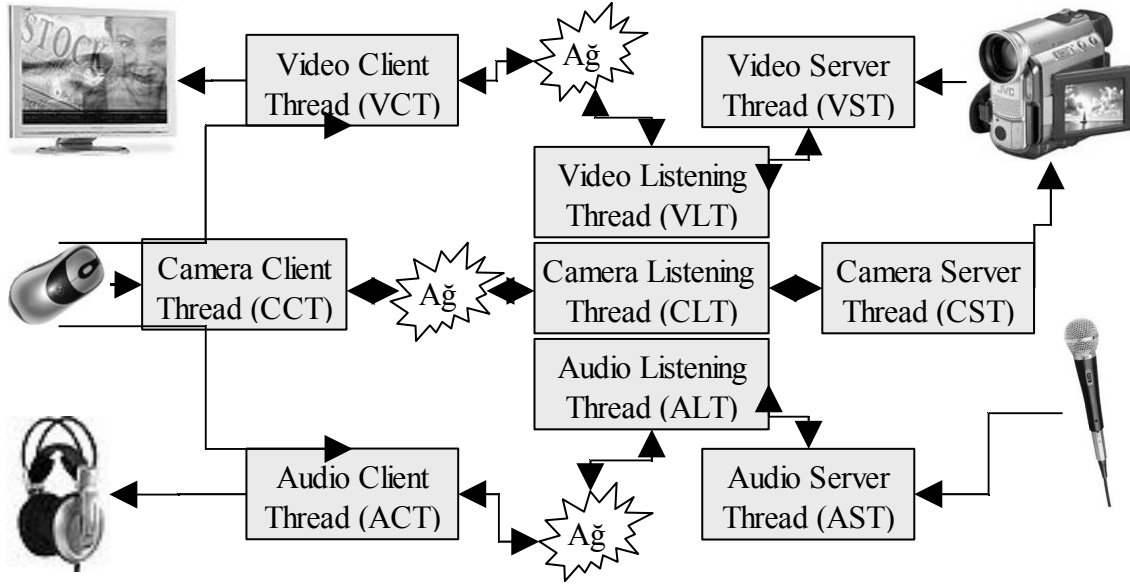


3.BÖLÜM

GELİŞTİRİLEN SİSTEMİN YAPISI



Şekil 3.1 Geliştirilen sistemin temel yapısı

Geliştirilen sistemin temel yapısı şekil 3.1’de görülmektedir. Anlaşıldığı üzere sistem istemci ve sunucu olmak üzere iki ana parçadan oluşmaktadır. Sunucu tarafında bir kamera aracılığıyla video ile mikروفon aracılığıyla ses alınarak mevcut ağ üzerinden istemciye iletilir. Mevcut ağın türü için herhangi bir sınırlama yoktur (örneğin yerel ağ veya internet) ve iletim protokolü olarak TCP/IP kullanılmıştır. Bunun yanında sunucuya birden fazla sayıda istemci bağlanabilir ve sunucu istemcilerin her birine aynı anda hizmet verebilir.

Geliştirilen sistemin basitçe istek sürülen bir sistem olduğu söylenebilir. Temel olarak istemci sunucudan çeşitli isteklerde bulunur ve sunucu bu isteklere karşılık uygun birimi ile gerekli hizmeti verir. İstekler ve hizmetler temelde üç ana başlık altında toplanmışlardır: video, ses ve kamera. Aşağıda tam listesi verilmiş olan mevcut istekler “istekler.h” dosyasında tanımlanmışlardır.

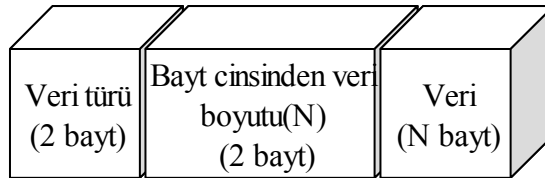
- Video istekleri
 - START_VIDEO
 - STOP_VIDEO
 - GRAYSCALE
 - COLOR
- Ses istekleri
 - START_AUDIO
 - STOP_AUDIO
- Kamera istekleri

- UP
- DOWN
- LEFT
- RIGHT
- UP_LEFT
- UP_RIGHT
- DOWN_LEFT
- DOWN_RIGHT
- WHITE_INC
- WHITE_DEC
- ZOOM_OUT
- ZOOM_IN
- FOCUS_INC
- FOCUS_DEC
- STOP_CAMERA
- POSITION
- RESOLUTION

Sunucu tarafındaki kamera hareketli motorlara sahiptir ve böylece istemciden gelen isteklere göre herhangi bir yöne yönelebilir ya da yakınlaştırma ve odaklama gibi çeşitli işlemler gerçekleştirebilir. Bu ise sistemin kullanıcı ile etkileşimini artırır. Ayrıca kullanıcının gerektiğinde çözünürlüğü de değiştirebilmesi sistemin kullanıcı dostu olma özelliğini pekiştirir.

