ÖZET

Yüksek Lisans Tezi

İNSAN YÜZÜ RESİMLERİNİN KODLANMASI VE ARŞİVLENMESİ

Şükrü GÖRGÜLÜ

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Doç.Dr. Ömer Nezih Gerek
2006, 66 Sayfa

Bu çalışmada insan yüzü resimlerinde, göz, kaş, burun, dudak gibi yüz bölgelerinin çeşitli örüntü tanıma yöntemleri kullanılarak otomatik olarak tanımlanıp bölütlendirilmesi, her bölgenin ayrı ayrı değerlendirildiği bir kodlama yöntemi kullanılarak arşivlenmesi ve sisteme girilen bir yüz resminin arşivde bulunması (tanıma) durumunda arşivden çağırma ile yüz resminin geri çatılması işlemlerini gerçekleştirmek üzere Microsoft Windows ortamında geliştirilmiş bir yazılım ve uygulanan yöntemler tanıtılmıştır.

Anahtar Kelimeler: Yüz tanıma, yüz bölütleme, yüz sıkıştırma, yüz arşivleme, vektör nicemleme

ABSTRACT

Master of Science Thesis

**CODING AND ARCHIVAL
OF HUMAN FACE IMAGES**

Şükrü GÖRGÜLÜ

**Anadolu University
Graduate School of Sciences
Electrical and Electronics Engineering Program**

**Supervisor: Assoc. Prof. Dr. Ömer Nezih Gerek
2006, 66 Pages**

In this study, a computer application that is developed to handle archiving of human face images is introduced. The application includes operations such as automatically segmenting any face image into facial features (eyes, eyebrows, nose, lips etc.) using various pattern recognition methods; archiving those segmented features/parts by a method in which each feature is encoded separately; and retrieving a given image if it is recognized from the archive, by reconstruction of the image from the archival data.

Keywords : Face detection, face segmentation, archiving, retrieval, vector quantization

TEŞEKKÜR

Bu çok yönlü çalışmada çözüm yollarını göremediğim zamanlarda verdiği büyük destek ile sonuca ulaşabilmemde büyük katkısı olan ve bana değerli zamanını ayıran danışmanım Doç. Dr. Ömer Nezih Gerek'e teşekkür ederim. Kendisinden teorinin yanında motivasyonu, 'tez öğrencisine nasıl davranılır?' ı ve çok daha fazlasını öğrendim.

Aileme de koşulsuz sağladıkları her türlü maddi ve manevi destekleri için minnettarım.

Çalışmada kullanılan sayısal fotoğraf arşivini oluşturmak için fotoğraf çekimine gönüllü katılarak destek veren akademik personelimize, idari personelimize ve öğrencilerimize de teşekkür ederim.

Çalışmada bana yardımcı olmuş bölüm-içi, bölüm-dışı bütün arkadaşlarıma ve hocalarıma da teşekkür ederim.

