinde binlerce farklı karakter kullanılabilir. Unicode karakter sisteminde tüm ülkelerin kullandığı karakterler ve birimlerde kullanılan semboller, tabloya çakışmayacak biçimde yerleştirilmiştir. Tabi, metnin unicode yazılmış olması, bunun pürüzsüz bir biçimde görüntüleneceği anlamına gelmez. Görüntüleme işlemi ayrı bir konudur. Görüntüleyecek olan program, karakteri anlamlandırabildiği halde görüntülemeyebilir.

Win32 sistemleri, unicode çalışmayı destekleyen sistemlerdir. Sistemin API fonksiyonları çeşitli yazıları ASCII yada unicode olarak alabilmektedir. Bu konu ile ilgili daha detaylı bilgi www.unicode.org adresinden elde edilebilir(Aslan, 2005).

3.1.4. Borland C++Builder

Gün geçmiyorki, bilişim pazarında yeni bir yazılım geliştirme aracı çıkmasın; öyle ki bunların hepsi de en hızlı şekilde, en hatasız işleyen programları ürettiklerini iddia etmektedirler. Özellikle günümüzde çok popüler olan nesneye dayalı programlama dalında Internet ve veritabanı uygulamalarında kendilerinin en iyi yazılım olduğunu iddia eden araçlar bolca mevcuttur. Borland C++ Builder bütün bunların hepsini aynı anda entegre bir biçimde sunabilecek ölçüde bir üründür.

Diğer RAD (Rapid Application Development) hızlı uygulama-geliştirme araçlarının tersine, C++Builder tam anlamıyla esnektir; yani C++Builder ile oluşturamayacağınız uygulama bulunmamaktadır. İster bir veritabanı uygulaması, ister bir ActiveX kontrolü yaratmak olsun, C++Builder sayesinde bunların üstesinden en kısa zamanda gelmek mümkündür. Elbette ki, C++Builder yaratmayı planladığınız programı otomatik olarak oluşturmayacaktır; sadece size bunu oluşturmak için gerekli platformu hazırlayacak ve sizin emirlerinizi uygulayacaktır. Burada esnekliği sağlayan en önemli özellik, C++Builder'ın görsel bileşenlerinin oluşturduğu kullanıcı ara birimin yanısıra, C++Builder'i diğer RAD araçlarından farklı kılıp hayat veren tabanındaki C++ dilidir. C++ gerçek anlamda nesneye dayalı (OOP-Object Oriented Programming) bir dildir; yani inheritance, polymorphism ve encapsulation özelliklerini tam olarak barındırır. Bu özelliklerden inheritance sayesinde, hem mevcut bileşenleri geliştirebilir hem de sil baştan yeni bileşenler yaratabilirsiniz. Görsel bileşenler kütüphanesinde (VCL -Visual Component Library) yer alan 120 civarındaki öğe sayesinde, Windows işletim sistemi altında görmeye alıştığımız her türlü uygulamayı hazırlamak çok kolay bir hale gelmektedir. Hatta gerektiğinde C, Assembly ve Pascal ile yazılmış olan kaynak kodları kullanabilmeye izin vermesi, Windows API'sine direkt komutlar gönderebilmesi C++'ın gücüne destek vermektedir. Hazırda bulunan DLL (Dynanmic Link Library), ActiveX ve bunun gibi diğer Windows nesnelerini kendi programlarınızda kullanabilir, gerektiğinde kendiniz de bunlardan yaratabilirsiniz. MFC, ve OWL framework desteğiyle hem bileşenleri hem de nesneleri kullanabilir, DOS altında çalışacak konsol uygulamalar yazabilir, C++Builder'ın ayrı yumurta ikizi olan Delphi'de oluşturulmuş formları, unit'leri ve component'leri kullanabilirsiniz. Tüm bu sayılanları yaparken ANSI C++, Windows 95, Windows NT, Win32 API, ActiveX, OLE Automation, ODBC, DCOM, MAPI, Unicode WinSock, ISAPI ve NSAPI standartlarına bağlı kalırsınız. Günümüzde Internet'in çok popüler olması, program geliştirme araçlarını "Internet uyumlu" olmaya zorladı; böylece bu araçlar kendilerine sürekli eklemeler yaptılar. C++ Builder' da Internet uygulamaları

geliştirmek için bulunan komponentler, sonradan "Internet uyumlu" olma kaygısı içinde eklenmemiştir. Bunlar zaten C++Builder 1.0'da entegre bir şekilde mevcuttu ve 3.0 ile yeni ilaveler yapıldı. Barındırdığı 17 protokol bileşeni sayesinde Internet'e erişim sağlayabilecek programlar (ağ tarayıcıları gibi) ve diğer bileşenler sayesinde, Internet üzerinde veritabanı uygulamaları yazabilirsiniz. ActiveForms ve Web Deployment ile ağ üzerinde client/server uygulamaları geliştirebilir, Web Server Extension ile herhangi bir ağ tarayıcısına veriyi HTML olarak gönderebilir, WebBridge yardımıyla NSAPI, ISAPI DLL'leri ve WinCGI, CGI uygulamaları oluşturabilirsiniz. C++Builder sonuçta bir HTML editörü değildir ama zengin Internet desteği ile çok karmaşık olan ağ üzerinde veritabanı uygulamalarının bile kolaylıkla üstesinden gelmenizi sağlayacaktır. Veritabanı uygulamaları için hazırlanmış olan gerek veriye erişim, gerek verinin görsel olarak sunulmasını sağlayan bir çok öğe yine işinizi büyük ölçüde kolaylaştırmaktadır. Bu uygulamalarda BDE (Borland Database Engine) sayesinde Paradox, Oracle,Sybase, dBASE, InterBase DB2 ve diğer birçok veri tabanına, ODBC üzerinden Access ,FoxPro gibi veritabanlarına bağlanabilirsiniz. QuickReport ögeleri sayesinde güzel görünümlü raporlar yaratabilirsiniz. C++Builder'ın Client/Server Suite sürümünü ile network üzerinde karmaşık veritabanları, son teknoloji "multi-tier" uygulamalar yaratabilirsiniz.

C++Builder'ı kullanarak yapabilecekleriniz, tek başına "Neden C++Builder?" sorusunun cevabını vermeye yeterli değildir. Bunları yaparken hızlı ve "bug"sız yani hatasız bir şekilde olması da çok önemlidir. C++ derleyicisi, en hızlı derleyicilerden biri olup, Incremental Linker ve bir çok optimizasyon sayesinde, büyük projeleri derlerken bile yalnızca gereken yerleri derleyip sürat kazanmanızı sağlar. Bu hızın yanında programın doğru derlenmesi de çok önemlidir. Şurası bir gerçektir ki, hatasız olarak derlenen bir C++ kodu çok büyük bir ihtimalle çalışır. Mantık hataları yapılmış ise programınız istediğiniz gibi çalışmayacaktır. Elbette bir program yazılırken bu tür hatalar sık sık yapıldığı için, "Debugger" (hata ayıklamaya yarayan bir mekanizma) olması gerekir. C++Builder da hataları kısa sürede bulmanıza yarayan bir çok yardımcı araç geliştirilmiştir. Visual Form Designer, Component Palette ve Object Inspector'u kullanarak dilediğiniz görünümdeki formları kısa sürede yaratabilirsiniz.

Ayrıca bu yarattığınız formları Object Repository'ye kaydedip tekrar tekrar kullanabilirsiniz. Böylece her defasında "benzer özellikleri" taşıyan formları baştan yaratmak zorunda kalmazsınız (inheritance sayesinde kaydetmiş olduğunuz formun eski özelliklerine eklemeler yapabilir, fonksiyonlarında değişiklikler yapabilirsiniz). Başta Database Form Wizard olmak üzere; sihirbazlar sayesinde zahmetli işleri bir kaç soruya cevap vererek çok kısa bir zamanda yapabilir, hatta. kendi Wizard'larınızı bile yaratabilirsiniz.

C++Builder'da da diğer Borland ürünlerinde olduğu gibi programcının ihtiyaç duyabileceği her şey düşünülmüş, bunun için bazı yardımcı araçlar da programcının kullanımına sunulmuştur. Bunlardan Image Editor, programınızın ikonunu , Speed Button'larda ve başka yerlerde kullanılan resimleri yaratabileceğiniz/değiştirebileceğiniz bir araçtır. Neredeyse her Windows programında görmeye alıştığımız yardım dosyalarını hazırlayabilmek için Microsoft Help WorkShop, ve veritabanı uygulamalarında table yaratma, içeriklerini izleme, alias yaratma gibi işlevleri gerçekleştiren Database Desktop, Database Explorer, Interbase Client ve Database Administrator gibi programlar pakete

dahil edilmiştir. Ayrıca programcının yapmış olduğu programı kullanıcıya en doğru biçimde yüklenmesini sağlamak için InstallShield Express'te bu yardımcı araçlar arasına eklenmiş (Anonim, 2005c).

3.1.5. Resim Formatları

Aşağıdakilerin tamamıda resim formatlarıdır. Kodlama, sıkıştırma ve algoritmalar gibi farklı temel özellikleri vardır.

3.1.5.1. BMP(Bitmap)

Microsoft Windows ortamına özgü bitmap dosya formatıdır. Özellikler mapped veya unmapped RGB resimleri 1-,4-, 8-, veya 24-bits olarak kaydeder. Veri işlenmemiş halde veya 4-bit, 8-bit RLE data sıkıştırma algoritması kullanılarak depolanabilir. BMP, basit ve yaygın olarak desteklenen bir RGB resim formatıdır, pratik olarak tüm raster resim depolamada kullanılabilir. Mevcut BMP, 32 bit kalite seviyesine kadar destekler. BMP'nin en önemli avantajı kullanılabilirliği ve geniş uygulama desteğidir.

3.1.5.2. GIF (Graphics Interchange Format)

GIF formatı LZW-encoded bitmap veri iletim protokolünü tanımlamak için geliştirilmiş veri akış tabanlı resim formatıdır. GIF resimler her zaman sıkıştırılmış ve 8 bite (256 renk) kadar derinliği olabilir. GIF 8 bit renk desteklemesine rağmen GIF hala düşük çözünürlükte resim saklamak için populer bir araçtır. GIF dosya formatı, resimleri index renk modunda tutmak ve iletmek için kullanılır. Format kayıpsız LZW sıkıştırma formatı, tek bir dosya bir resimler kümesini içerebilir ve aynı zamanda interlaced modu destekler. 1989'da format güncellenmiş ve transparent modu ve animasyonu da destekler hale gelmiştir. Günümüzde en popüler formatlardan biridir. 256 renkten fazla desteklemediği için gerçek kaliteyi yakalayamamasına rağmen, en çok animasyon ve harmanlama olmayan resimleri göstermek için kullanılır.

GIF formatı, JPEG formatından farklı olarak kayıpsız sıkıştırma algoritması kullanır. Bunun anlamı sıkıştırma/açma esnasında herhangi bir biti kaybolmaz. GIF patentli LZW(Lempel Zev Welch) sıkıştırma/açma algoritması kullanır. Interlaced özelliği resim yüklenirken yenileme imkânı sağlar. Büyük dosyalar için çok uygundur. Resmin tamamını yükleninceye kadar görebilirsiniz.

3.1.5.3. JPEG (Joint Photographics Experts Group)

JPEG formatı grafik veri ve resimleri dijital iletim ortamlarında iletmek için dizayn edilen ve genellikler tam renk gerçek resimleri tutmak için kullanılan bir formattır. JPEG'den önce 24 bit yarım tone resmi destekleyen çok az format vardı. TIFF ve BMP formatları 24 bit veriyi destekler fakat binlerce rengi bünyesinde tutan yüksek kalite seviyesinde kayıpsız sıkıştırma işlemini gerçekleştiremez.

JPEG fotoğrafları kalite kaybıyla sıkıştırır. Sıkıştırma algoritması birleştirme için veri silinmesiyle gerçekleşir. Veri piksel blokları olarak belli bir rengin yoğunluk bilgisini

kaydeder. İnsan gözü yoğunluk kaybını çok az fark eder. Fotoğraflar ve çok renkli resimlerde bu format network üzerinden aktarmak için idealdir. JPEG resimleri yenileme imkânı yoktur, fakat dosya büyülüğünü düşürerek görüntü kalitesini artırmak mümkündür. JPEG formatı gerçek görüntüyü yakalamak için yüksek miktarda renk içeren resimler için kullanılır.

Sıkıştırma miktarı resim içeriğine göre değişir. Sıkıştırma oranı 25:1' e kadar kalitede görünür bir kayıp olmadan gerçekleşebilir.

JPEG, GIF formatındaki interlaced standardına benzeyen Progressive JEG standardını destekler, bu şekilde JPEG resimlerde görüntülenirken parça parça yüklenebilir, fakat bu yeni bir standardı ve bazı programları desteklemeyebilir.

3.1.5.4. PDF (Portable Document Format)

PDF ise güvenli ve güvenilir elektronik doküman ve form dağıtımı ve değişimi için defakto bir standarttır. PDF oluşturulduğu ortam veya uygulama ne olursa olsun, fontlar, resimler, grafikler, formlar veya herhangi bir doküman kaynağını saklayan evrensel bir dosya formatıdır. Adobe® PDF dosyaları basit ve komple bir formattır, ücretsiz Adobe Reader® programı ile paylaşılabilir, görüntülenebilir veya çıktı alınabilir.

3.1.5.5. PNG (Portable Netwok Graphic)

PNG dosya formatı, GIF formatının yerini tutabilecek yeni bir resim formatıdır. GIF formatının zayıf olduğu kısımlarda PNG bazı yeniliklere sahiptir.

PNG deflate algoritmasıyla kayıpsız sıkıştırma işlemi yapabilir ve interlaced modu destekler. PNG 256 geçirgen seviye tutabilir, bu şekilde resim parçalı olarak geçirgen olabilir. PNG formatı 48 bit renk derinliğini destekler ve kayıpsız sıkıştırma yapabilir, gerçek görüntüde resim tutabilir. Sıkıştırma ve açma esnasında resmin kalitesinde herhangi bir bozulma olmaz. PNG formatı daha çok ağ ortamı için kullanılır.

3.1.5.6. TIFF (Tagged Image File Format)

TIFF formatı 1986'da çıkmıştır ve genellikle taranmış siyah-beyaz resimleri saklamak için kullanılır. 1987'de karmaşık RGB renkleri destekler hale getirildi. 1988'de palet renkleri ve LZW sıkıştırma algoritmasını destekleyen TIFF 5.0 formatı çıkarıldı. 1992 de ise CMYK ve YcbCr renkleri ile JPEG sıkıştırma algoritmasını destekleyen TIFF 6.0 çıkarıldı.

Şu anda TIFF(TIFF 6.0) zengin ve esnek bir formattır ve çoğu program destekler. Format farklı piksel derinlikleriyle yarım tone resimleri kaydetmeyi desteklemektedir. Bunun için grafik veri depolamada ve işlemede ideal bir formattır.

TIFF formatı en yaygın kullanılan çok amaçlı ve entegre bir raster formattır. Kolayca platformlar arası geçiş sağlayabilir ve internet haricinde birçok amaçla kullanılabilir (Anonim, 2005d).