Video, ses ve kamera işlemleri birbirlerini kesmemeleri (bloklamamaları) için ayrı modüllere (threadlere) yerleştirilmişlerdir. Bu modüller video işlemlerini gerçekleştiren Video Server Thread (VST), ses işlemlerini gerçekleştiren Audio Server Thread (AST) ve kamera işlemlerini gerçekleştiren Camera Server Thread (CST) modülleridir.

Sistem tasarlanırken, sokete veri yazma ve soketten veri okuma gibi o andaki threadi bloklayan işlemlerin ayrı threadlere dağıtılmasına ve kritik bölge içerisinde en az süre ile kalmak için bu bölgelerden, ekrana çizim yapmak gibi uzun zaman alan işlemlerin çıkartılmasına dikkat edilmiştir. Aynı zamanda video, ses ve kamera işlemlerinin ayrı threadlere parçalanması en az bloklanmayı sağlayarak en etkin çözümü vermiştir.



Şekil 3.2 Veri iletim protokolü

Sunucu ile istemci arasında kullanılan veri iletim protokolü şekil 3.2’de görüldüğü gibi verinin iki bayt uzunluğundaki veri türü ve yine iki bayt uzunluğundaki bayt cinsinden veri boyutu bilgilerini takip ettiği bir biçimdedir. Eğer verinin boyutu N bayt ise, protokol için gerekli bilgiler eklendikten sonra istemciye gönderilmeye hazır olan verinin boyutu N+4 bayt olur. Gönderilecek veriyi bu protokole uygun hale getirmekten sorumlu olan metot SendDataEx() metodudur. Bu metot SendData() metodunun üzerinde bir katman olarak işlev görür. SendData() metodu ise istenilen verinin istemciye tamamen iletilmesinden sorumludur.

Aşağıda tam listesi verilmiş olan ve kullanılan protokol için gerekli veri türleri “datatypes.h”

dosyasında tanımlanmışlardır. Söz konusu veri türlerinin açıklamaları ilerleyen bölümlerde yapılacaktır.

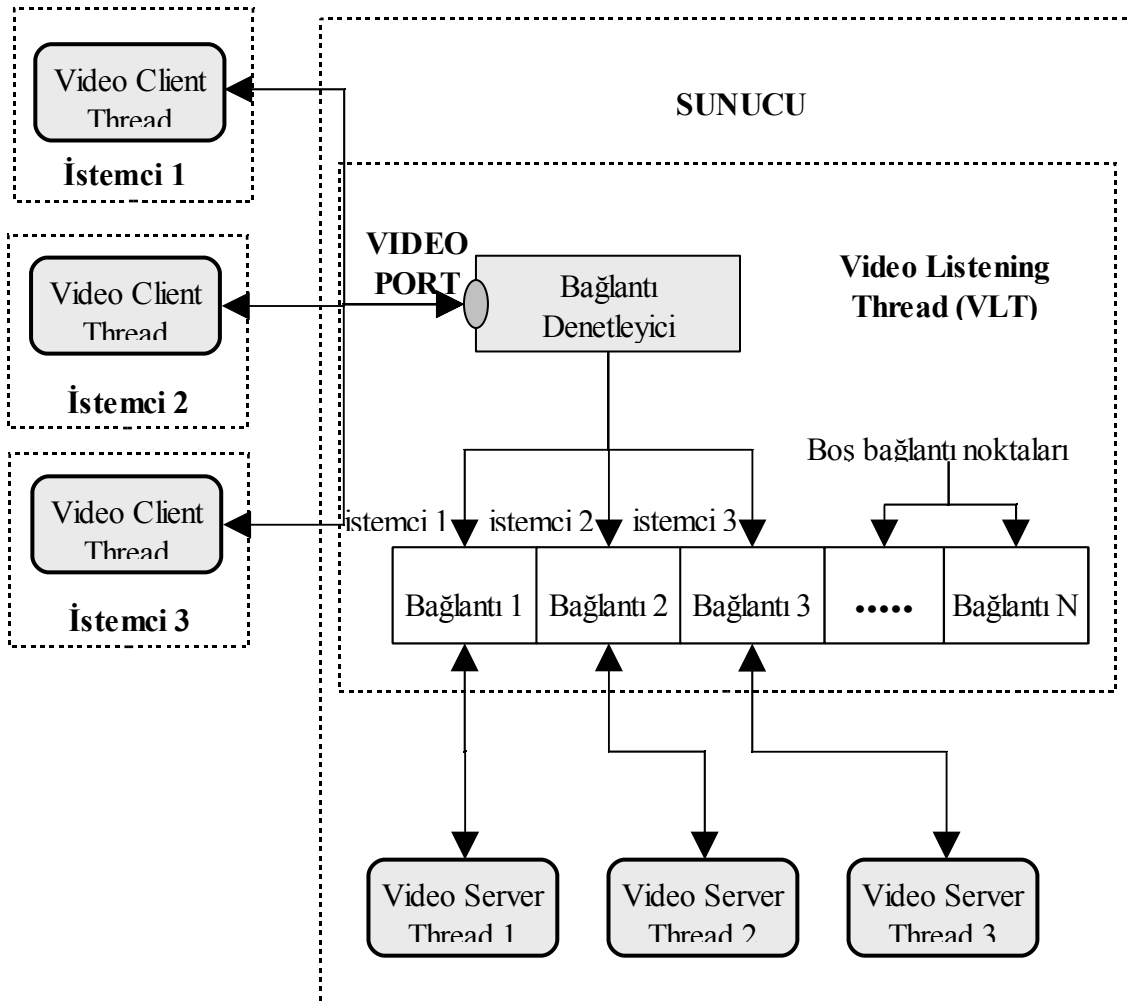
- BMP_HEADER
- FULL_COLOR_FRAME
- FULL_GRAYSCALE_FRAME
- BLOCKED_GRAYSCALE_FRAME
- BLOCKED_COLOR_FRAME
- AUDIO_FRAME
- PING