Şükrü GÖRGÜLÜ

Ağustos 2006

İÇİNDEKİLER

Sayfa

ÖZET ..... **i**

ABSTRACT ..... **ii**

TEŞEKKÜR ..... **iii**

İÇİNDEKİLER ..... **iv**

ŞEKİLLER DİZİNİ ..... **vi**

TABLOLAR DİZİNİ ..... **vii**

1. GİRİŞ **1**

1.1 Genel Bakış 1

1.2 Amaç Ve Kapsam 2

1.3 Yüz Görüntü Arşivlerine Genel Bakış 2

1.4 Görüntüdeki Belirgin Bilginin Tutulması 4

1.4.1 Örüntü Tanıma Ve Sınıflandırma 5

2. VEKTÖR NİCEMLEME (VECTOR QUANTIZATION) **8**

2.1 Nicemlemeye Genel Bakış 8

2.2 Çok Boyutlu Nicemleme 8

2.3 Bir Vektör Nicemleyicinin Yapısı 10

2.4 En Az Bozulmalı VN 15

2.4.1 Kodlayıcı Eniyiliği 15

2.4.2 Kod-açıcı Eniyiliği 16

2.5 GirdiVektörlerinin Kullanımıyla VN Kod-kitabının Tasarlanması 17

2.6 VN Uygulamaları ve Örnekler 22

2.6.1 Sıkıştırma 22

2.6.2 Dönüşüm Nicemlemesi ve Tah. Kodlama Katsayıları 22

2.6.3 Sinyallerin Doğrudan Vektör Nicemlenmesi 23

2.6.4 Sınıflandırma ve Kümelendirme 23

2.6.5 Renk Azaltımı 24

3. UYGULAMA 25

3.1 Veri Tabanı İçin Çekilmiş Yüz Resimleri 25

3.2 Uygulamanın Aşamaları 25

3.3 Görüntü Veri Biçimi 26

3.4 Görüntüde Yüz Bölgesinin Tespiti İçin Bir Yöntem 26

3.5 Gözlerin İşaretlenmesi 28

3.6 Gözler İçin Şablon Kullanımı 29

3.6.1 İşaretlenen Alanın Taranması 30

3.7 Çevirme Ve Normalizasyon 31

3.8 Maske Uygulanması 31

3.9 Arşivleme (VN Kullanımı) 33

3.10 Arşivden Geri Çatım İle Yüzün Oluşturulması 34

4. YAZILIM VE SONUÇLAR 36

4.1 Yazılım Görünümü Ve Aşamalar 36

4.2 Maskeleme Aşaması 39

4.3 Toplu Arşiv Aşaması 40

4.4 İleri Çalışmalar 41

KAYNAKLAR 42

EKLER

EK-1: GELİŞTİRİLENYAZILIM 45

1.1 CImage C++ Sınıfı Ve Üye Fonksiyonların Tanımları 45

1.2 CPIXEL C++ Sınıfı Ve Üye Fonksiyonların Tanımları 54

1.3 MFC Arayüz Sınıfları Ve Üye Fonksiyonların Tanımları 57

1.4 CVector C++ Sınıfı ve Üye Fonksiyonların Tanımları 64

1.5 CVQ C++ Sınıfı ve Üye Fonksiyonların Tanımları 66

ŞEKİLLER DİZİNİ

Sayfa

2.1 Kodlayıcı girdi vektörüne ‘8’ endeksini üretir 10
2.2 Girdi vektörünü kod-kitaptaki kod-vektörlerle karşılaştırma 11
2.3 Kod-açıcıda giriş endeksine göre kod-vektörlerin üretimi 11
2.4 Tipik VN bölgeleri 13
3.1 YCbCr ↔ RGB dönüşüm matrisleri 27
3.2 Ten rengi ile yüz bölgesinin tahmini 28
3.3 Elde edilen göz resimlerinden ortalama şablon elde etme 30
3.4 Solda elle işaretlenmiş gözler, sağda ise tespit ed.noktalar görülmektedir 30
3.5 İki göz merk. arasına çiz. çizgi ve resmin yataya hizalanmış hali 31
3.6 Sadece gözlerin tespiti yüzün diğer ögelerini saptamaya yeterlidir 32
3.7 Kesilen öge dizinleri 32
3.8 Parçaların kesilmesiyle kalan kısmın doldurulması 32
3.9 Vektör Nicemleme işleminin kodlama ve kod çözme aşaması 33
3.10 VN sırasında mesafeleri ölçülen ve ortalamaları alınan piksel örnekleri . 34
3.11 Yeni bir yüz resm. organ kısımlarının mevcut veri tabanında sorgul. 34
4.1 Ana pencere 36
4.2 ‘Ekle’ penceresi 36
4.3 ‘Ekle’ penceresinde yüklenmiş resim 37
4.4 Sağ gözün fare yardımıyla işaretlenmesi 38
4.5 Sol gözün fare yardımıyla işaretlenmesi 38
4.6 İşlem alanının yüze doğru daraltılması 39
4.7 Yüz parçalarını ayırmak için kullanılan maskeler 39
4.8 ‘_MG_0831.jpg’ dosyasındaki yüz gör. parçalarına ayrılmış hali 40
4.9 Kod-kitap tasarım penceresi 40