3.1.6. TCP/IP

Dünyanın her yerindeki bilgisayar ve networkün internet üzerinde bilgi ve mesaj paylaşımı yapmalarını sağlayan düşünce aslında basittir. Her mesaj veya bilgi, paket (packet) denilen ufak parçalara ayrılır, bu paketler doğru yere ulaştırılır ve paketler yerine ulaştıktan sonra yeniden birleştirilerek alıcı bilgisayarların kullanabileceği veya sunabileceği şekilde orijinal haline gelir. Bu işleri yapmak, internet üzerindeki en önemli iki iletişim protokolü olan Transmissiol Control Protocol(TCP) ve İnternet Protocol(IP)'ün görevidir. Genelde bunlara TCP/IP denir. TCP paketlere ayırma ve yeniden birleştirme işini yaparken, IP paketlerin doğru hedefe gitmelerini sağlamakla yükümlüdür. Paket anahtarlamalı (packet-switched) bir network olan internette, gönderen ve alıcı arasında sürekli ve tek bir bağlantı yoktur. Bunun yerine; bilgi, gönderildiği zaman ufak paketlere bölünür ve (alıcı tarafından yeniden birleştirmek üzere) değişik rotalardan aynı anda gönderilir. Buna karşılık, telefon sistemi bir devre-anahtarlamalı (circuit-switched) networktür. Bir devre-anahtarlamalı networkte bir bağlantı yapıldığı zaman (örneğin bir telefon görüşmesi), networkün o kısmı sadece bu tek bağlantı için kullanılır.

1. İnternet bir paket-anahtarlamalı networktür, bunun anlamı bir bilgisayardan diğerine internet üzerinden bilgi gönderildiğinde, bu verinin ufak paketlere bölünmesi demektir. Bir seri anahtar olan routerlar her paketi internet üzerinden ayrı ayrı gönderirler. Bütün paketler alıcı tarafa ulaştığı zaman, bunlar yeniden birleştirilerek uniform ve orijinal haline getirilir. Veriyi paketlere bölme, bu paketleri internet üzerinde yönlendirme ve diğer tarafta yeniden birleştirme işlemleri iki protokol tarafından yapılır: Veriyi yönlendiren İnternet Protocol ve veriyi paketlere ayırıp, alıcı bilgisayarda yeniden birleştiren Transmission Control Protocol.

2. Donanım limitlerinin de içinde olduğu birkaç nedenden dolayı internet üzerinden gönderilen bilgi, her biri 1.500 karakterden daha ufak paketlere bölünmelidir. Her pakete bir başlık (header) verilir. Bu başlıkta çeşitli bilgiler bulunur, örneğin paketlerin diğer paketlerle birleştirilmesi durumunda bu işin hangi sırayla yapılacağı gibi. TCP her paketi yaratırken ayrıca bir de kontrol toplamı (checksum) hesap eder ve başlığa ekler. Bu TCP tarafından alıcı tarafta kullanılan bir sayıdır ve TCP bu sayıyı kontrol ederek transfer sırasında pakette herhangi bir hata meydana gelip gelmediğini kontrol eder. Checksum paketteki veri miktarının hassas bir ölçümüdür.

3. Her paket ayrı IP "zarflarına" konur. Bu zarflar internete veriyi nereye göndermesi gerektiğini söyleyen adres bilgileri içerir. Her veri için kullanılan her zarfta, yeniden birleştirilmek üzere aynı adrese gönderilmesi için, aynı adres bilgileri bulunur. IP "zarfları" gönderici adresi, alıcı adresi, imha edilmeden önce geçmesi gereken zaman gibi bilgiler içerir.

4. Paketler İnternet üzerinden gönderilirken, yol üzerindeki routerlar (yönlendirici) IP zarflarını inceler ve adreslerine bakarlar. Bu routerlar, hedefe en yakın olan bir sonraki routera paketleri yollamak için en uygun rotayı seçerler. Paketler bir dizi routerdan geçtikten sonra yerlerine ulaşırlar. İnternet üzerindeki trafik yükü sürekli değiştiğinden, paketler farklı rotalardan yollanabilir ve karışık sıralı varabilirler.

5. Paketler hedefe ulaştıkça TCP her paket için bir checksum hesaplar. Sonra bunu paket içinde gönderilen checksum ile karşılaştırır. Eğer checksum'lar birbirine uymuyorsa, TCP paketin içindeki verinin transfer sırasında hasar gördüğünü anlar.Daha sonra bu paketi imha ederek orijinal paketin yeniden transfer edilmesini talep eder.

6. Bütün hasarsız paketler alıcı bilgisayar tarafından alındığında, TCP bunları birleştirerek orijinal ve uniform hale getirir.

Şekil 3.6.'da TCP protokolü tarafından hazırlanan segmentin(taşınabilecek uygun uzunluktaki parça) formatı görülmektedir.

Şekil 3.6. TCP Protokolü Başlığı (Güler, 1999).

TCP Protokolü başlığı içinde kullanılan alanlar aşağıda açıklanmıştır:

- Gönderici Port No: Bir üst katmanda TCP hizmetini isteyen uygulama protokol prosesinin kimliği durumundadır. Karşı mesaj geldiğinde bir üst katmana iletmek için, o protokolün adı değil de port numarası kullanılır.

- Alıcı Port No: Gönderilen veri paketinin alıcı tarafta hangi uygulama prosesine ait olduğunu belirtir.

- Sıra Numarası: Gönderilen paketin sıra numarasını gösterir. Gönderilmeden önce daha küçük parçalara ayrılan verinin, alıcı kısımda yeniden aynı sırada elde edilmesinde kullanılır.

- Onay Numarası: Gönderilen verinin en son hangi sekizlisinin alındığını göndericiye iletmek için kullanılır. Örneğin 4 sayısı gönderilmişse 4'e kadar olanlar alınmış 4. Paketin istendiğini bildirir.

- Başlık Uzunluğu: TCP başlığında var olan 32 bit uzunluğundaki sözcüklerin sayısını gösterir.

- Saklı Alan: İlerde olabilecek genişlememem için gizli tutulmuştur.

- Kod Bitleri: Kontrol bilgilerini taşımak için kullanılır.

- Pencere: Alıcının tamponunda (buffer) kullanılan alanın oktet olarak ifade edilmesi.

- Hata Sınama Bitleri: Verinin ve başlığın hatasız olarak aktarılıp aktarılmadığını kontrol etmek için kullanılır.

- Acil İşaretçisi: Acil veri alıcının uygulama katmanında öncelikle değerlendirilmesi gereken veridir.

- Veri: Değerlendirilmesi istenen verinin bölüm içindeki yerini işaret eder.

TCP/IP protokolünün sağlamış olduğu bazı avantajlar aşağıda belirtilmiştir:

- IBM, 3Com, DEC, Sun, HP ve benzeri firmaların çoğu TCP/IP protokolünü benimsemişlerdir.

- Her türlü bilgisayar ortamında rahatlıkla çalışmaktadır.(PC, Server, İş İstasyonu, Mainframe gibi)

- Unix ortamına çok iyi entegrasyon sağlar.

- Dinamik router(yönlendirici) teknolojisini destekler.

- İstemci – server mimarisini destekler.

- Ethernet, X.25 ve Token Ring gibi birçok yerel ve genel ağ protokollerini destekler.

- Peer to peer (noktadan noktaya) mimarisini destekler.

- OSI uygulamaları TCP/IP protokolü üzerinde rahatlıkla çalıştırılabilir (Anonim, 1999).

3.1.6.1. TCP/IP Mimarisi

Bu bölümde TCP/IP'nin alt katmanlarında bulunan protokoller ile ilgili detaylı bilgiler verilecek ve sistemin temel çalışma prensipleri açıklanmaya çalışılacaktır.

3.1.6.1.1 Fiziksel Katman

Fiziksel katman için herhangi bir protokol tanımlanmamıştır. Bu katmanda ağ bağlantı cihazları, ethernet ve benzeri ağ protokolleri, kablolama alt yapısı ve repeaterlar(tekrarlayıcı) gibi ağın fiziksel yönü ile ilgili araçlar bulunur.

Çizelge 3.2. TCP/IP Protokolü Alt Katmanları (Çavdar, 2000).

TCP/IP MODEL

UYGULAMA KATMANI
TAŞIMA (ULAŞIM) KATMANI
AĞ (YÖNLENDİRME) KATMANI
FİZİKSEL KATMAN

Fiziksel katman ham bitleri bir haberleşme kanalı üzerinden iletmekle ilgilidir. Tasarımının amacı, bir uçtan 1 biti(bilgisi) gönderildiğinde karşı taraftan da 1 bilgisini alınmasını sağlamaktır.

Fiziksel katman gerçekte Data Link Connection (DLC) ve Fiziksel ortamı içermektedir. Günümüzde pek çok bilgisayar ağının Etherneti temel iletişim ortamı olarak kullanmasından dolayı da Ethernet teknolojisi örnek olarak verilebilir. Dolayısıyla burada Ethernet ortamının TCP/IP ile olan iletişimini ele alınacaktır. Ethernet kendine has bir adresleme kullanır. Ethernet tasarlanırken dünya üzerinde herhangi bir yerde kullanılan bir Ethernet kartının tüm diğer kartlardan ayrılmasını sağlayan bir mantık izlenmiştir. Ayrıca, kullanıcının Ethernet adresinin ne olduğunu düşünmemesi için her Ethernet kartı fabrika çıkışında kendisine has bir adresle piyasaya verilmektedir. Her Ethernet kartının kendine has numarası olmasını sağlayan tasarım 48 bitlik fiziksel adres yapısıdır. Ethernet kartı üretici firmalar merkezi bir otoriteden üretecekleri kartlar için belirli büyüklükte numara blokları alır ve üretimlerinde bu numaraları kullanırlar. Böylece başka bir üreticinin kartı ile bir çakışma meydana gelmez.

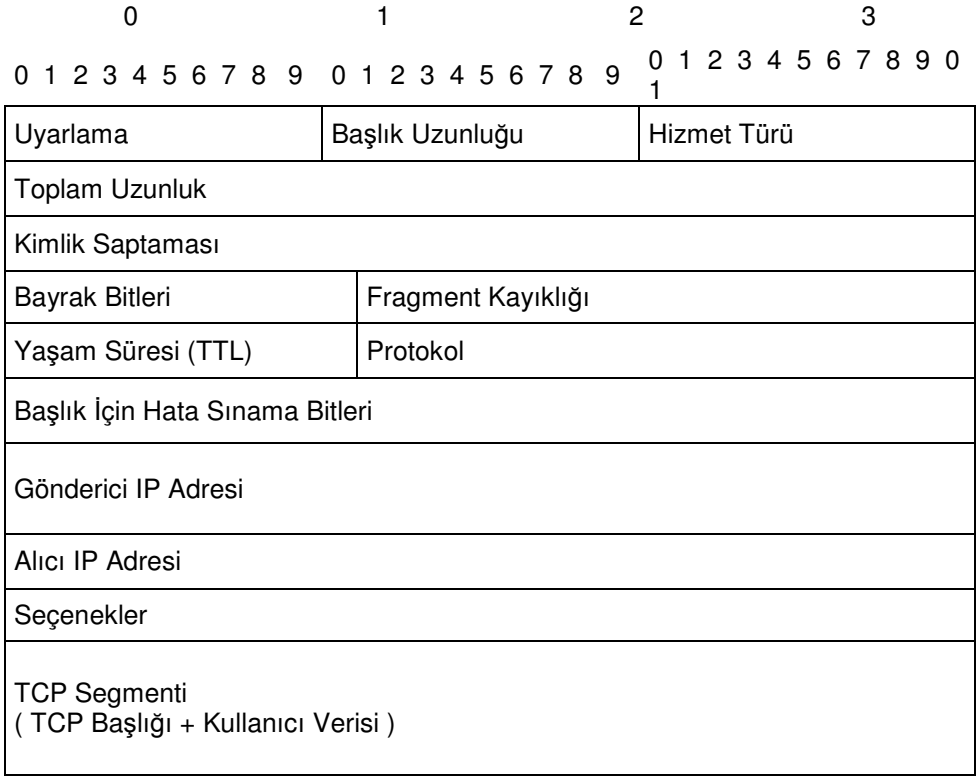
Ethernet, teknoloji olarak yayın teknolojisini (Broadcast Medium) kullanır. Yani bir istasyondan Ethernet ortamına yollanan bir paketi o Ethernet ağındaki tüm istasyonlar görür. Ancak doğru varış noktasının kim olduğunu, o ağa bağlı makineler Ethernet başlığından anlarlar. Her Ethernet paketi 14 octet'lik bir başlığa sahiptir. Bu başlıkta kaynak ve varış Ethernet adresi ve bir tip kodu vardır. Dolayısıyla ağ üzerindeki her makine bir paketin kendine ait olup, olmadığını bu başlıktaki varış noktası bilgisine bakarak anlar (Bu Ethernet teknolojisindeki en önemli güvenlik boşluklarından birisidir). Bu noktada Ethernet adresleri ile Internet adresleri arasında bir bağlantı olmadığını belirtmekte yarar var. Her makine hangi Ethernet adresinin hangi Internet adresine karşılık geldiğini tutan bir tablo tutmak durumundadır. Tip kodu alanı aynı ağ üzerinde farklı protokollerin kullanılmasını sağlar. Dolayısıyla aynı anda TCP/IP, DECnet, IPX/SPX gibi protokoller aynı ağ üzerinde çalışabilir. Her protokol başlıktaki tip alanına kendine has numarasını koyar. Kontrol toplamı (Checksum) alanındaki değer ile komple paket kontrol edilir. Alıcı ve vericinin hesapladığı değerler birbirine uymuyorsa paket yok edilir. Ancak burada kontrol toplamı başlığın içine değil de paketin sonuna konulur. Ethernet katmanında işlenip gönderilen mesaj ya da bilgi aşağıdaki formatı alır.

Bu paketler (frame) varış noktasında alındığında bütün başlıklar uygun katmanlarca atılır. Ethernet ara yüzü Ethernet başlık ve kontrol toplamını atar. Tip koduna bakarak protokol tipini belirler ve Ethernet cihaz sürücüsü (Device Driver) bu datagramı IP katmanına geçirir. IP katmanı kendisi ile ilgili katmanı atar ve protokol alanına bakar,

protokol alanında TCP olduğu için segmenti TCP katmanına geçirir. TCP sıra numarasına bakar, bu bilgiyi ve diğer bilgileri iletilen dosyayı orijinal durumuna getirmek için kullanır. Sonuçta bir bilgisayar diğer bir bilgisayar ile iletişimi tamamlar (Çavdar, 2000).

3.1.6.1.2. Yönlendirme(Ağ) Katmanı

TCP katmanına gelen bilgi segmentlere ayrıldıktan sonra IP katmanına yollanır. IP katmanı, kendisine gelen TCP segmentinin içinde ne olduğu ile ilgilenmez. Sadece kendisine verilen bu bilgiyi ilgili IP adresine yollamak amacındadır. IP katmanının görevi bu segment için ulaşılmak istenen noktaya gidecek bir "yol" (route) bulmaktır. Arada geçilecek sistemler ve geçiş yollarının bu paketi doğru yere geçirmesi için kendi başlık bilgisini TCP katmanından gelen segment'e ekler. TCP katmanından gelen segmentlere IP başlığının eklenmesi ile oluşturulan IP paket birimlerine datagram adı verilir. IP başlık bilgisinin formatı Şekil 3.7.'de gösterilmiştir.



Şekil 3.7. IP Başlığı İçindeki Alanlar (Anonim, 1999).

IP başlığında kullanılan alanların amaçları aşağıda açıklanmıştır:

- Uyarlama: O anda kullanılan IP'nin uyarlanmasını gösterir. Farklı uyarlamada başlıktaki alanların yerleri değişiklik göstereceğinden, paketin doğru yorumlanması için kullanılır.

- Başlık Uzunluğu: Datagram başlığının gerçek uzunluğunu gösterir. Başlık alanının kaç adet 32 bitlik sözcükten oluştuğunu gösterir.

- Hizmet Türü: Datagramın nasıl yönlendirileceğini belirler. Yönlendirmede yapılan yol seçiminde ve bağlantıda kullanılır. Datagramlara bu alan aracılığıyla önem düzeyi atanabilir. Göndericinin ağdan beklediği güvenilirlik, hız ve gecikmenin düzeyini belirtir. Ancak bu alanı mevcut yönlendiricilerin pek azı değerlendirmektedir.