Sunucu başlatıldığında üç ana modül aktif hale geçer. Bu modüller video bağlantılarını bekleyen Video Listening Thread (VLT), ses bağlantılarını dinleyen Audio Listening Thread (ALT) ve kamera bağlantılarını dinleyen Camera Listening Thread (CLT) modülleridir. Bunların dışında sistem; VLT modülünün denetiminde bir veya daha fazla VST modülü, ALT modülünün denetiminde bir veya daha fazla AST modülü ve CLT modülünün denetiminde bir CST modülü (kamerayı aynı anda en çok bir istemci kontrol edebilir) içerir.

İstemci tarafında ise kullanıcının isteğine göre başlatılan ya da sonlandırılan; video işlemleri için Video Client Thread (VCT), ses işlemleri için Audio Client Thread (ACT) ve kamera işlemleri için Camera Client Thread (CCT) modülleri bulunmaktadır.

Yukarıda değinilen tüm modüller ayrıntıları ile aşağıda anlatılmıştır.

Video Listening Thread (VLT) modülü



Şekil 3.3 VLT modülünün genel yapısı

Yapısı şekil 3.3'te görülen VLT modülünün birincil görevi, kendisine yaratılırken parametre ile gönderilen video portunu (geliştirdiğim projede 5965) sonsuz döngü içerisinde dinlemek ve gelen bağlantı isteklerine gerekli hizmeti vermektir.

Genelde (geliştirdiğim proje de dahil olmak üzere) bir sunucu yalnız bir VLT modülüne ihtiyaç duyar. Bunun yanında koşulların gerektirmesi halinde her biri farklı portu dinleyen birden fazla VLT modülü yaratmak mümkündür. Burada dikkat edilmesi gereken nokta bir portu dinleyen en çok bir VLT modülünün olabileceğidir.

Video portunu dinleyen VLT modülüne bir bağlantı isteği geldiğinde, modül bağlantıyı boş bağlantı noktalarından birine yerleştirir. Eğer hiç boş bağlantı noktası kalmamışsa bağlantı isteği reddedilir. Her bağlantı için yeni bir VST (Video Server Thread) modülü yaratılır ve yaratılan her modül kendine ait bağlantıya hizmet eder. Herhangi bir bağlantı sonlandığında o bağlantıya ait bağlantı noktası boşaltılır ve ona hizmet eden VST modülü yok edilir.

Bağlantı noktaları bir integer dizidir. Başlangıçta bu dizinin tüm elemanları 0'dır. Herhangi bir bağlantı noktasına bir bağlantı yerleştirildiğinde değeri 1 olarak değiştirilir. Her bir bağlantıya ait VST modülüne bu bağlantı noktası aracılığıyla erişilir. Daha detaylı ifade edecek olursak; bağlantı noktaları connections dizisi ile ve bu bağlantılara hizmet eden VST modülleri de videoServerThread dizisi ile ifade edilsin.

```
int connections[MAX_CONNECTIONS];  
VST *videoServerThread[MAX_CONNECTIONS];
```

MAX_CONNECTIONS maksimum istemci (bağlantı) sayısına karşılık gelir ve değeri geliştirdiğim projede 10 olarak belirlenmiştir. Ondan fazla istemciye video akışı sağlamak sistemi çok zorlayacak ve sunucudan yeterli verim alınamamasına neden olacaktır. Ancak gelişen sistemlerle birlikte bu değer zamanla yükseltilebilecektir.

Bir bağlantı isteği geldiğinde bu bağlantı connections dizisi içerisinde bulunan ilk boş yere yerleştirilir. Gelen bağlantının connections dizisinin sıfırıncı elemanına yerleştirildiğini varsayarsak bu durumda connections[0] değeri 1 yapılır ve videoServerThread[0] elemanına yer ayrılır.

```
connections[0] = 1;  
videoServerThread[0] = new VST;
```

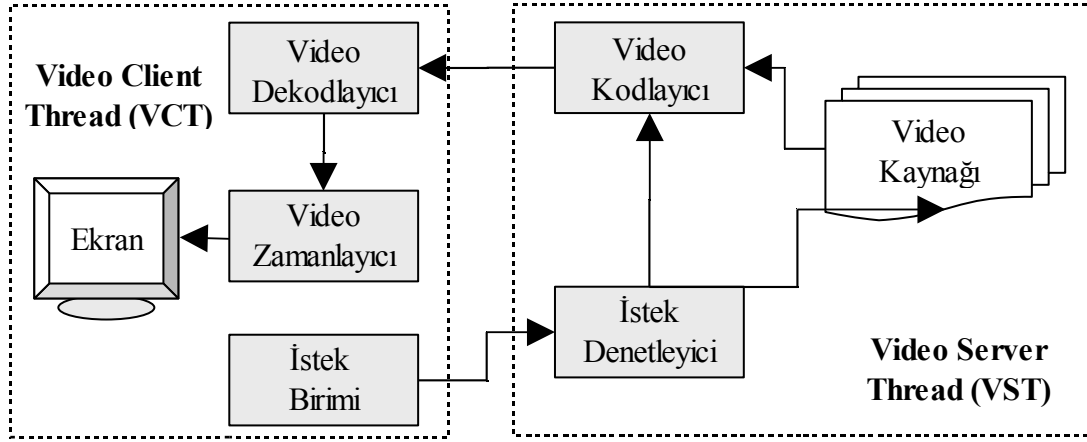
Yaratılan VST modülü sonlanırken bellekte kendine ayrılan alanı boşaltmalı ve connections dizisinde kendine ait elemanı 0 yapmalıdır. VST modülü bağlantı noktasını boşaltmak için VLT modülü içerisindeki FreeConnection() metodunu kullanır.

Bağlantı dizisi içinde boş yer arama işlemini ise VLT modülü içerisinde bulunan FindFreeID() metodu üstlenmiştir. Bu metod connections dizisi içerisinde boş yer varsa bulduğu ilk boş alanın indeksini, aksi halde MAX_CONNECTIONS döndürür.

VLT modülü video portunu dinlemekte iken gelen her bağlantı isteği için accept() fonksiyonu yeni bir socket numarası döndürür. Eğer bağlantı için uygun bir bağlantı noktası bulunabilirse bu socket numarası bağlantıya hizmet edecek olan VST modülüne parametre olarak iletilir. Ancak hiç boş bağlantı noktası kalmaması halinde socket kapatılır.

Bu modülün dışında ses portunu (5966) dinleyen ALT modülü ve kamera portunu (5967) dinleyen CLT modülü de aynı temel mantık üzerine kurulmuşlardır. Aradaki tek fark istemci tarafındaki modüllerin ve sunucu tarafında bu modüllere hizmet veren modüllerin farklı oluşudur. Ses işlemleri için istemci tarafında ACT modülü ile sunucu tarafında AST modülü ve kamera işlemleri için istemci tarafında CCT modülü ile sunucu tarafında CST modülü kullanılır.

Video Server Thread (VST) modülü



Şekil 3.4 VST modülünün genel yapısı

Genel yapısı şekil 3.4'te verilen VST modülü, kendisine bağlı olan VCT modülünün isteklerine gerekli hizmeti vermekle yükümlüdür. Aşağıda bir listesi verilmiş olan bu istekler "istekler.h" dosyasında tanımlanmıştır.