- Toplam Uzunluk: Tüm IP paketinin uzunluğunu belirtir. Toplam uzunluk maksimum 65535 bit olabilir.

- Kimlik Saptaması: Kullanıcı karşı tarafla etkileşim içindeyken, mesajlar parçalanarak birçok datagram içinde gönderilebilir. Bu alan aynı kullanıcı mesajının farklı datagramlar içinde bulunması durumunu açıklayan kimlik bilgisi içerir.

- Bayrak Bitleri: Üç tane olan bayrak bitlerinden ilki (D biti) içinde bulunduğu datagramın kaç parçadan oluştuğunu belirtir. Eğer 1 ise gönderilen verinin tek datagramdan oluştuğu anlaşılır. Alıcıya veri gönderildikten sonra başka mesaj olmadığına dair mesaj gönderilir. İkinci bayraksa, parçalanıp birçok datagram haline gönderilen verinin en son olduğunu belirtir. Üçüncüsü saklı tutulmuştur.

- Fragment Kayıklığı: 8 byte'lık birimler halinde fragmentin datagram içindeki konumun gösterir. DF (Don't Fragment) yönlendiricilerden datagramı fragmentlere bölmemesini söyleyen 1 bitlik bir istek alanıdır. Alıcının fragmentleri birleştiremediği durumlarda gereklidir. MF (More Fragment), bir datagramın son fragmenti dışındaki tüm fragmentlerinde MF=1'dir.

- Yaşam Süresi(TTL): Datagramın ağ üzerinde dolaşma süresini belirler. Verici tarafında yerleştirilen dolaşma değeri her düğümden geçerken azaltılır. Sıfıra ulaşırsa kaybolmuş olduğu varsayılarak datagram ağdan çıkarılır. TTL alanına başlangıçta 255 veya daha küçük bir tam sayı yerleştirilir. Her yönlendiricide bu alandaki değer 1 eksiltilir. Ayrıca yönlendiricide paket bir bekleme kuyruğuna alınırsa her geçen birim zamanda TTL alanındaki sayı 1 eksiltilir. Sayı 0'a ulaşırsa paket geçersiz sayılır ve atılır. Paketi atan yönlendirici paketin gönderildiği yönlendiriciye bir uyarı paketi gönderir.

- Protokol: Bir datagramın hangi üst katman protokolüne ait olduğunu belirtir. Alıcı tarafın IP katmanı bu alana bakarak paketi bir üstünde bulunan protokollerden hangisine ileteceğini anlar.

- Başlı Hata Sınama Bitleri: Datagram başlığında bir bozulma olup, olmadığını belirlemeye yarar. Her yönlendiricide bu alandaki değer kullanılarak datagramın bozulup, bozulmadığı araştırılır. Sonuç olumlu ise paket bir sonraki yönlendiriciye gönderilir bu arada başlıktaki bazı değerlerle birlikte (örneğin TTL) bu alandaki değer de gönderilen pakette yeniden hesaplanır. Yöntem yalnızca başlıktaki hataları açığa çıkardığı için ulaşım katmanının verideki muhtemel bozuklukları yakalayacak önlemler alması gerekebilir.

- Gönderici IP Adresi: Datagramı gönderen yerin gerçek Internet adresi yerleştirilir.

- Alıcı IP Adresi: Datagramın gideceği yerin Internet adresi yerleştirilir.

• Seçenekler: Bu alan değişik amaçlar için kullanılır. Güvenlik hata raporlama gibi seçimliktir. Ancak kullanılırsa 32 bitin katları uzunluğunda olmalıdır. Yani uzunluğu 32 bitin katlarına tamamlanmalıdır.

• TCP Segmenti: Bir üst katmandan gelen veriyi içerir (Anonim, 1999).

3.1.6.1.3. Taşıma (Ulaşım) Katmanı

Ulaşım katmanın temel işlevi, hazırlanan veriyi alıp, ihtiyaç duyulduğunda küçük bileşenlere ayırıp ağ katmanına geçirerek, diğer uca bu parçaların doğru bir şekilde ulaştığına emin olmaktır. Normal şartlar altında, ulaşım katmanı, uygulama katmanı tarafından ihtiyaç duyulan her taşıma bağlantısı için bir sanal ağ bağlantısı oluşturur.

Eğer uygulama katmanı tarafından ihtiyaç duyulan taşıma bağlantısı yüksek bir kapasite isterse, ulaşım katmanı birçok ağ bağlantısı oluşturup, kapasiteyi artırmak için veriyi bu bağlantılara paylaştırır. Öte yandan, farklı ağ bağlantılarının oluşturulması maliyeti artırdığı durumlarda ulaşım katmanı çeşitli taşıma bağlantılarını bir ağ bağlantısı üzerinde maliyeti azaltmak için birleştirebilir. Tüm durumlarda ulaşım katmanı birleştirme işinin uygulama katmanına yansımaması için gereklidir.

Ulaşım katmanı ayrıca uygulama katmanına sonuç olarak ağ kullanıcılarına ne tip servisler sunulacağına karar verir. Ulaşım bağlantısının en popüler tipi gönderildiği sıra ile hatasız uçtan-uca ulaştıran kanaldır. Ancak, diğer tip taşıma, servis ve taşıma bilgisi ayrılmış mesajları değişik lokasyonlara ileten ve hedefine ulaştırma konusunda herhangi bir garanti vermeyenidir. Servis tipi bağlantı sağlandığında belirlenir.

Ulaşım katmanı, gerçek bir kaynaktan hedefe veya uçtan uca katmandır. Başka bir değişle, Kaynak sistemde çalışan bir program mesaj başlıkları ve denetim mesajlarını kullanarak, hedef sistemdeki benzeri bir programla konuşur.

Birçok bilgisayar üstünde birden fazla programı çalıştırır, yani sisteme giren ve çıkan birçok bağlantı vardır. Bu yüzden hangi mesajın hangi bağlantıya ait olduğunun belirlenmesi için bir metoda ihtiyaç duyulur. Taşıma başlığı bu bilginin koyulabileceği bir yerdir.

Değişik mesajları bir kanal içinde birleştirmenin yanında, taşıma katmanı ağ boyunca bağlantıların kurulması ve kaldırılmasını da takip etmelidir. Bu, bir bilgisayar üzerinde kiminle konuştuğunu tarif edecek bir tür isimlendirme mekanizması gerekliliğini doğurur. Ayrıca hızlı bir bilgisayarın yavaş bir bilgisayarı aşmaması için bilgi akışını düzenleyecek bir mekanizmanın olması gereklidir (Güler, 1999).

3.1.6.1.4. Uygulama Katmanı

Uygulama katmanı çokça ihtiyaç duyulan birçok protokolü içerir. Örneğin dünyada birbirine uyumsuz yüzlerce terminal, uç birim tipi vardır. Örneğin her biri farklı ekran düzenleri, metin silme ve düzenleme için farklı ara yüzler, imleç konumlandırması vs. kullanan değişik uç birimlerle çalışan bir tam ekran metin editörünü ele alalım.

Bu problemi çözmenin yolu editörlerin ve diğer programların yazabildiği sanal bir ağ uç birimi oluşturmaktır. Her uç birim tipini karşılamak için, sanal uç birimin fonksiyonlarının gerçek uç birim üzerine eşleşmesini sağlamak için bir yazılım yazılmalıdır. Örneğin bu yazılım, editör sanal uç birimin imlecini sol üst köşeye konumlandırdığında, yazılım gerçek uç birimde imlecin asıl konumuna yerleşimi için düzgün komut dizisini işlemelidir. Tüm sanal uç birim yazılımları uygulama katmanındadır.

Uygulama katmanının diğer bir işlevi ise dosya transferidir. Değişik dosya sistemleri, değişik dosya isimlendirme tanımlamalarına, metin bilgisinin temsili için değişik metotlara sahiptir. Değişik dosya sistemlerinden dosya transferleri bu uyumsuzlukları ortadan kaldırmayı gerektirir. Bu iş, yine elektronik posta, dizin taraması ve diğer özel ve genel amaçlı işlevlerde yapıldığı gibi uygulama katmanına aittir.

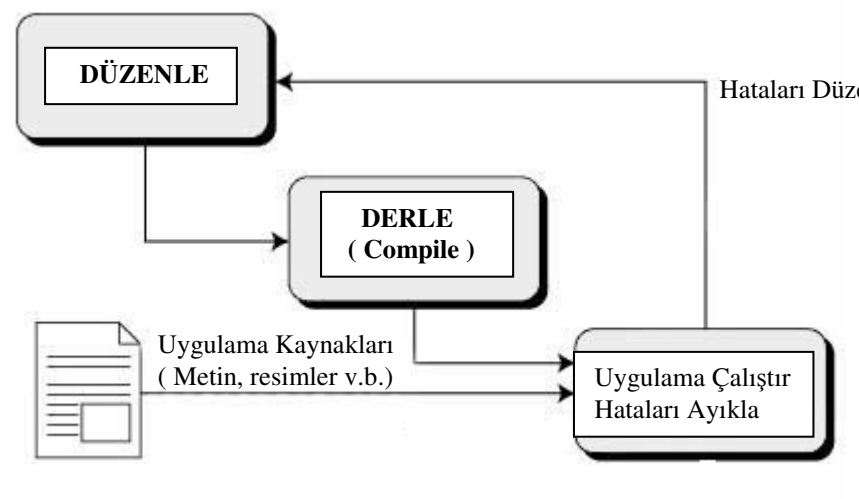
TCP/IP'nin kullanıldığı en önemli servislerden birisi elektronik postadır.(e-mail) E-posta servisi için bir uygulama protokolü belirlenmiştir (SMTP). Bu protokol e-mail'in bir bilgisayardan bir başka bilgisayara nasıl iletileceğini belirler. Yani e- postayı gönderen ve alan kişinin adreslerinin belirlenmesi, mektup içeriğinin hazırlanması vs. gibi. Ancak e-posta servisi bu mektubun bilgisayarlar arasında nasıl iletileceği ile ilgilenmez, iki bilgisayar arasında bir iletişimin olduğunu varsayarak mektubun yollanması görevini TCP ve IP katmanlarına bırakır. TCP katmanı komutların karşı tarafa ulaştırılmasından sorumludur. Karşı tarafa ne yollandığı ve hatalı yollanan mesajların tekrar yollanmasının kayıtlarını tutarak gerekli kontrolleri yapar. Eğer gönderilecek mesaj bir kerede gönderilemeyecek kadar büyük ise (Örneğin uzunca bir e-posta gönderiliyorsa) TCP onu uygun boydaki segment'lere (TCP katmanlarının iletişim için kullandıkları birim bilgi miktarı) böler ve bu segment'lerin karşı tarafa doğru sırada, hatasız olarak ulaşmalarını sağlar. Internet üzerindeki tek servis e-posta olmadığı için ve segment'lerin karşı tarafa hatasız ulaştırılmasını sağlayan iletişim yöntemine tüm diğer servisler de ihtiyaç duyduğu için TCP ayrı bir katman olarak çalışmakta ve tüm diğer servisler onun üzerinde yer almaktadır. Böylece yeni bir takım uygulamalar da daha kolay geliştirilebilmektedir. Üst seviye uygulama protokollerinin TCP katmanını çağırmaları gibi benzer şekilde TCP de IP katmanını çağırmaktadır. Ayrıca bazı servisler TCP katmanına ihtiyaç duymamakta ve bunlar direkt olarak IP katmanı ile görüşmektedirler. Böyle belirli görevler için belirli hazır yordamlar oluşturulması ve protokol seviyeleri inşa edilmesi stratejisine "katmanlaşma" adı verilir. Yukarıda verilen örnekteki e- posta servisi (SMTP), TCP ve IP ayrı katmanlardır ve her katman altındaki diğer katman ile konuşmakta diğer bir deyişle onu çağırmakta ya da onun sunduğu servisleri kullanmaktadır (Güler, 1999).

3.2. Metot

3.2.1 Uygulama Geliştirmede Algoritmaların Kullanımı

Uygulama geliştirilirken bir problemin nihai sonucuna hemen ulaşılamaz. Sonuca ulaşma birçok denemeyi gerektiren öteleme işlemleridir. Şekil 3.8 akış şeması biçiminde bu deneme işlemlerini (algoritma kullanım döngüsü) göstermektedir. Başarılı bir ürün elde etmek için gerekli olan ilk şey yapılacak prototipin geliştirilmesidir.

Programlanabilir cihazlar için uygulama geliştirirken geliştirme ve test döngüsü için algoritmalar kullanılır. Algoritmalardan elde edilen çözüm yöntemi herhangi bir programlama dilinde kodlanır. Geliştirilen uygulamanın son hali yani hatalardan ayıklanmış hedeflenen çözüm cihaza yüklenir. Gerçekten algoritmalar uygulama geliştirmek için iyi bir çözümdür.



Şekil 3.8. Uygulama geliştirmede algoritmaların kullanım döngüsü.

3.2.2. Geliştirilen Uygulama

Borland C ++ Builder programlama dili kullanılarak geliştirilen uygulamada ki temel amaç USB webcam'lar ile elde edilen gerçek zamanlı görüntüyü yakalamaktır. Sonrasında ise server tarafından yakalanan bu görüntünün TCP/IP kullanılarak client konumundaki cihazlara ağ veya internet ortamı kullanılarak aktarılmasıdır. Uygulama iki ana bölümden oluşmaktadır. Bunlar:

1. Server kısmı.
2. Client kısmı.

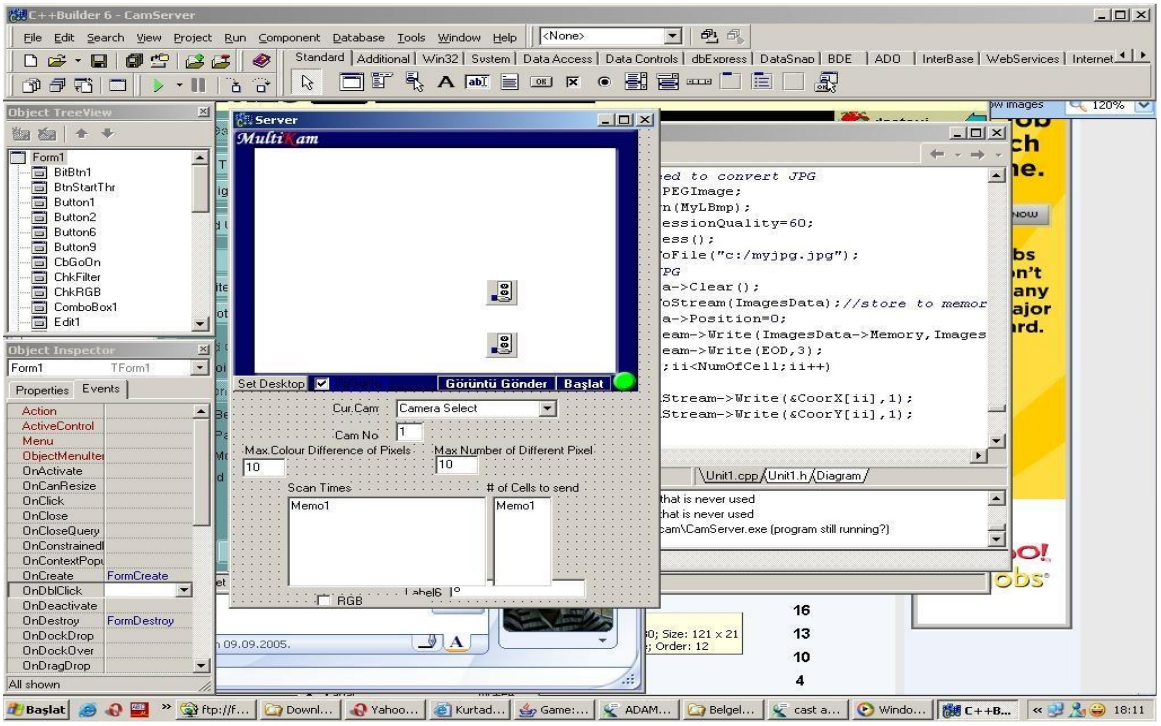
3.2.2.1. Geliştirilen Uygulamanın Server Kısmı

Şekil 3.9'da Borland C++ Builder ortamı ve geliştirilen uygulamanın server kısmının kullanıcı ara yüzü görülmektedir. Programın bu kısmında kullanılan metot, USB portuna bağlı webcam görüntüsünü Capture API'lerini kullanarak yakalamak ve bunları client durumundaki cihazlara göndermek için hazır tutmaktır.