- **START_VIDEO**
Video akışını başlat
- **STOP_VIDEO**
Video akışını durdur
- **GRAYSCALE**
Gri seviye video akışı
- **COLOR**
Renkli video akışı

Bu isteklere cevap ya da hizmet amacıyla VST modülü kendisine bağlı VCT modülüne çeşitli veriler gönderir. Bu verilerin aşağıda belirtilmiş olan türleri "datatypes.h" adlı dosyada tanımlanmıştır.

- **BMP_HEADER**
Video format bilgisi
- **FULL_COLOR_FRAME**
Ham, renkli çerçeve
- **FULL_GRAYSCALE_FRAME**
Ham, gri seviye çerçeve
- **BLOCKED_GRAYSCALE_FRAME**
Sıkıştırılmış gri seviye çerçeve
- **BLOCKED_COLOR_FRAME**
Sıkıştırılmış renkli çerçeve
- **PING**
İstemcinin bağlı olup olmadığını kontrol etmek amacıyla gönderilen test verisi

Yukarıda belirtilen veriler VCT modülüne SendDataEx() metodu ile gönderilirler. Bu metod, SendData() metodunun üstünde bir katman olarak işlev görür ve veriyi sunucu ile istemci arasındaki

protokole uygun biçime sokarak istemciye gönderme görevini üstlenmiştir.

VST modülüne START_VIDEO isteği geldiğinde, bu modül hizmet verdiği VCT modülüne ilk olarak o anda video kaynağından alınmakta olan görüntünün formatını bildirir. Söz konusu format gerçekte bir BITMAPINFOHEADER yapısıdır. Bu yüzden gönderilecek verinin türü adlandırılırken BMP_HEADER ifadesi kullanılmıştır. Video kaynağından o anda yakalanmakta olan görüntünün eni, boyu, boyutu ve piksel başına düşen bit sayısı gibi özelliklerini belirten BITMAPINFOHEADER yapısı aşağıdaki üyelerden oluşur.

- **biSize**
Yapının bayt cinsinden boyutunu saklar.
- **biWidth**
Görüntünün piksel cinsinden enini belirtir.
- **biHeight**
Görüntünün piksel cinsinden boyunu belirtir. Eğer biHeight pozitif bir değer ise görüntü aşağıdan yukarıya kodlanmıştır. Yani başlangıcı sol alt köşedir. biHeight negatif olduğunda ise görüntü yukarıdan aşağıyadır ve başlangıcı sol üst köşedir.
- **biPlanes**
Hedef aygıtın düzlem sayısını belirtir. Değeri 1 olmalıdır.
- **biBitCount**
Piksel başına düşen bit sayısını belirtir. biBitCount üyesi her bir pikseli tanımlayan bit sayısını ve görüntüdeki maksimum renk sayısını belirler. Örneğin sekiz bitlik bir görüntüde maksimum renk sayısı 256, yirmi dört bitlik bir görüntüde ise 16,777,216'dır (2^{24}).
- **biCompression**
Kullanılan sıkıştırma türünü belirtir. Bu değer 0 ise görüntü sıkıştırılmamıştır.
- **biSizeImage**
Resmin bayt cinsinden boyutunu belirtir.
- **biXPelsPerMeter**
Yatay çözünürlüğü belirtir.
- **biYPelsPerMeter**
Dikey çözünürlüğü belirtir.
- **biClrUsed**
Mevcut renk indislerinden görüntü tarafından kullanılanların sayısını içerir. Eğer bu değer 0 ise görüntü biBitCount üyesinin ifade ettiği maksimum renk sayısını kullanır.
- **biClrImportant**
Görüntüyü göstermek için gereken renk indislerinin sayısını belirtir. Eğer bu değer 0 ise, tüm renkler gereklidir.

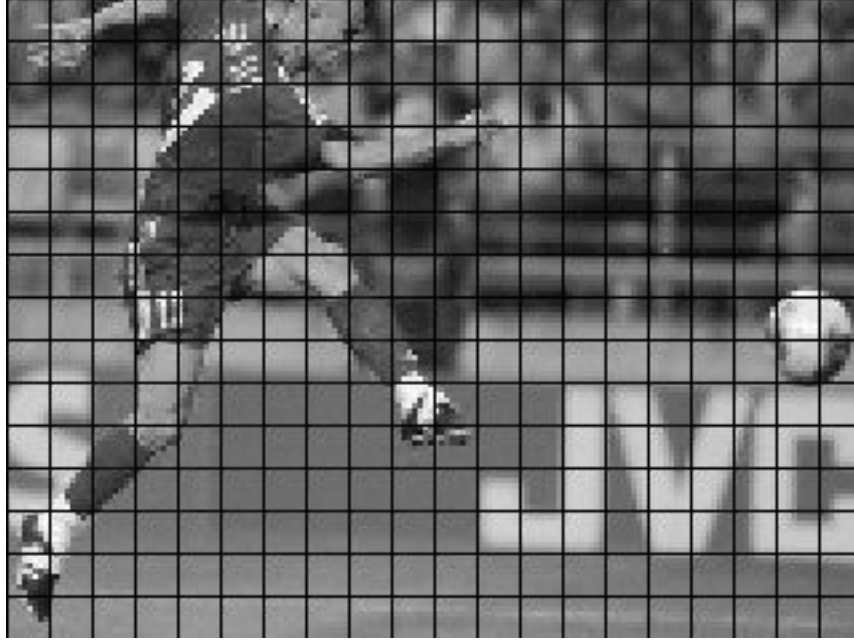
Geliştirdiğim projede renk uzayı olarak YV12 seçilmiştir. Bu renk uzayı için BITMAPINFOHEADER yapısının gerekli üyelerinin alacağı değerler aşağıda verilmiştir.

- **biBitCount** = 12
- **biCompression** = 808596553
- **biSizeImage** = $biWidth * biHeight * biBitCount / 8$;

Sunucu tarafında video formatı değiştiğinde, bağlı istemciler bu değişiklikten BMP_HEADER verisi aracılığıyla haberdar edilirler. Bu şekilde istemcinin koşma esnasında oluşan değişikliklere adapte olması sağlanır.

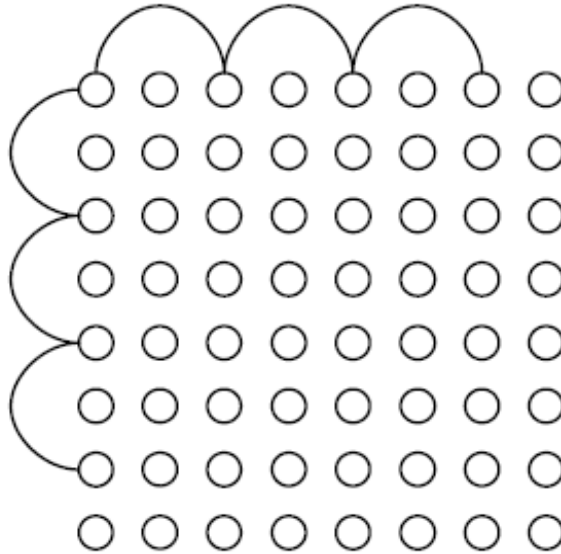
Görüntü formatı VCT modülüne gönderildikten sonra video kaynağından okunan çerçeveler sıkıştırılarak bu modüle gönderilmeye başlanır. Sıkıştırma işlemi video sıkıştırma bölümünde ayrıntılı olarak anlatılmıştır. İşlem iki temel adımdan oluşur: ardışık çerçeveler arasındaki değişimlerin bulunması ve bu değişimlerin JPEG kullanılarak sıkıştırılması.

Değişimlerin bulunması için çerçeveler 8x8'lik bloklara ayrılırlar. Şekil 3.5'te 160x120 boyutlarında bir görüntü üzerinde 300 adet 8x8 piksellik blok görülmektedir. Daha sonra ardışık iki çerçevenin aynı yerdeki blokları karşılaştırılır ve belli eşik değerlerine göre bloğun değişip değişmediğine karar verilir. Bahsedilen eşik değerleri herhangi bir pikselin değiştiğine karar vermek için gerekli piksel fark eşiği ve bloğun değiştiğine karar vermek için gerekli blok fark eşiğidir. Başka bir deyişle aynı uzaysal konumda bulunan piksel genlikleri arasındaki fark piksel fark eşiğinden büyükse o piksel değişmiş olarak işaretlenir ve herhangi bir blok içinde değişen piksellerin sayısı blok fark eşiğini aşmışsa bu blok değişmiş kabul edilir ve karşı tarafa gönderilir.



Şekil 3.5 Üzerinde 300 adet 8x8 piksellik blok bulunan görüntü

Söz konusu işlemleri gerçekleştirmek amacıyla çerçevenin tutulduğu diziden, verilen blok numarasına göre blok bilgisini okuyan GetBlockData() ve verilen iki bloğu test edip değişim olup olmadığına karar veren CompareBlocks() metotları kullanılır. CompareBlocks() metodu kendisine gelen blokları şekil 3.6'da gösterildiği gibi bloğun yarı çözünürlüğünde örnekleyerek işlem yükünü azaltır.