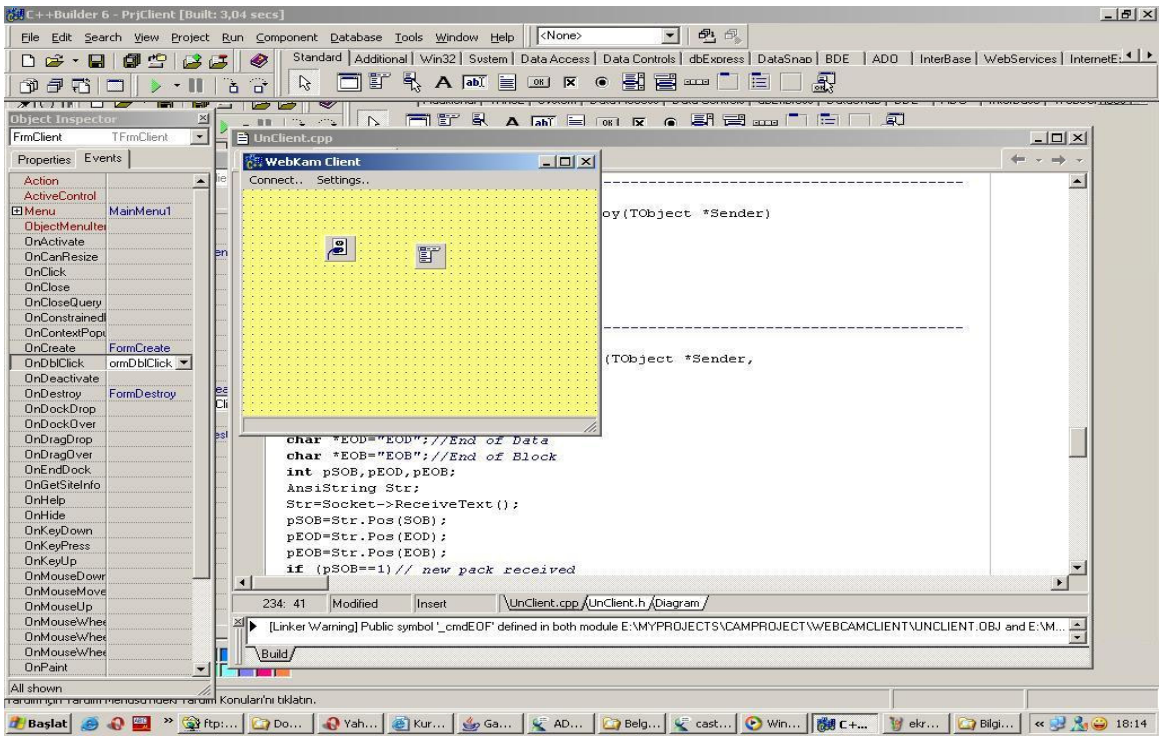
3.2.2.2. Geliştirilen Uygulamanın Client Kısmı

Programın Client kısmında ise server kısmında hazır tutulan webcam görüntüsü ağ veya internet ortamında TCP/IP aracılığı ile 76 numaralı rezerve edilmemiş Port kullanılarak Client konumundaki cihaz veya cihazlar tarafından gerçek zamanlı olarak elde

edilmektedir. Şekil 3.10.' da Borland C++ Builder ortamı ve uygulamanın client kısmının kullanıcı ara yüzü görülmektedir.



Şekil 3.9. Borland C++ Builder ortamı ve uygulamanın server kısmının kullanıcı ara yüzü.



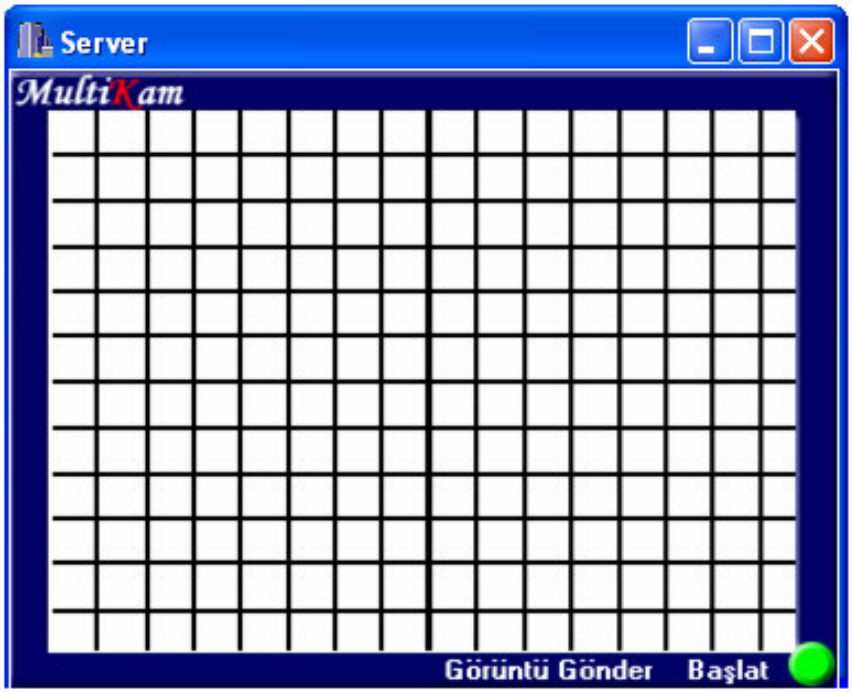
Şekil 3.10. Borland C++ Builder ortamı ve uygulamanın client kısmının kullanıcı ara yüzü.

4. BULGULAR VE TARTIŞMA

Tez projemiz önceki bölümlerde de izah edildiği şekilde server ve client olarak iki bölümden oluşturulmuştur. Bu bölümler de yapılan işlemler ve gelişmeleri aşağıda açıklanmıştır:

4.1. Uygulamanın Server Kısmı

Server, uygulamanın web kameranın bağlı olduğu PC veya Laptop' ta çalışan kısmıdır. Server tarafında öncelikli olarak Windows Create Capture API' sı ile resim yakalama penceresi oluşturulmuştur. Kamera görüntü ekranı Şekil 4.1.' de gösterildiği gibi 16X12 hücre olarak seçilmiştir. Her bir hücre 20X20 Piksel olup, çözünürlük 320X240 Piksel' dır. Daha sonra Get Dc API' sı ile USB Kameradan alınan görüntünün içeriği Create Capture API' sı ile oluşturulan pencere içerisine alınmaktadır. Kameradan alınan görüntüler Bitmap resim formatındadır. Server kısmında elde edilen bu görüntü oluşturulan ekranda izlenmektedir.



Şekil 4.1. Uygulamanın Server Ekranı

Server kısmının ikinci aşamasında 16X12 hücreden oluşturduğumuz ve her hücresi 20X20 Piksel olan ve izleme konumunda bulunan server ekranı, istememiz halinde server IP numarasının tanıtılması ile connect olabilen client' a gönderilebilmektedir. Bu gönderme işlemi; 192 hücre(76.800 Byte) renk ve piksel olarak karşılaştırılmakta ve bu karşılaştırma sonucu Bitmap olarak değişen hücreler koordinat bilgileri ile birlikte JPEG sıkıştırma algoritması kullanılarak JPG formatında paketlenmekte ve server tarafından verilecek talimat ile ağ veya internet ortamında TCP/IP ile client'a gönderme şeklinde gerçekleştirilmektedir.

Uygulamanın server tarafında program işlem sırası aşağıda belirtildiği şekilde gerçekleşmektedir:

```
Procedure InitCam;                  //Kamera ayarı için ilk olarak yapılması gereken işlemler
        ScrW←320                    //Kamera görüntüsünün boyutları
        ScrH←240
        DivXCount←16;               //İncelenecek hücre matrisinin boyutları
        DivYCount←12;
        CreateCaptureWindow;        //Kamera için capture penceresi yaratılıyor
        MyDc←GetDc;                 //Kameranın içeriği alınıyor
        SetCamParamaters;           //Kameranın parametreleri set ediliyor
End Procedure;

Procedure SendChangedCells;                 //Değişen hücrelerin tespiti ve uzaktaki client' a
gönderilmesi
        Read ColorDif;                      //Renkler arası farklılık set ediliyor
        Read DiffCount;                     //Maksimum farklı nokta sayısı okunuyor
        for (j=0;j<DivYCount;j++)  //Her bir hücre ayrı ayrı taranıyor
        {
                for (i=0;i<DivXCount;i++)

                {
                    MyBmp←MyDc[i,j]    //Kıyaslama için gerekli hücre Bitmap'e alınıyor
                                        //Bitmap görüntü parcacigi
                                //Bir kamera görüntüsünde DivXCount*DivYCount kadar
hücre vardır

                    GetBitmapPointer(MyBmp,p1)          //Hafıza kıyaslama işlemi için
Bitmap'in hafıza işaretçisi alınıyor
                                                        //Applyin median filter to R G B
                                                       // Renkteki diğer dalgalandırmaların
bulanıklaştırılması ile renk kıyaslaması yapılır
                    ApplyMedianFilter(p1,  CellW,CellH);        //Red
                    ApplyMedianFilter(p1+1,CellW,CellH);     //Green
                    ApplyMedianFilter(p1+2,CellW,CellH);     //Blue
                                //Hafıza içerisinde renkler RGB olarak sırasıyla bulunur
                                //Her bir renk 1 byte uzunluğundadır
                  CompareMem(p1,p2, imageSize);         //p1 şu anki görüntü p2 ise bir
önceki karedeki görüntü
                    If Changed then
                      Begin
                            P2←P1                   //Eski hücre görüntüsünün içine yeni
görüntü ekleniyor
                            Counter++;
                      End;
                    If Counter>100 or LastCell then       //Eğer değişen hücre sayısı 100
den fazla ise yada son hücrede ise
```

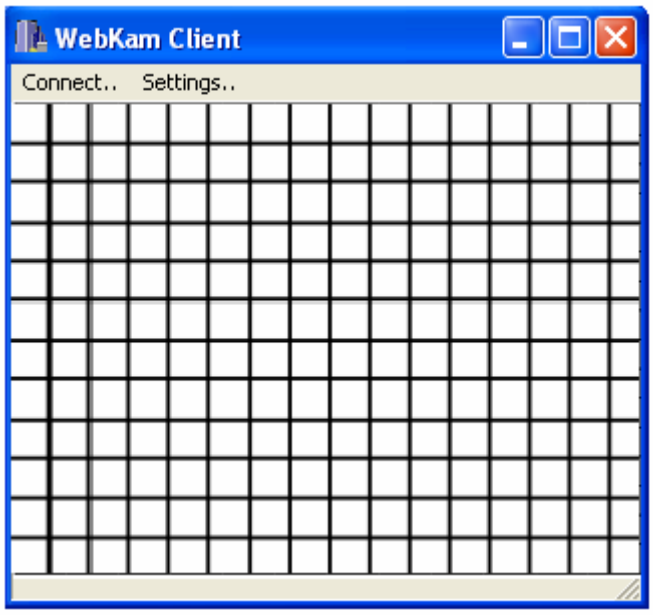
```
                    Begin
                     PrepareCells;       //Bu kısımda ise tüm değişen hücreler bir
araya getiriliyor ve tek bir görüntü haline geliyor

                      ConvertJpg;        //Fazla bir büyüklük olmaması için bu
BMP görüntüsü JPG ye çevriliyor
                     AddCellCoordinates;          //Değişen hücrelerin hücre
adreslerini gönderilecek pakete ekleniyor
                     SendAllConnectedClients;     //Tüm bağlı olan Client'lara
kamera görüntüsü aktarılıyor
                    End;                          //Hücre tarama işleminin sonu
End;    //Procedure 'un sonu
```

Gönderilen Paketin İçeriği:

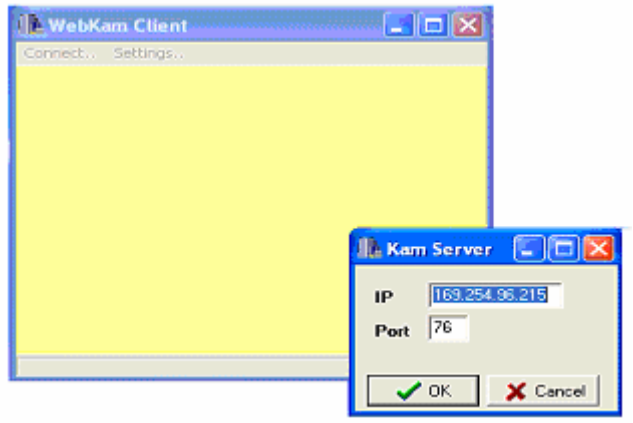
```
SOB (Start of buffers)
 ...
 ...
 ...
 image data
 ..
 ..
EOD(End of data)
 ..
 ..
 coordinations of cells
 ..
 ..
EOB(end of buffer)
```

4.2. Uygulamanın Client Kısmı

Client, uygulamanın server' dan farklı yerlerde bulunan bilgisayarlar da çalışan kısmıdır. Client tarafında da ilk önce Windows Create Capture API' sı ile resim yakalama penceresi oluşturulmuştur(Şekil 4.2.). Client penceresi üzerine iki adet buton yerleştirilmiştir. Bunlardan birisi "Settings" butonudur. "Settings" butonu ile bağlanacağımız server' ın IP numarası set edilmektedir. İkinci buton ise "Connect" butonudur. Bu buton ile set edilen IP adresindeki server' a ulaşılır. Eğer server gönderme konumunda ise 76 numaralı deneme ve test amaçlı kullanılan rezerve port üzerinden görüntü elde edilir(Şekil 4.3). İlk bağlantı durumunda server client' a 12X16 hücrenin tamamını gönderir. Daha sonra ise hücreleri karşılaştırarak sadece değişen bölümleri koordinatları ile JPEG sıkıştırma algoritması ile JPG formatında client'a gönderir. Client ise gelen sıkıştırılmış bu JPG formatındaki paketi açarak Bitmap formatına dönüştürür ve karşıdan gelen yeni hücreleri yine aynı paket ile gelen koordinattaki hücrede bulunan görüntüler ile değiştirir.