Şekil 3.6 Blokların örnekleme

Blok bit haritası

Değişmiş olarak kabul edilen bloklar JPEG ile sıkıştırılarak aralarına herhangi bir blok numarası belirteci koyulmaksızın bir bellek alanına ardışık olarak yazılırlar. Değişmeyen bloklar için ise istemciye herhangi bir bilgi gönderilmez. Ama siz de tahmin edersiniz ki istemcinin gelen veriyi ekranda gösterebilmesi için aldığı verilerin hangi bloğa ait olduğunu bilmesi gerekir. Bu noktada bant genişliği sınırlamaları göz önüne alınarak iletilecek veri miktarını azaltmak amacıyla bir blok bit haritası gerçekleştirilmiştir. Bu harita blok sayısı kadar bit sayısına sahiptir. Değişen blokların harita üzerindeki bit alanı 1 olarak, değişmeyen blokların bit alanı ise 0 olarak işaretlenir.

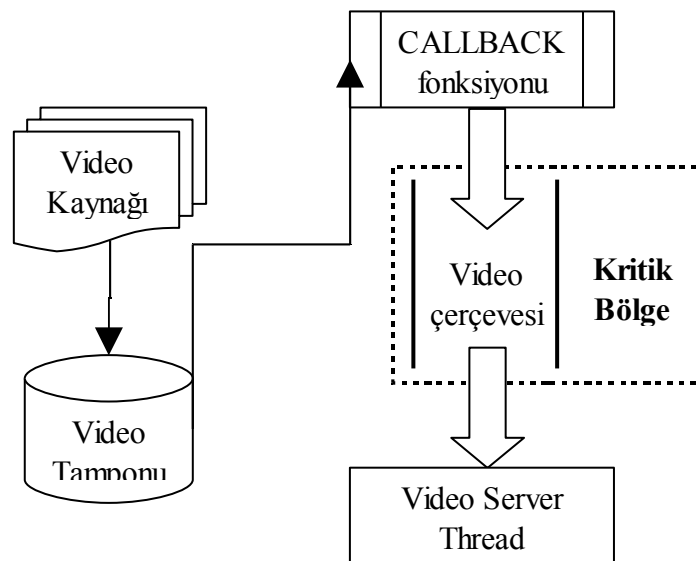
Örnek olarak elimizdeki görüntünün 16 bloktan oluştuğunu düşünelim. Bu durumda bit haritası $16/8 = 2$ bayt uzunluğunda olacaktır. Görüntü üzerinde yalnızca 3,5,7,11 ve 13. blokların değiştiğini varsayarsak istemciye gönderilecek iki bayt uzunluğundaki blok bit haritası aşağıdaki gibi olacaktır.

0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0

Blok bit haritası, değişen blokların saklandığı dizinin başına eklenerek istemciye gönderilir. İstemci önce blok haritasını okur ve daha sonra kendisine ulaşan diziden blokları sıra ile okumaya başlar. Okunan blok verisinin hangi blok numarasına ait olduğunu belirlemek için blok bit haritası üzerine değişen blok olarak işaretlenmiş, sıradaki ilk bloğu arar. Bulduğu blok numarası diziden okuduğu bloğa aittir.

300 bloktan oluşan 160×120 çözünürlüğünde bir görüntü için blok bit haritası $300/8 = 37.5$ (38) bayt uzunluğunda olmaktadır. Her bir blok numarası iki bayt ile ifade edilseydi 300 blok için 600 bayta ihtiyaç duyulacaktı. Bu değer değişen blokların sayısına bağlı olsa da ortalama olarak görüntünün %50'sinin değiştiği göz önüne alınırsa (150 adet blok değişmişse) gereken bayt sayısı 300 olacaktır. Bu durumda blok bit haritası kullanılarak yaklaşık olarak 7:1 oranında sıkıştırma sağlandığı söylenebilir.

VST modülünün video verisine erişimi



Şekil 3.7 Video verisine erişim

Sunucuya baęlı video kaynaęından sunucunun birincil threadi tarafından okunan video verisi bir bayt dizisine yazılır. Bu yazma iřlemi video tamponunun her doluřunda gerekleřir. Örneęin video kaynaęından 25 ereve/saniye oranında video yakalanıyorsa her 40ms’de bir ereve yenilenecek ya da bařka bir deyiřle yazma iřlemi gerekleřecektir. Dięer taraftan mevcut VST modülleri gerektięi zaman ereve bilgisine eriřmek istediklerinde bu diziden okuma yapacaklardır. Bu yüzden video verisi kritik bölge olarak tanımlanmıřtır. Bu řekilde, video verisini okuyan VST modüllerinin ve video kaynaęından görüntü yakalayıp video dizisine yazan birincil threadin video dizisine aynı anda eriřmeleri engellenmiřtir. řekil 3.7’de yukarıda anlatılan yapı gösterilmiřtir.