Şekil 4.2. Uygulamanın Client Ekranı



Şekil 4.3. Client Bağlantı Ekranı

Uygulamanın client tarafında program işlem sırası aşağıda belirtildiği şekilde gerçekleşmektedir:

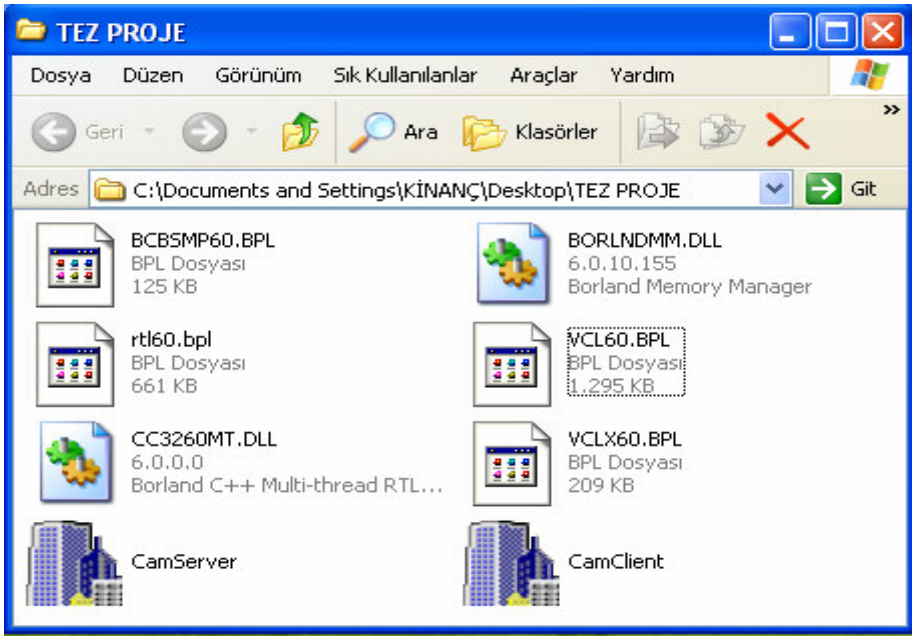
```
Procedure ConnectionSettings      //Server'a bağlanmak için gerekli ayarlar
        SetRemoteIp;              //Uzaktaki Kamera Server'ına ait Ip adresi
        SetRemotePort;            //Uzaktaki Kamera Port'unun belirlenmesi
End procedure

Procedure RecivePacks
  ReceivePack;                    //Uzaktaki serverdan tüm buffer alınıyor
  ExtractJpg;                     //Buffer içindeki jpg alınıyor
  ExtractMap;                     //Hücre haritası alınıyor
```

DrawJpgs; //Hücre haritasında ekranın gerekli yerlerine resimler basılıyor.
End procedure

4.3. Uygulamanın Çalıştırılması

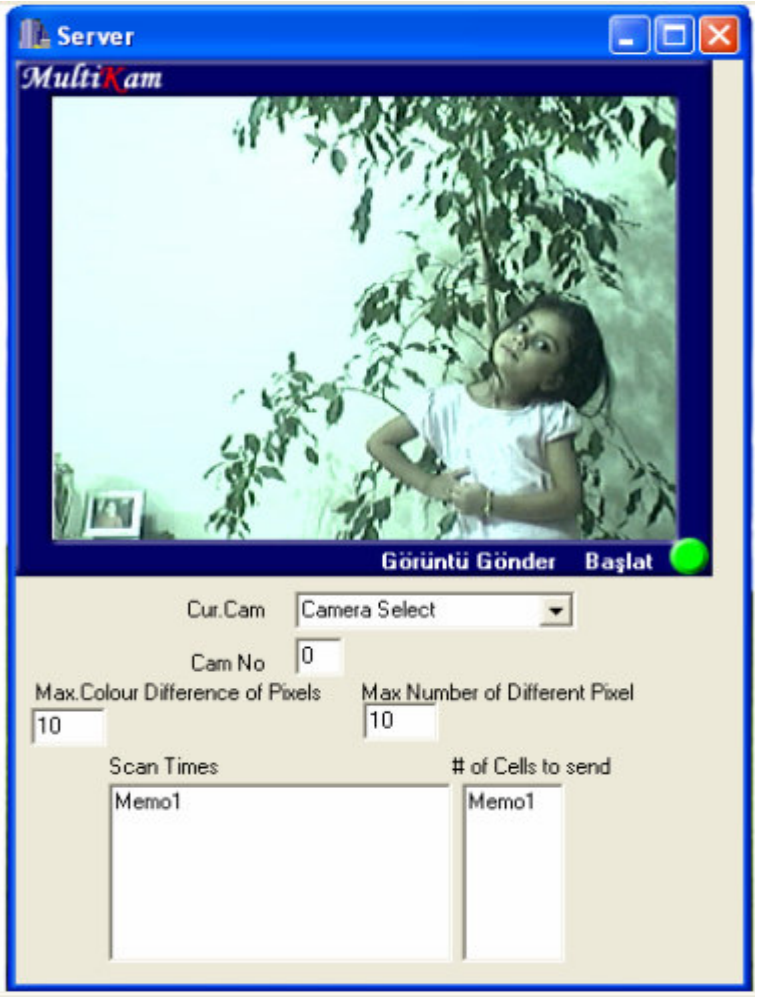
Uygulamanın server kısmı "Tez Proje" dosyasında(Şekil 4.4.) bulunan CamServer şekli üzerine çift tıkladığımızda ekrana gelmektedir. Ekran üzerinde bulunan "Cam No" bölümüne USB Web Kameranın numarası girilir(Server'a birden fazla kamera takılabilir). Başlat düğmesi ile de program çalıştırılarak web kameranın görüntüsü elde edilir(Şekil 4.5.).

Server ekranı üzerinde bulunan "Max Colour Difference of Pixsels" ve "Max Number of Different Pixel" kutucukları ile Clinet'a göndereceğimiz görüntünün renk ve hücre sayısında meydana gelen değişikliklerin kriterlerini belirliyoruz. Server bu kriterlere göre yaptığı kıyaslama sonucunda elde ettiği değişiklikleri client tarafına gönderiyor. Server ekranı üzerine oluşturulan "Scan Times" ve "# of Cells to send" kutucuklarından ise daha önce belirlemiş olduğumuz kriterlere göre meydana gelen renk ve hücre değişiminin tarama süresi ve client' a giden değişik hücre sayısı takip edilebilmektedir(Şekil 4.5.).



Şekil 4.4. Tez Proje Dosyası

Uygulamanın client kısmı "Tez Proje" dosyasında(Şekil 4.4.) bulunan CamClient şekli üzerine çift tıkladığımızda ekrana gelmektedir. Bu ekran üzerinde bulunan "Settings" butonunu tıkladığımız zaman karşımıza "KamServer" ekranı gelmektedir. Bu akran üzerinden görüntü alınacak server' a ait IP girilecek ve "OK" butonuna basılarak client'ın bağlanacağı server adresi set edilecektir. Daha sonra yine client ekranı üzerinde bulunan "Connect" butonuna tıklanarak server' a bağlantı istenecektir. Server ekranında bulunan "Görüntü Gönder" butonunun çalıştırılması ile de client ile server arasındaki bağlantı sağlanacak ve görüntü elde edilecektir(Şekil 4.6.).



Şekil 4.5. CamServer Penceresi Görüntüsü

Şekil 4.6. CamClient Penceresi Görüntüsü

- Uygulamada 240 X 320 çözünürlük seçilmiştir. 1024 X 768 veya daha farklı boyutlar da seçilebilirdi. Ancak nesnelerin minimum ayırt edilebilme çözünürlüğü yaklaşık 240X320 ebadıdır. Daha az bir çözünürlükte nesneleri ayırt etmek güçleşir. Daha yüksek çözünürlüklerde ise piksel sayısı artacağından görüntü işleme esnasında işlem sayısı çoğalacak, buda bize zaman kaybı yaratacaktır. Ayrıca büyük ebattaki bir görüntünün aktarımı da yüksek hızda iletim sistemlerine ihtiyaç duyacaktır ve görüntü aktarım kalitesi düşecektir. Görüntü kesikli bir şekilde aktarılacaktır. Bütün bunlara ilave olarak ta daha yüksek çözünürlükte webcam gerekecektir. Bu konu da uygulama sırasında yapılan denemelerde Çizelge 4.1.' de görülen sonuçlar elde edilmiştir. Bu sonuçlar da 240X320 çözünürlüğün uygun olduğunu göstermektedir.

- Görüntü için JPEG sıkıştırma algoritması kullanılmıştır. Görüntü webcam' dan Bitmap olarak alınıp, üzerinde meydana gelen değişiklikler işlendikten sonra JPEG sıkıştırma algoritması ile JPG formatına dönüştürülüp, istemde bulunan client' a gönderilmektedir. Client tarafından alınan bu sıkıştırılmış görüntü açılarak ekrana yansıtılmaktadır. Değişik formatlarda kullanılabilir, fakat JPEG sıkıştırma algoritması resmi en küçük boyuta indirebilen bir algoritmadır. Bu yüzden bu format tercih edilmiştir.

- Uygulamanın çalışması için gerekli özellikler: Windows XP işletim sistemi, donanım olarak Windows XP işletim sisteminin çalışabileceği konfigürasyon da bir PC veya Laptop, USB web kamera, ağ veya internet ve kaliteli bir görüntü için 512 Kbit/sn veya üzerinde bir hızda iletim ortamıdır(ADSL, Frame Relay v.b.). Bu konuda çalışmalar sırasında yapılan denemelerde iyi bir görüntünün 512 Kbit/sn ve üzerindeki hızlarda elde edildiği görülmektedir(Çizelge 4.2.).

Çizelge 4.1. Çözünürlüğe Bağlı Görüntü Gecikme Tablosu.(Elbistan Türk Telekom-K.Maraş Türk Telekom arası 2 Mb/s hızında iletim ortamın da alınan değerler).

ÇÖZÜNÜRLÜK	CLİENT GÖRÜNTÜ GECİKMESİ (milisaniye)
1024X 768	70
800X600	55
640X480	35
320X240	20

Çizelge 4.2. İletim Hızına Bağlı Görüntü Gecikme Tablosu.

İLETİM HIZI	CLİENT GÖRÜNTÜ GECİKMESİ (milisaniye)
2 Mb/s	20 (Görüntü Kaliteli)
512 Kb/s	30 (Görüntü Kaliteli)
256 Kb/s	55(Görüntü az durağan)
56 Kb/s (33.46 Kb/s Bağlantı hızı)	90(Görüntü durağan)

5. SONUÇ VE ÖNERİLER

Uygulamalı olarak yapılan bu tezde Borland C++ Builder yazılım dili kullanılarak gerçek zamanlı görüntü server konumunda bulunan ana makinede elde edilmiş ve elde edilen bu görüntü ise istek halinde client durumunda bulunan makinelere(PC veya Laptop) ağ veya internet ortamında TCP/IP iletim protokolü kullanılarak yine gerçek zamanlı olarak iletilmiştir.

Tez içerisinde Borland C++ Builder'ın tercih edilmesinin sebepleri, işletim sisteminin Windows olmasının faydaları ve üstünlükleri ve TCP/IP Protokolü geniş olarak açıklanmıştır.

İnternet ve internete bağımlı teknolojiler sürekli yaygınlaşmakta, hitap etmiş olduğu kitle ise sürekli genişlemekte ve sınır tanımamaktadır. Bu süreç içerisinde yeni yeni uygulamalar hayata geçirilmekte ve zamanla lüks olarak düşünülen birçok şeyi ihtiyaç olarak karşımızda bulmaktayız. Ülkemizde bundan 3–5 yıl önce Dial-Up olarak bağlandığımız 56 kbit/s hızındaki (bu erişim ki çoğu zaman 40kbit/s' ye ulaşamıyor iken) internet erişimi ADSL teknolojisi ile en düşük 256kbit/s' ye yükselmesine rağmen kullanıcılar hala daha yüksek hızlar talep etmektedir. İlerleyen zamanda iletişim ortamındaki ücretlerin düşmesi ile bireysel kullanıcılar bile 512kbit/s, 1Mb/s gibi hızlara rahatlıkla erişebilecektir. Bütün bu gelişmeler bilgisayar ve erişim sistemine sahip kullanıcılara birçok hizmeti ayaklarına kadar getirme olanağı sağlamaktadır. Uygulama projemiz de veri iletimi yapılan sistemin hızının yükselmesi ile daha kaliteli görüntü elde edileceği açıkça görülmektedir(Çizelge 4.2).

Yapmış olduğumuz uygulama ile çocuğunu kreşe veren bir velinin kreşte bulunan bilgisayarın IP'sini bilmesi halinde işyerindeki bilgisayarından, metroda seyahat eder iken Laptop' undan (kablosuz internet erişimi bulunması halinde) client programını server IP'sini girerek çalıştırması ile takip etmesi mümkün olabilecektir. Buna benzer örnekleri çoğaltmak mümkündür; İşyerinde çalışanlarını izlemek isteyen bir işadamı, öğrencilerini sınava alan bir öğretmenin sınıf dışından sınavı kontrolü, bir ameliyat operasyonunun uzaktan izlenmesi ve farklı hocaların bu ortama dahil olması gibi.

Günümüzde büyük tır filoları ve gemiler uydu aracılığı ile üzerlerine takılan cihazlar ile takip edilebilmektedir. Ancak bu takip harita üzerinde oluşturulan sanal bir güzergah ile algılanabilmektedir. Yapmış olduğumuz uygulamanın bu tır ve gemilerde bulundurulacak bir Laptop' a server kısmının kurulması ve bir USB Webcam' in da seyir güzergahını gösterecek şekilde ayarlanması halinde gerçek olarak takibi sağlanabilecektir. Uydu erişimini pahalı olması bugün için bu uygulamanın kullanımında tek dezavantajdır.

İleriki uygulamalarda Üçüncü Nesil Cep Telefonları ve PDA' lara uygun yazılım yapılması halinde client kısmı tamamen mobil hale gelecek ve PC ve Laptop gibi birimlerin cep telefonu ve el bilgisayarı gibi daha küçük cihazlara dönüşmesi de uygulamayı daha esnek yapacaktır.

USB'nin gelişimi ve kullanımının yaygınlaşması ile ilgili bilgiler de tez içerisinde açıklanmıştır. USB bir Flash Disk ile uygulama rahatlıkla taşınmakta ve tez içerisinde

belirtildiği üzere bir konfigürasyona sahip olan tüm bilgisayarlara kurulabilmektedir.

Tez içerisinde web kameralar ile ilgili bilgilerde verilmiştir. Günümüzde optik band görüntü algılayıcıları için iki farklı yarı iletken teknolojisi kullanılmaktadır. Bunlar CCD (Charge Coupled Device) ve CMOS (Complementary Metal Oxide Semiconductor). Yaklaşık yirmi yıllık bir geçmişi olan CCD teknolojisi, günümüzde artık olgunluğa ulaşmış ve en yüksek çözünürlüklü, en düşük gürültülü görüntü algılayıcılarını üretmek için kullanılan baskın teknolojidir. Henüz birkaç yıllık geçmişi olan CMOS görüntü algılayıcı teknolojisi ise hızlı bir gelişme göstermektedir. CMOS görüntü algılayıcılarının bir kaç yıl içinde performans seviyesi bakımından CCD algılayıcıları yakalaması beklenmektedir.

Kamera teknolojisindeki bu gelişmeler sonucunda ise yakın zamanda yüksek çözünürlük ve minimum gürültülü, ebatları olabildiğince küçük, nerede ise gerçek görüntü veren ürünler bizlere hizmet verecektir.

KAYNAKLAR

AKYOL, B.Ö 1999. Sayısal Görüntü İşlemede Görüntü Karşılaştırma. Gazi Üniversitesi Yüksek Lisans Tezi, Ankara, sI.

ALPARSLAN, E., AYDÖNER, C., CANAN, S., DÖNERTAŞ, A.S., YÜCE, H., KAFAROV, R., ERKAN, B. 2004. Uçakla Digital Fotoğraf Çekimi Çalışmaları. TUBITAK-MAM Yer ve Deniz Bilimleri Araştırma Enstitüsü Çalışmaları Yayınları, s1.

ANONİM, 1998. Hürriyet Dergi Grubu PC Net dergisi. (Ocak 98, Mayıs 98, Ağustos 98, Ocak 99, Kasım 99, Aralık 99)

ANONİM, 1999. TCP/IP Yönlendirme, http://www.ilkertemir.com/document/routing.pdf

ANONİM, 2000.Yeni Binyıl Gazetesi "Binyıl Net" bilgisayar teknolojisi eki (04.05.2000)

ANONİM, 2004. Hipernex.com Elektronik Ticaret, http://www.hipernex.com/ Categories.aspx?CategoryID=12

ANONİM, 2005a. CCD ve CMOS Görüntü Sensörleri, Bağlan Bilgisayar ve İletişim Sistemleri Ltd.Şti., http://www.baglan.com.tr/urunler/axis/camera/ccd_cmos.html

ANONİM, 2005b. USB (universial serial bus-evrensel seri veri yolu), http://www. wekatronik.com/USB1.asp.

ANONİM, 2005c. C ++ Builder, http://www.cikolata.net/ yararli/ programcilik/ c++builder.htm

ANONİM, 2005d. Resim Formatları, Aktif Bilişim ve Danışmanlık Hizmetleri Bilgisayar San. ve Tic. Ltd. Şti., http://www.dijitalarsiv.com/dijitalarsiv.html

ANONİM, 2005e. Amiga and PC, http://amimagazine.8k.com/comments1.htm

ASLAN, K. 2005. C ve Sistem Programcıları Derneği- Windows API Ders Notları- www.cysyem.org s2-7.

ÇAVDAR, O. 2000. Uludağ Üniversitesi Bilgisayar Programcılığı Bilgisayar Ağları Ders Notları .

EROL, A. 1999. Registration of Digital İmages. Dokuz Eylül Üniversitesi Yüksek Lisans Tezi, İzmir,sIII.

GÜLER, B. 1999. Uludağ Üniversitesi Bilgisayar Programcılığı Internet Ders Notları

GÜMÜŞER, M. 2001. Digital Görüntü İşleme. Yıldız Teknik Üniversitesi Yüksek Lisans Tezi, İstanbul, s2.

LEBLEBİCİ, D., AKURGAL, A., GERAY, H., PAYZIN E. ve SARPER S. 2004. Bilgi ve İletişim Teknolojileri Stratejisi – Vizyon 2023 Projesi, Ağustos 2004 ANKARA, s.10-11

ÜNSAL, A., MERT, A., CENGİZ, O.2004. Landsat 7TM Uydu Verileri Kullanılarak Dedegöl Dağı ve Çevresinin Çizgisellik ve Jeolojik Özelliklerinin Belirlenmesi Çalışması.Süleyman Demirel Üniversitesi Uzaktan Algılama Araştırma ve Uygulama Merkezi Isparta, 3. Coğrafi Bilgi Sistemleri (3rd GIS Days in Turkey) , 6-9 Ekim 2004, s4.

YAMAN, K., SARUCAN, A.,ATAK, M., AKTÜRK, N. 2001. Dinamik çizelgeleme için görüntü işleme ve ARIMA modelleri yardımıyla veri hazırlama. Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, Cilt 16, No 1, s19-40.