Audio Server Thread (AST) modülü

Genel yapısı VST modülüne ok benzeyen AST modülü, kendisine baęlı olan ACT modülünün isteklerine gerekli hizmeti vermeye yükümlüdür. Ařaęıda bir listesi verilmiř olan bu istekler “istekler.h” dosyasında tanımlanmıřtır.

- **START_AUDIO**
Ses akıřını bařlat
- **STOP_AUDIO**
Ses akıřını durdur

İstemciden ses akıřı isteęi geldięinde ses verisi **AUDIO_FRAME** veri türü ile birlikte istemciye gönderilmeye bařlanır.

Camera Server Thread (CST) modülü

CST modülü kendisine baęlı CCT modülünün isteklerine gerekli hizmeti vermekten sorumludur. Bu istekler ařaęıda verilmiřtir.

- **UP**
Kamerayı yukarı yönde hareket ettir.
- **DOWN**
Kamerayı ařaęı yönde hareket ettir.
- **LEFT**
Kamerayı sola hareket ettir.
- **RIGHT**
Kamerayı saęa hareket ettir.
- **UP_LEFT**
Kamerayı sol üst köřeeye hareket ettir.
- **UP_RIGHT**
Kamerayı saę üst köřeeye hareket ettir.
- **DOWN_LEFT**
Kamerayı sol alt köřeeye hareket ettir.
- **DOWN_RIGHT**
Kamerayı saę alt köřeeye hareket ettir.
- **WHITE_INC**
Kameranın beyazlıęını artır.
- **WHITE_DEC**
Kameranın beyazlıęını azalt.

- ZOOM_OUT
Uzaklařtır.
- ZOOM_IN
Yakınlařtır.
- FOCUS_INC
İleri yönde odak ayarı.
- FOCUS_DEC
Geri yönde odak ayarı.
- STOP_CAMERA
Kameranın o anda yaptığı işlemi durdur.
- POSITION
Kamerayı bir bölgeye yönelt. Bu istekten sonra istenen bölgenin koordinatları gönderilir.
- RESOLUTION
Videonun çözünürlüğünü deęiřtir. Bu istekten sonra istenen çözünürlük gönderilir.

4.BÖLÜM

SONUÇLAR, SORUNLAR VE ÖNERİLER

Bu çalışmada sistemin sağlam ve geliştirilebilir olmasına birinci derecede önem verilmiştir. Geliştirilen sistemin temel yapısı değiştirilmeden, sisteme eklentiler yapılması oldukça kolaydır. Bunun nedeni kullanıcı dostu olarak tasarlanan sistemin temelinde bulunan istek tabanlı mantıktır. Diğer bir deyişle sistemin geliştirmeye yatkınlığıyla programcı dostu olduğu da söylenebilir.

Çalışma esnasında karşılaşılan en büyük sorun mevcut ağ bant genişlikleridir. Geliştiriciden bağımsız oluşan bu sorun dışında karşılaşılan diğer sorunlar geliştirme aşamalarında aşılmıştır.

Elde edilen sevindirici sonuçların yanında video sıkıştırma üzerinde yapılacak geliştirmeler sistemin kullanılabilirliği açısından yararlı olacaktır; zira günümüz internet bant genişliği göz önüne alınırsa elde edilen sonuçlar oldukça iyi olsa da her zaman daha iyisini isteyen insanoğlunu tatmin etmemektedir. Görünen o ki oldukça kısa bir süre zarfında video sıkıştırma kısmında iyileştirmeler gerçekleştirilebilir.

KAYNAKLAR

- [1] Video Codecs and Pixel Formats. <http://www.fourcc.org>.
- [2] Hawk Software. <http://www.hawksoft.com>.
- [3] Msdn Library. <http://msdn.microsoft.com>.
- [4] Peter Wayner. Compression Algorithms for Real Programmers, JPEG, sayfa 125-139, 2000.
- [5] Iain E. G. Richardson. H.264 and MPEG-4 Video Compression, Video Formats and Quality, sayfa 9-19, 2003.
- [6] John G. Apostolopoulos, Wai-tian Tan, Susie J. Wee. Video Streaming: Concepts, Algorithms, and Systems; Mobile and Media Systems Laboratory, HP Laboratories Palo Alto; 18 Eylül 2002.
- [7] Sumedh Mungee, Nagarajan Surendran, Douglas C. Schmidt. The Design and Performance of a CORBA Audio/Video Streaming Service, Ocak 1999.
- [8] Jarrod Hollingworth, Don Butterfield, Bob Swart, Jamie Allsop. C++ Builder 5 Developer's Guide, 2001.