EKLER

Borland C++ Builder Kodları

Webcam.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        return 0;
}
//---------------------------------------------------------------------------
```

unit1.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <vfw.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

HWND CaptureWindow = NULL;
CAPDRIVERCAPS DriverCaps;
CAPTUREPARMS CapParams;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
```

```
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
HANDLE hwndParent = Form1->Handle;

HANDLE hWndC = capCreateCaptureWindow ( "My Own Capture Window",
   WS_CHILD | WS_VISIBLE , 0, 0, 160, 120, hwndParent, 0);

SendMessage (hWndC, WM_CAP_DRIVER_CONNECT, 0 /* wIndex */, 0L);


capOverlay(hWndC , true);
SendMessage (hWndC, WM_CAP_SEQUENCE, 0, 0L);


}
void __fastcall TForm1::edStopCaptureClick(TObject *Sender)
{
 ReleaseDC(ghCapWnd,MyDc);
 DestroyWindow(ghCapWnd);

}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
// GetBitmapBits(

 CamChecked=False;

 SetPriorityClass(Application->Handle,IDLE_PRIORITY_CLASS );
 mycounter=0;
// MyDc= GetDC(0);
 FullSend=True;
 ScrW=320;
 ScrH=240;
 DivXCount=16;
 DivYCount=12;
 Label3->Caption=IntToStr(DivXCount*DivYCount);

 MyBmp1 = new Graphics::TBitmap;//minimum cell of image to process
 MyBmp1->Width=CellW=(ScrW / DivXCount);
 MyBmp1->Height=CellH=(ScrH / DivYCount);
```

```
MyBmp1->PixelFormat= pf24bit;

 GetDIBSizes(MyBmp1->Handle,headerSize,imageSize);
 pOld=(byte *)malloc(imageSize);
 p1=(byte *)malloc(imageSize);
 p2=(byte *)malloc(imageSize*DivXCount*DivYCount);

 ////////selectting

 char szDeviceName[80];
   char szDeviceVersion[80];

   for(int wIndex = 0; wIndex < 10; wIndex++)
   {
      if(capGetDriverDescription(wIndex, szDeviceName,
         sizeof(szDeviceName), szDeviceVersion, sizeof(szDeviceVersion)))
      {
         // Append name to list of installed capture drivers
         // and then let the user select a driver to use.
         ComboBox1->Items->Add(szDeviceName);
      }
   }

   CapParams.dwRequestMicroSecPerFrame = 4444;
   CapParams.fYield = TRUE;
   CapParams.fCaptureAudio = FALSE;


}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::BtnStartMovieClick(TObject *Sender)
{
       if (CamChecked)
        {
          ReleaseDC(ghCapWnd,MyDc);
          CloseWindow(ghCapWnd);
          DestroyWindow(ghCapWnd);

        }
       CamChecked=True;

          ghCapWnd = capCreateCaptureWindow("OnyurtSW",
            WS_CHILD | WS_VISIBLE, 20, 20,
            ScrW, ScrH,
            Handle, 1);
       MyDc= GetDC(ghCapWnd);
```

```
      // MyDc= GetDC(0);

       SendMessage(ghCapWnd, WM_CAP_DRIVER_CONNECT, StrToInt(EdtCamNo-
>Text) , 0L);
          capPreviewScale(ghCapWnd, false);
       capPreviewRate(ghCapWnd, 80);
          capOverlay(ghCapWnd, 0);
          capPreview(ghCapWnd, 1);
/*
          ghCapWnd2 = capCreateCaptureWindow("OnyurtSW",
            WS_CHILD | WS_VISIBLE, 20, 20,
            ScrW, ScrH,
            Handle, 1);
       SendMessage (ghCapWnd2, WM_CAP_DRIVER_CONNECT, 1,  0L);
          capPreviewScale(ghCapWnd2, 1);
       capPreviewRate(ghCapWnd2, 40);
          capOverlay(ghCapWnd2, 0);
          capPreview(ghCapWnd2, 1);
 */
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BtnCaptureFirstClick(TObject *Sender)
{
 byte *imgp,*imgR,*imgG,*imgB;
 int imgS;
 imgS=ScrW*ScrH*3;/*24 bit*/
 imgp=(byte *) malloc(imgS);
 imgR=(byte *) malloc(imgS);
 imgG=(byte *) malloc(imgS);
 imgB=(byte *) malloc(imgS);

  BitBlt(MyBmp2->Canvas->Handle,0,0,320 ,240,MyDc,0,0,SRCCOPY) ;
  Form1->Canvas->Draw(0,ScrH, MyBmp2);



 GetBitmapBits(MyBmp2->Handle, imgS,imgp);
 int i;
 for (i=0;i<imgS/3;i++)
  {
  *(imgR+3*i)=*(imgp+3*i);
  *(imgG+3*i+1)=*(imgp+3*i+1);
  *(imgB+3*i+2)=*(imgp+3*i+2);
  }
 // we have devide into RGB
 int x,y,xi,yi,ci,cc;
 unsigned int k,ort;
```

```
byte pa[20],pc[20],pv,pvs[20],mv,mi,ca[20] ;
for (x=1;x<ScrW-1;x++)
 for (y=1;y<ScrH-1;y++)
 {
  k=0;
  ort=0;
  cc=0;
  for (xi=-1;xi<2;xi++)
   for (yi=-1;yi<2;yi++)
    {
     pv=*(imgR+3*(x+xi+(y+yi)*ScrW));
    // pv=Filter(pv);

 //    ort=ort+pv;
      ci=0 ;
      ca[k]=pv;
      while ((ci<cc) && (pv!=pvs[ci]))
       ci++;
      if (ci<cc)
       {
       pc[ci]++;
       }
      else
       {
        pvs[ci]=pv;
        pc[ci]=1;
        cc++;
       }
      k++;
     }
/*      ort=ort / k;
     *(imgR+3*(x+y*ScrW))=ort;*/

     mv=0;
     int kk;
     for (kk=0;kk<cc;kk++)
      {
       if (mv<pc[kk])
        {
         mv=pc[kk];
         mi=kk;
        }
      }
      //need to sort
      int j,mi;
      mv=0;
      mi=-1;
```

```
      for (i=0;i<k-1;i++)
       for (j=i+1;j<k;j++)
        {
        if (ca[i]<ca[j])
         {
          mv=ca[i];
          ca[i]=ca[j];
          ca[j]=mv;
         }
        }

        *(imgR+3*(x+y*ScrW))=ca[k/2];
//      *(imgR+3*(x+y*ScrW))=pvs[mi];
//     *(imgR+3*(x+y*ScrW))=pv;

  }



/* SetBitmapBits(MyBmpR->Handle, imgS,imgR);
 Form1->Canvas->Draw(ScrW,ScrH, MyBmpR);

 SetBitmapBits(MyBmpG->Handle, imgS,imgG);
 Form1->Canvas->Draw(ScrW*2,ScrH, MyBmpG);

 SetBitmapBits(MyBmpB->Handle, imgS,imgB);
 Form1->Canvas->Draw(ScrW*2,0, MyBmpB);

 free(imgp);

 free(imgR);
 free(imgG);
 free(imgB);
  */

}



void __fastcall TForm1::BtnCaptureSecondClick(TObject *Sender)
{
  BitBlt(MyBmp2->Canvas->Handle,0,0,160 ,120,MyDc,0,0,SRCCOPY) ;
  Form1->Canvas->Draw(160,200, MyBmp2);

}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::BtnTraceClick(TObject *Sender)
{
int i,j,ccount,k;
bool res;
AnsiString myS;
byte b,c;
int ii,ColorDif;
unsigned int CurrentAddr;
char *MyPackEnd;
int MyPackEndLen,DifCount;
int dif1;
TMemoryStream *PackedStream;
  PackedStream= new TMemoryStream;
  ms = new TMemoryStream();
   LblTime1->Caption=DateTimeToStr(Now());

  CurrentAddr=0;
  ColorDif=StrToInt(edtColorDif->Text);
  DifCount=StrToInt(EdtDiffCount->Text);
  ccount=0;

/*     BitBlt(MyBmp2->Canvas->Handle,0,0,ScrW ,ScrH,MyDc,0,0,SRCCOPY) ;
     GetBitmapBits(MyBmp2->Handle, imageSize*DivXCount*DivYCount,p1);

     ApplyMedianFilter(p1,ScrW,ScrH);
     ApplyMedianFilter(p1+1,ScrW,ScrH);
     ApplyMedianFilter(p1+2,ScrW,ScrH);
     SetBitmapBits(MyBmp2->Handle, imageSize*DivXCount*DivYCount,p1);
  */


     for (j=0;j<DivYCount;j++)
     {
      for (i=0;i<DivXCount;i++)
      {
//         BitBlt(MyBmp1->Canvas->Handle,0,0,CellW ,CellH,MyBmp2->Canvas-
>Handle,i*CellW,j*CellH,SRCCOPY) ;
         BitBlt(MyBmp1->Canvas->Handle,0,0,CellW
,CellH,MyDc,i*CellW,j*CellH,SRCCOPY) ;
         GetBitmapBits(MyBmp1->Handle, imageSize,p1+CurrentAddr);
         ApplyMedianFilter(p1+CurrentAddr,  CellW,CellH);
         ApplyMedianFilter(p1+CurrentAddr+1,CellW,CellH);
         ApplyMedianFilter(p1+CurrentAddr+2,CellW,CellH);


       //  GetBitmapBits(MyBmp1->Handle, imageSize,pOld);
            k=0;
```

```
          dif1=0;
          while ((k<(imageSize/3)))
           {
            if ( (abs(*(p1+CurrentAddr+3*k)-*(p2+CurrentAddr+3*k))>ColorDif ) ||
             (abs(*(p1+CurrentAddr+3*k+1)-*(p2+CurrentAddr+3*k+1))>ColorDif ) ||
             (abs(*(p1+CurrentAddr+3*k+2)-*(p2+CurrentAddr+3*k+2))>ColorDif ) )
            dif1++;
            k++;
           }

    res=dif1<DifCount;
    if ((res) && (! FullSend))
    {
    }
    else
     {
      memmove(p2+CurrentAddr,p1+CurrentAddr,imageSize);
      SetBitmapBits(MyBmp1->Handle, imageSize,p1+CurrentAddr);
      MyBmp1->SaveToStream(ms);
/*     jp->Assign(MyBmp1);
      jp->Compress();
      ms->Clear();
      jp->SaveToStream(ms);*/
      ms->Position=ms->Size;
      myS=cmdEOF;
      ms->Write(myS.data(),myS.Length());
      ms->Position=ms->Size;
      b=i;
      ms->Write(&b,1);
      ms->Position=ms->Size;
      c=j;
      ms->Write(&c,1);
//       ms->Position=0;
//       ms->SaveToFile("c:\w"+IntToStr(i)+IntToStr(j)+".xxx");
      ms->Position=0;
      PackedStream->Position=PackedStream->Size;
      PackedStream->Write(ms->Memory,ms->Size);
      ms->Clear();
      ccount++;
     }
     CurrentAddr+=imageSize;
    }
  }
   PackedStream->Position=0;
   LblTime2->Caption=DateTimeToStr(Now())+"->"+IntToStr(PackedStream->Size);
//  PackedStream->SaveToFile("C:\ZZZZ.TXT");
```

```
  ServerS->Socket->Connections[0]->SendStream(PackedStream);
  Label4->Caption=IntToStr(ccount);
  ServerS->Socket->Connections[0]->SendText("EOS");
  FullSend=true;
  ms->Free();

 // LblTime2->Caption=DateTimeToStr(Now());


}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
 char *ptr1,*ptr2;
 ptr1=(char *)malloc(100);
 ptr2=(char *)malloc(100);
 strcpy(ptr1+4,"deneme");
 strcpy(ptr2,"hayda");

 ShowMessage(ptr1);
 ShowMessage(ptr2);
 memmove(ptr2+5,ptr1+1,5);
 ShowMessage(ptr2);
 if (CompareMem(ptr2+2,ptr1+1,4))
  {
  ShowMessage("Same")   ;
  }
 else
 {
  ShowMessage("Different");
  }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
  TMemoryStream *M1,*M2;
  char *c;
  M1=new TMemoryStream;
  M1->SetSize(10);
  M2=new TMemoryStream;
  M2->SetSize(10);
```

```
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button9Click(TObject *Sender)
{
 char *b1,*b2="abcdefghijklmnopqrstuvwxyz0123456789";
 b1=(char * )malloc(30);
 memmove(b1,b2+5,16);
 ShowMessage(b1);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
 BtnCaptureFirstClick(Sender);
}
byte TForm1::Filter(byte val)
{

//
/*      if (val>150 )
       {
       return 250;
       } else
       return 10;
       /*/


      if (val>200 )
       {
       return 250;
       } else
       if (val>150 )
       {
       return 200;
       } else
       if (val>100 )
       {
       return 150;
       } else
       if (val>50 )
       {
       return 100;
       } else
       {
       return 50;
```

```
      }


}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
/* MyBmpR->FreeImage();
 MyBmpG->FreeImage();
 MyBmpB->FreeImage();*/
 ServerS->Active=False;
 MyBmp1->FreeImage();
 free(p1);
 free(p2);
 ReleaseDC(ghCapWnd,MyDc);
 CloseWindow(ghCapWnd);
 /* free(pr);
 free(pg);
 free(pb);
  */
}
//---------------------------------------------------------------------------

void TForm1::ApplyMedianFilter(byte *val, int w, int h)
{

 int x,y,xi,yi,ci,cc,i;
 unsigned int k,ort;
 byte pa[20],pc[20],pv,pvs[20],mv,mi,ca[20] ;
 for (x=0;x<w;x++)
  for (y=0;y<h;y++)
  {
   k=0;
   ort=0;
   cc=0;
   for (xi=-1;xi<2;xi++)
    for (yi=-1;yi<2;yi++)
     {
     if (((x+xi)>=0) && ((x+xi)<w) && ((y+yi)>=0) && ((y+yi)<h))
     {

     pv=*(val+3*(x+xi+(y+yi)*w));
     if (ChkFilter->Checked)  pv=Filter(pv);
     ci=0 ;
     ca[k]=pv;
     k++;
```

```
      }
      }
       //need to sort
       int j,mi;
       mv=0;
       mi=-1;

       for (i=0;i<k-1;i++)
        for (j=i+1;j<k;j++)
         {
         if (ca[i]<ca[j])
          {
           mv=ca[i];
           ca[i]=ca[j];
           ca[j]=mv;
          }
         }

         *(val+3*(x+y*w))=ca[k/2];
        }


}

void __fastcall TForm1::ServerSAccept(TObject *Sender,
     TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="Accepted";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSClientConnect(TObject *Sender,
     TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="Client Connected";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSClientDisconnect(TObject *Sender,
     TCustomWinSocket *Socket)
{

// StatusBar1->SimpleText="Client Disconnected";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSClientError(TObject *Sender,
```

```
    TCustomWinSocket *Socket, TErrorEvent ErrorEvent, int &ErrorCode)
{
// StatusBar1->SimpleText="Client Error";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSClientRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="Client Read";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSClientWrite(TObject *Sender,
    TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="Client Write";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSGetSocket(TObject *Sender, int Socket,
    TServerClientWinSocket *&ClientSocket)
{

// StatusBar1->SimpleText="Get Socket";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSGetThread(TObject *Sender,
    TServerClientWinSocket *ClientSocket,
    TServerClientThread *&SocketThread)
{
// StatusBar1->SimpleText="Get Thread";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSListen(TObject *Sender,
    TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="Listen";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSThreadEnd(TObject *Sender,
    TServerClientThread *Thread)
{
// StatusBar1->SimpleText="Thread End";
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ServerSThreadStart(TObject *Sender,
      TServerClientThread *Thread)
{
// StatusBar1->SimpleText="Thread Start";
}
//---------------------------------------------------------------------------




void __fastcall TForm1::BtnStartThrClick(TObject *Sender)
{
 MyScaner= new TCamScaner(true);
 MyScaner->Priority = tpLower;
 MyScaner->Resume();

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button3Click(TObject *Sender)
{
/* TMemoryStream *m1,*m2;
 m1= new TMemoryStream;
 m2= new TMemoryStream;
 Memo1->Lines->SaveToStream(m1);
 Memo2->Lines->SaveToStream(m2);
 m1->Position=m1->Size;
 m1->Write(m2->Memory,m2->Size);
 m1->Position=0;
 Memo3->Lines->LoadFromStream(m1);
  */


}
//---------------------------------------------------------------------------

void __fastcall TForm1::BtnNewSenderClick(TObject *Sender)
{
  char *SOB="SOB";//Start of Block
  char *EOD="EOD";//End of Data
  char *EOB="EOB";//End of Block
  byte tb;// for temparary byte
```

```
  TMemoryStream *PackedStream,*ImagesData;
  int dif1;
  int i,j,ii;

  unsigned int CurrentAddr=0;
  int ColorDif=StrToInt(edtColorDif->Text); //color difference in RGB (Red Green Blue)
0-255
  int DifCount=StrToInt(EdtDiffCount->Text);
  int ccount=0,k;
  byte CoorX[102];//Coordination array
  byte CoorY[102];
  int NumOfCell=0;//number of cell need to be sent

  bool res;

  ImagesData= new TMemoryStream;
   Memo1->Lines->Add(FormatDateTime("Start-hh:nn:ss:zzz",Now()));
     for (j=0;j<DivYCount;j++)
     { for (i=0;i<DivXCount;i++)
      {
        BitBlt(MyBmp1->Canvas->Handle,0,0,CellW
,CellH,MyDc,i*CellW,j*CellH,SRCCOPY) ;
        //Gettin Canvas's required cell in  to cell bitmap
        GetBitmapBits(MyBmp1->Handle, imageSize,p1);

        //applyin median filter to R G B

        // renkteki diğer dalgalandırmaların bulanıklaştırılması ile renk kıyaslaması yapılır

        ApplyMedianFilter(p1,  CellW,CellH); //Red
        ApplyMedianFilter(p1+1,CellW,CellH); // Green
        ApplyMedianFilter(p1+2,CellW,CellH); // Blue

        dif1=0;
        k=0;
        //sonuçta üç farklı renk var üçü içinde renk rarklılarını check ediyoz
        //Renk Fark hassaslığı set edilebiliyo
        // p1 şu  anki görüntünün pointer'ı
        //p2 ise bi önceki trace'den kalan bmp

        while ((k<(imageSize/3))&&(dif1<DifCount) )
         {
          //p2 bir önceki kamera görüntüsünün pointer'ı
          //
          if ( (abs(*(p1+3*k)-*(p2+CurrentAddr+3*k))>ColorDif )  ||
               (abs(*(p1+3*k+1)-*(p2+CurrentAddr+3*k+1))>ColorDif )  ||
               (abs(*(p1+3*k+2)-*(p2+CurrentAddr+3*k+2))>ColorDif ) )
```

```
          dif1++;
         k++;
         }

       res=dif1<DifCount;
//       FullSend=True;
       if ((res) && (! FullSend))
       {
        }

       else
       {
        FullSend=0;

        //cell is different then prior so need to store
        memmove(p2+CurrentAddr,p1,imageSize);
        NumOfCell++;//increase the number of different cells
        GetBitmapBits(MyBmp1->Handle, imageSize,pOld);//set into a bmp to convert
JPG
        //we need to use origal bmp cell coz of corruption by median filter
        ImagesData->Write(pOld,imageSize);//all cells are drawn into this image
//        ImagesData->Write(p1,imageSize);//this datas r filtered
        CoorX[NumOfCell-1]=i;
        CoorY[NumOfCell-1]=j;
        if ((NumOfCell>100)|((j==DivYCount-1)&(i==DivXCount-1) ))//maximum pack
size
        {
         /*
          SOB
           ...
           ...
           ...
           image data
           ..
           ..
           ..
          EOD
           ..
           ..
           coordinations of cells
           ..
           ..
          EOB

         */
         PackedStream= new TMemoryStream;
         PackedStream->Write(SOB,3);
```

```
          ImagesData->Position=0;
          MyLBmp = new Graphics::TBitmap;
          MyLBmp->Width=CellW;
          MyLBmp->Height=CellH*NumOfCell;
          MyLBmp->PixelFormat= pf24bit;
          SetBitmapBits(MyLBmp->Handle,NumOfCell*imageSize, ImagesData-
>Memory);
          //creating image pack BMP
          //then need to convert JPG
          jp=new TJPEGImage;
          jp->Assign(MyLBmp);
          jp->CompressionQuality=60;
          jp->Compress();
          jp->SaveToFile("c:/myjpg.jpg");
          // have JPG
          ImagesData->Clear();
          jp->SaveToStream(ImagesData);//store to memory stream
          ImagesData->Position=0;
          PackedStream->Write(ImagesData->Memory,ImagesData->Size);
          PackedStream->Write(EOD,3);
          for (ii=0;ii<NumOfCell;ii++)
          {
            PackedStream->Write(&CoorX[ii],1);
            PackedStream->Write(&CoorY[ii],1);
            }
          //
          PackedStream->Write(EOB,3);
          PackedStream->Position=0;
//          PackedStream->SaveToFile("C:/MYpack.txt");
//          ShowMessage("");
//          ProgressBar1->Position=PackedStream->Size;
//          LblTime1->Caption=IntToStr(PackedStream->Size);
         Memo2->Lines->Add(IntToStr(NumOfCell));
          if (ServerS->Socket->ActiveConnections>0)// there can be option to send all
clients
           {
//            for (int kk=0;kk<ServerS->Socket->ActiveConnections;kk++)
 //            {
              ServerS->Socket->Connections[0]->SendStream(PackedStream);
 //            }
           }
          //reset all datas
          NumOfCell=0;
          MyLBmp->Free();
          jp->Free();
          ImagesData->Clear();
```

```
          }//if numofcell>100

         }
         CurrentAddr+=imageSize;
      }
   }//end of trace loop
 //prepare pack
 Memo1->Lines->Add(FormatDateTime("Finish-hh:nn:ss:zzz",Now()));
 Application->ProcessMessages();
 if ((ServerS->Socket->ActiveConnections>0)& (CbGoOn->Checked))
   BtnNewSenderClick(Sender);

}
//---------------------------------------------------------------------------


void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{
 MyDc=GetDC(0);
}
//---------------------------------------------------------------------------

unit1.h

//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Buttons.hpp>
#include <ScktComp.hpp>
#include <jpeg.hpp>
#include <ComCtrls.hpp>
#include "UnCamScaner.h"
char const *cmdEOF="EOF";
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:   // IDE-managed Components
        TButton *Button1;
        TBitBtn *BitBtn1;
        TButton *Button6;
```

```
TEdit *Edit1;
TButton *Button2;
TButton *Button9;
TLabel *Label1;
TLabel *Label2;
TLabel *Label3;
TLabel *Label4;
TCheckBox *ChkFilter;
TCheckBox *ChkRGB;
TListBox *LstFark;
TEdit *edtColorDif;
TLabel *Label5;
TServerSocket *ServerS;
TEdit *EdtDiffCount;
TButton *BtnStartThr;
TLabel *LblTime1;
TLabel *LblTime2;
TServerSocket *ServerSocket1;
TImage *Image4;
TSpeedButton *SpeedButton1;
TSpeedButton *SpeedButton2;
TEdit *Edit2;
TSpeedButton *SpeedButton3;
TCheckBox *CbGoOn;
TMemo *Memo1;
TMemo *Memo2;
TLabel *Label6;
TLabel *Label7;
TComboBox *ComboBox1;
TLabel *Label8;
TEdit *EdtCamNo;
TLabel *Label9;
TLabel *Label10;
TLabel *Label11;
void __fastcall Button1Click(TObject *Sender);
void __fastcall BtnStartMovieClick(TObject *Sender);
void __fastcall edStopCaptureClick(TObject *Sender);
void __fastcall BtnCaptureFirstClick(TObject *Sender);
void __fastcall FormCreate(TObject *Sender);
void __fastcall BtnCaptureSecondClick(TObject *Sender);
void __fastcall BtnTraceClick(TObject *Sender);
void __fastcall Button2Click(TObject *Sender);
void __fastcall Button4Click(TObject *Sender);
void __fastcall Button9Click(TObject *Sender);
void __fastcall Timer1Timer(TObject *Sender);
void __fastcall FormDestroy(TObject *Sender);
void __fastcall ServerSAccept(TObject *Sender,
```

```
        TCustomWinSocket *Socket);
      void __fastcall ServerSClientConnect(TObject *Sender,
        TCustomWinSocket *Socket);
      void __fastcall ServerSClientDisconnect(TObject *Sender,
        TCustomWinSocket *Socket);
      void __fastcall ServerSClientError(TObject *Sender,
        TCustomWinSocket *Socket, TErrorEvent ErrorEvent,
        int &ErrorCode);
      void __fastcall ServerSClientRead(TObject *Sender,
        TCustomWinSocket *Socket);
      void __fastcall ServerSClientWrite(TObject *Sender,
        TCustomWinSocket *Socket);
      void __fastcall ServerSGetSocket(TObject *Sender, int Socket,
        TServerClientWinSocket *&ClientSocket);
      void __fastcall ServerSGetThread(TObject *Sender,
        TServerClientWinSocket *ClientSocket,
        TServerClientThread *&SocketThread);
      void __fastcall ServerSListen(TObject *Sender,
        TCustomWinSocket *Socket);
      void __fastcall ServerSThreadEnd(TObject *Sender,
        TServerClientThread *Thread);
      void __fastcall ServerSThreadStart(TObject *Sender,
        TServerClientThread *Thread);
      void __fastcall BtnStartThrClick(TObject *Sender);
      void __fastcall Button3Click(TObject *Sender);
      void __fastcall BtnNewSenderClick(TObject *Sender);
      void __fastcall SpeedButton3Click(TObject *Sender);
private:        // User declarations
public:         // User declarations
 HWND ghCapWnd,ghCapWnd2;
  HDC MyDc;
  Graphics::TBitmap *MyBmp1,*MyLBmp;
  Graphics::TBitmap *MyBmp2;
  TMemoryStream* ms;

/*  Graphics::TBitmap *MyBmpR;
  Graphics::TBitmap *MyBmpG;
  Graphics::TBitmap *MyBmpB;*/

//  byte   *MainPointer,*CellPointer;
  byte *p1,*p2,*pOld;//,*pb,*pg,*pr;
  int mycounter,exptime;
  TJPEGImage *jp;
  bool FullSend;
  int CellW,CellH,ScrW,ScrH,DivXCount,DivYCount;
  unsigned int headerSize,imageSize    ;
  bool CamChecked;
```

```
  TCamScaner *MyScaner;


//TMemoryStream* Str1;
//TMemoryStream* Str2;

  byte Filter(byte val);
  void ApplyMedianFilter(byte *val, int w, int h);
  __fastcall TForm1(TComponent* Owner);

};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

uncamscream.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdlib.h>
#include <stdlib.h>

#pragma hdrstop

#include "UnCamScaner.h"
#include "Unit1.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall Unit2::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

__fastcall TCamScaner::TCamScaner(bool CreateSuspended)
        : TThread(CreateSuspended)
{
```

```
}
//---------------------------------------------------------------------------
void __fastcall TCamScaner::Execute()
{
      //---- Place thread code here ----
  char *SOB="SOB";//Start of Block
  char *EOD="EOD";//End of Data
  char *EOB="EOB";//End of Block
  byte tb;// for temparary byte
  TMemoryStream *PackedStream,*ImagesData;
  int dif1;
  int i,j,ii;

  unsigned int CurrentAddr=0;
  int ColorDif=1;//StrToInt(edtColorDif->Text);
  int DifCount=2;//StrToInt(EdtDiffCount->Text);
  int ccount=0,k;
  byte CoorX[32*32+1];
  byte CoorY[32*32+1];
  int NumOfCell=0;

  bool res;
  ImagesData= new TMemoryStream;


      while (1)
       {



       }

}
//---------------------------------------------------------------------------
```

unscrean.h

```
//---------------------------------------------------------------------------

#ifndef UnCamScanerH
#define UnCamScanerH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <stdlib.h>

//---------------------------------------------------------------------------
```

```
class TCamScaner : public TThread
{
private:
protected:
	void __fastcall Execute();
public:
	__fastcall TCamScaner(bool CreateSuspended);
};
//---------------------------------------------------------------------------
#endif
```

PrjClient.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USEFORM("UnClient.cpp", FrmClient);
USEFORM("UnSetting.cpp", FrmSetting);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
	try
	{
		Application->Initialize();
		Application->CreateForm(__classid(TFrmClient), &FrmClient);
		Application->CreateForm(__classid(TFrmSetting), &FrmSetting);
		Application->Run();
	}
	catch (Exception &exception)
	{
		Application->ShowException(&exception);
	}
	return 0;
}
//---------------------------------------------------------------------------
```

UnClient.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include "UnPrinterThread.h"

#pragma hdrstop
```

```
#include "UnClient.h"
#include "UnSetting.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFrmClient *FrmClient;
//---------------------------------------------------------------------------
__fastcall TFrmClient::TFrmClient(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientSocket1Connect(TObject *Sender,
      TCustomWinSocket *Socket)
{

   Settings1->Enabled=true;
  Settings1->Caption="Disconnect..";
  StatusBar1->SimpleText="Connected to "+ClientS1->Address ;
}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientSocket1Connecting(TObject *Sender,
      TCustomWinSocket *Socket)
{
 Settings1->Enabled=False;
 StatusBar1->SimpleText="Connecting to "+ClientS1->Address;
}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientSocket1Disconnect(TObject *Sender,
      TCustomWinSocket *Socket)
{
 StatusBar1->SimpleText="DisConnected";
 Settings1->Enabled=true;
 Settings1->Caption="Reconnect";

}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientSocket1Error(TObject *Sender,
      TCustomWinSocket *Socket, TErrorEvent ErrorEvent, int &ErrorCode)
{
 StatusBar1->SimpleText="Error Connected";
}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientSocket1Lookup(TObject *Sender,
      TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="Look up";
```

```
}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientS1Read(TObject *Sender,
     TCustomWinSocket *Socket)
{
  AnsiString Str,StrSub;
  int yer,i,j;
  char *tmps;
  byte x,y;
/*
    EOF+ColNo+RowNo
*/

//  Memo1->Text=Memo1->Text+Str;
//  jp->LoadFromStream(rs);
// Canvas->Draw(0,0,jp);
  //
  //223 76 66 372

  StrSub=cmdEOF;
  Str= Socket->ReceiveText();
  while (Str.Pos(StrSub)>0)
   {
    yer=Str.Pos(StrSub);
    rs->Position=rs->Size;
    rs->Write(Str.data(),yer);
    Str.Delete(1,yer+StrSub.Length()-1);


    x=*(Str.c_str());
    y=*(Str.c_str()+1);
    Str.Delete(1,2);
    rs->Position=0;
    try
    {
    jp[x][y]->LoadFromStream(rs);
//    Image1->Canvas->Draw(x*10,y*10,jp[x][y]);
    isDrawn[x][y]=False;
    }
   __except(1)
     {
       isDrawn[x][y]=True;
     }

    rs->Clear() ;
   }
  yer=Str.Pos("EOS");
```

```
  if (yer>=0)
  {
    Str.Delete(yer,3);

    for (i=0;i<32;i++)
     for (j=0;j<32;j++)
     {
      if (! isDrawn[i][j])
       {
//      ImgRemoteWin->Canvas->Draw(i*10,j*10,jp[i][j]);
      isDrawn[i][j]=True;
       }
      }
      Str="";
      rs->Clear();

  } else
  {
  rs->Position=rs->Size;
  rs->Write(Str.data(),Str.Length());
  }
  Application->ProcessMessages();

}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::ClientSocket1Write(TObject *Sender,
      TCustomWinSocket *Socket)
{
// StatusBar1->SimpleText="on write";
}
//---------------------------------------------------------------------------
void __fastcall TFrmClient::Button1Click(TObject *Sender)
{
 ClientS1->Active=False;
 ClientS1->Port=StrToInt(FrmSetting->EdPort->Text);
 ClientS1->Address=FrmSetting->EdIP->Text;
 ClientS1->Active=True;


}
//---------------------------------------------------------------------------

void __fastcall TFrmClient::FormCreate(TObject *Sender)
{
 int i,j;
 scCount=0;
 scl=1;
```

```
 rs = new TMemoryStream();
 rsmap = new TMemoryStream();
 for (i=0;i<32;i++)
 for (j=0;j<32;j++)
  {
  jp[i][j] = new Graphics::TBitmap;
  jp[i][j]->Width=10;
  jp[i][j]->Height=10;
  jp[i][j]->PixelFormat= pf24bit;
  isDrawn[i][j]=True;
  }
//  MyReg=new TRegistry();
//  MyReg->CU ("HKEY")

}
//---------------------------------------------------------------------------


void __fastcall TFrmClient::Button3Click(TObject *Sender)
{
   byte b,c;
   int i,j,w=20,h=20;
   byte x,y;
   TJPEGImage *myjp;

   myjp = new TJPEGImage();
   try
   {

   myjp->LoadFromStream(rs);
//   myjp->Scale =jsHalf         ;

   Graphics::TBitmap *MyBmp,*MyBmp2;
   MyBmp= new Graphics::TBitmap;
   MyBmp->Width=w;
//   MyBmp->Height=10680;
   MyBmp->PixelFormat= pf16bit;

   MyBmp->Assign(myjp);

   MyBmp2= new Graphics::TBitmap;
   MyBmp2->Width=w;
   MyBmp2->Height=h;
   MyBmp2->PixelFormat= pf16bit;

   i=0;
   while (rsmap->Position < rsmap->Size)
```

```
   {


        rsmap->Read(&x,1);
        rsmap->Read(&y,1);
       BitBlt(MyBmp2->Canvas->Handle,0,0,w ,h,MyBmp->Canvas-
>Handle,0,i*h,SRCCOPY) ;
//should store in bmp matrix
       Canvas->Draw(x*w,y*h,MyBmp2);
       i++;

    }
    rsmap->Clear();
    rs->Clear();
    myjp->Free();
    MyBmp2->Free();
    MyBmp->Free();

   }
    __except(1)
    {
    rsmap->Clear();
    rs->Clear();
    myjp->Free();
      }
//   Canvas->Draw(0,0,myjp);
//    Refresh();




}
//---------------------------------------------------------------------------

void __fastcall TFrmClient::FormDestroy(TObject *Sender)
{
int i,j;
 for (i=0;i<32;i++)
 for (j=0;j<32;j++)
     jp[i][j]->Free();
}
//---------------------------------------------------------------------------

void __fastcall TFrmClient::NewReader(TObject *Sender,
     TCustomWinSocket *Socket)
{
```

```
  char *SOB="SOB";//Start of Block
  char *EOD="EOD";//End of Data
  char *EOB="EOB";//End of Block
  int pSOB,pEOD,pEOB;
  AnsiString Str;
  Str=Socket->ReceiveText();
  pSOB=Str.Pos(SOB);
  pEOD=Str.Pos(EOD);
  pEOB=Str.Pos(EOB);
  if (pSOB==1)// new pack received
   {
    Str.Delete(1,3);//deleting the header flag
   }
  pEOD=Str.Pos(EOD);//position of EOD
  if (pEOD==0)
   {
    rs->Write(Str.data(),Str.Length());
   }
   else
   {
    rs->Write(Str.data(),pEOD-1);
    rs->Position=0;

    rs->SaveToFile("c:/pack.jpg");

//  rs->Clear();

    Str.Delete(1,pEOD+2);
   }

  pEOB=Str.Pos(EOB);
  if (pEOB!=0) //so pack is compited
   {

    rsmap->Write(Str.data(),pEOB-1)  ;
    rsmap->Position=0;
    Button3Click(Sender);
/*    TPrinterThread *MyThr;
    MyThr= new TPrinterThread(true);
    MyThr->Priority=tpLowest;
    MyThr->Resume();*/
   }
   Application->ProcessMessages();
 }
//---------------------------------------------------------------------------

void __fastcall TFrmClient::Settings1Click(TObject *Sender)
```

```
{
 if (ClientS1->Active)
  {
   ClientS1->Active=False;
  } else
  {


   ClientS1->Active=False;
   ClientS1->Port=StrToInt(FrmSetting->EdPort->Text);
   ClientS1->Address=FrmSetting->EdIP->Text;
   ClientS1->Active=True;


  }
}
//---------------------------------------------------------------------------

void __fastcall TFrmClient::Settings2Click(TObject *Sender)
{

// FrmSetting= new TFrmSetting(this);
 if (FrmSetting->ShowModal() ==mrOk )
  {
   ClientS1->Active=False;

  }

}
//---------------------------------------------------------------------------
```

unClient.h

```
//---------------------------------------------------------------------------

#ifndef UnClientH
#define UnClientH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ScktComp.hpp>
#include <jpeg.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
```

```
#include <Menus.hpp>
#include <Registry.hpp>
char const *cmdEOF="EOF";
int const w=20;
int const h=20;

//---------------------------------------------------------------------------
class TFrmClient : public TForm
{
__published:    // IDE-managed Components
        TClientSocket *ClientSocket1;
        TStatusBar *StatusBar1;
        TButton *Button1;
        TButton *Button2;
        TButton *Button3;
        TClientSocket *ClientS1;
        TMemo *Memo1;
        TLabel *Label3;
        TProgressBar *PbStatus;
        TMainMenu *MainMenu1;
        TMenuItem *Settings1;
        TMenuItem *Settings2;
        void __fastcall ClientSocket1Connect(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall ClientSocket1Connecting(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall ClientSocket1Disconnect(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall ClientSocket1Error(TObject *Sender,
          TCustomWinSocket *Socket, TErrorEvent ErrorEvent,
          int &ErrorCode);
        void __fastcall ClientSocket1Lookup(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall ClientS1Read(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall ClientSocket1Write(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall FormDestroy(TObject *Sender);
        void __fastcall NewReader(TObject *Sender,
          TCustomWinSocket *Socket);
        void __fastcall Settings1Click(TObject *Sender);
        void __fastcall Settings2Click(TObject *Sender);
private:        // User declarations
public:         // User declarations
```

```
        int scCount;
        int scl;
        TMemoryStream *MyStream;
        TMemoryStream *rs,*rsmap;
        TRegistry *MyReg;
//        TJPEGImage *jp[32][32];
        Graphics::TBitmap  *jp[32][32];
        bool isDrawn[32][32];
      __fastcall TFrmClient(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFrmClient *FrmClient;
//---------------------------------------------------------------------------
#endif
```

UnPrinterThread.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "UnPrinterThread.h"
#include "UnClient.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall Unit1::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

__fastcall TPrinterThread::TPrinterThread(bool CreateSuspended)
      : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall TPrinterThread::Execute()
{
```

```
        //---- Place thread code here ----
        double scl=FrmClient->scl;
        Graphics::TBitmap *MyBmp,*MyBmp2,*MyBmpStr;
        TMemoryStream *MyRs,*MyMap;
        byte b,c;
        int i,j;//w=FrmClient->w,h=FrmClient->h;
        byte x,y;
        TJPEGImage *myjp;
        myjp = new TJPEGImage();


        MyRs= new TMemoryStream;
        MyMap= new TMemoryStream;


        FrmClient->rs->Position=0;
        FrmClient->rsmap->Position=0;

        MyRs->CopyFrom(FrmClient->rs,FrmClient->rs->Size );
        MyMap->CopyFrom(FrmClient->rsmap,FrmClient->rsmap->Size );

        MyMap->Position=0;
        MyRs->Position=0;


   myjp->LoadFromStream(MyRs);
//   myjp->SaveToFile("c:/jj.jpg");

   MyBmp= new Graphics::TBitmap;
   MyBmp->Assign(myjp);

   MyBmp2= new Graphics::TBitmap;
   MyBmp2->Width=w;
   MyBmp2->Height=h;
   MyBmp2->PixelFormat= pf16bit;

   MyBmpStr= new Graphics::TBitmap;
   MyBmpStr->Width=w*scl;
   MyBmpStr->Height=h*scl;
   MyBmpStr->PixelFormat= pf16bit;
//   FrmClient->inThread=false;

  // SetStretchBltMode(MyBmp2->Canvas->Handle,HALFTONE);
  // SetStretchBltMode(MyBmpStr->Canvas->Handle,HALFTONE);
```

```
  i=0;
  while (MyMap->Position < MyMap->Size)
  {
     MyMap->Read(&x,1);
     MyMap->Read(&y,1);
     BitBlt(MyBmp2->Canvas->Handle,0,0,w,h,MyBmp->Canvas-
>Handle,0,i*h,SRCCOPY) ;
//      StretchBlt(MyBmpStr->Canvas->Handle,0,0,w*scl,h*scl,
//      MyBmp2->Canvas->Handle,0,0,w,h,SRCCOPY);
     FrmClient->Canvas->Draw(x*w*scl,y*h*scl,MyBmp2);
//      MyBmp2->SaveToFile("c:/bmp"+IntToStr(x)+IntToStr(y)+".jpg");
     FrmClient->PbStatus->Position!=FrmClient->PbStatus->Position;

//      FrmClient->Image1->Canvas->Draw(x*w*scl,y*h*scl,MyBmp);
     i++;
   }

   MyMap->Free();
   MyRs->Free();
   myjp->Free();
   MyBmp2->Free();
   MyBmp->Free();
   MyBmpStr->Free();

}
//---------------------------------------------------------------------------

UnPrinterThread.h

//---------------------------------------------------------------------------

#ifndef UnPrinterThreadH
#define UnPrinterThreadH
//---------------------------------------------------------------------------
#include <Classes.hpp>

//---------------------------------------------------------------------------
class TPrinterThread : public TThread
{
private:
protected:
      void __fastcall Execute();
public:
      __fastcall TPrinterThread(bool CreateSuspended);
};
//---------------------------------------------------------------------------
```

```
#endif
```

UnSetting.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "UnSetting.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFrmSetting *FrmSetting;
//---------------------------------------------------------------------------
__fastcall TFrmSetting::TFrmSetting(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

UnSetting.h

```
//---------------------------------------------------------------------------
#ifndef UnSettingH
#define UnSettingH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TFrmSetting : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TEdit *EdIP;
        TEdit *EdPort;
        TBitBtn *BitBtn1;
        TBitBtn *BitBtn2;
private:        // User declarations
public:         // User declarations
        __fastcall TFrmSetting(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFrmSetting *FrmSetting;
//---------------------------------------------------------------------------
#endif
```

ÖZGEÇMİŞ

Kadir AKTEPE 1967 yılında Kahramanmaraş'ta doğdu. İlk öğrenimini Kahramanmaraş İnönü İlkokulu'nda, orta öğrenimini Kahramanmaraş Merkez Ortaokulu'nda ve liseyi Kahramanmaraş Sütçü İmam Lisesi'nde tamamladı. 1986 yılında O.D.T.Ü Gaziantep Mühendislik Fakültesi Elektrik-Elektronik Mühendisliği Bölümüne girdi. 1992 yılında aynı bölümden Elektrik-Elektronik Mühendisi unvanı alarak mezun oldu. Yaklaşık bir yıl kadar Gaziantep'te özel bir şirkette çalıştı. 1993 yılında PTT'nin açmış olduğu sınava katıldı ve aynı yıl PTT'de göreve başladı. 1995 yılında PTT'nin Türk Telekomünikasyon A.Ş. ve Posta İşletmesi Genel Müdürlüğü olarak ikiye ayrılması sonucunda Türk Telekomünikasyon A.Ş.'de kaldı. 2004 yılında Anadolu Üniversitesi İktisat Fakültesi Kamu Yönetimi Bölümünü bitirdi. 2000-2005 yılları arasında Arslanbey Telekom Müdürlüğü görevini yürüttü. Görevi gereği eğitim amaçlı olarak İngiltere'de bulundu. Halen aynı şirkette Santral Şef Mühendisi olarak çalışmakta olup, 2002 yılında Kahramanmaraş Sütçü İmam Üniversitesi Fen Bilimleri Enstitüsü Elektrik Elektronik Mühendisliği Ana Bilim Dalı'nda Yüksek Lisansa başladı.