TABLOLAR DİZİNİ

3.1 Olası ten rengi için seçilen Cb, Cr kanal aralığı 27

1. GİRİŞ

1.1 Genel Bakış

Bir çok farklı biçimiyle faydalı ve değerli olan bilgi, özellikle de işlenmiş bilgi, günümüzde büyük bir hızla artış göstermektedir. Dolayısıyla bilginin verimli bir şekilde saklanabilmesi, iletilebilmesi ve bilgiye hızlı bir şekilde erişilebilmesi ayrı bir önem kazanmaktadır. Bu durum özellikle sayısal görüntülerde geçerlidir. Tek bir sayısal görüntüyü temsil etmek için bile büyük miktarda sayısal alana (hafıza) ihtiyaç vardır; görüntüleme teknolojisindeki ve sayısal elektronikteki hızlı ilerlemeler sayesinde geliştirilen yeni nesil ürünler, ihtiyaç duyulan hafıza miktarını (bit[1] sayısını) daha da artırmıştır. Sayısal görüntülerden verimli bir şekilde yararlanabilmek için de görüntünün ifadesinde gerekli olan bit sayısını azaltacak yeni ve özel tekniklere ihtiyaç vardır [1]. Sayısal görüntü işlemenin bu problemle ilgilenen dalına görüntü sıkıştırma ya da kodlama denmektedir. Sayısal görüntülerin hem miktar, hem de kullanım olarak artması sonucunda, 1950'li yıllardan beri üzerinde çalışılan görüntü sıkıştırma ve kodlama konusu günümüzde daha fazla önem kazanmıştır. Çevremizdeki örüntüleri tanıyabilen bilgisayarların tasarlanması ve geliştirilmesi, araştırmaların doğal olarak yöneldiği bir konudur. İnsan yüzü tanımadan parmak izi tespitine, optik karakter tanımadan DNA[2] dizisi tespiti ve daha fazlası için bilgisayarla hassas örüntü tanıma oldukça faydalıdır.

Görüntü sıkıştırma ve arşivleme, video-konferans, uzaktan algılama, doküman ve tıp görüntüleri ve faks iletimi gibi; ikili, gri ölçekli veya renkli görüntülerin verimli işlenmesi, saklanması ve iletilmesi uygulamalarında çok önemli bir rol oynar. Görüntü sıkıştırma yöntemleri kayıplı ve kayıpsız olmak üzere iki kategoriye ayrılabilir. Kayıpsız yöntemlerde özgün resim kesin olarak geri elde edilebilirken, kayıplı yöntemlerde ise özgün resime yakın bir görüntü elde edilebilir.

[1] Bit: binary digit: ikilik tabanda rakam (1 ya da 0). Sekiz bit, bir bayt'ı oluşturur.
[2] DNA: deoxyribonucleic acid: genetik bilgi taşıyan organik hücre birimi

Genel sayısal görüntüler için çeşitli algoritmalar geliştirilmiştir. Bütün yöntemlerin ortak hedefi, görüntüyü oluşturan bileşenlerin birbiriyle en az ilişkili olduğu gösterim şeklini bulmaktır, başka bir deyişle gereksiz bilginin göz ardı edilmesiyle sadece faydalı bilginin tutulacağı bir gösterim yolu bulmaktır.

1.2 Amaç ve Kapsam

Bu tez çalışmasının amacı yukarıda kısaca değinilen yeni ve özel görüntü kodlama ve arşivleme tekniklerinden – özellikle de "vektör nicemleme" tekniklerinden – ve çeşitli örüntü tanıma ve sınıflandırma tekniklerinden yararlanılarak bir yüz tanıma ve arşivleme yazılımının geliştirilmesidir. Yazılım çalışma platformu olarak Microsoft Windows 98, Me, 2000 ve XP sistemlerinde çalışacak şekilde Microsoft Visual Studio C++ 6.0 yazılım geliştirme aracı kullanılarak geliştirildi. Arayüz tasarımında ise özellikle MFC (Microsoft Foundation Classes) kütüphanesi kullanılarak performans artırımı sağlanmaya çalışıldı.

Çalışmanın genel uygulama kapsamı yüz görüntülerinin sık kullanılabildiği her alan olarak düşünülebilir. İlerleyen kısımlarda bazı örneklere değinilmiştir.

1.3 Yüz Görüntü Arşivlerine Genel Bakış

Yüz görüntülerinden (resim-fotoğraf-imge) kişilerin otomatik tespiti ve bu görüntülerin depolanması-saklanması, sosyal içerikli pek çok uygulamada ihtiyaç duyulan bir unsurdur. Depolama ve saklama işlemlerinin bilgisayar ortamında yapılması birçok kolaylık sağlamaktadır. Sayısal ortamlarda saklanan fotoğraflar üzerinde bilgisayarla arama ve tarama işlemleri de yapılabilmektedir. Uygulamalara verilebilecek örneklerden bazıları şöyledir:

- Emniyet/güvenlik birimlerinin elinde bulunan resim/fotoğraf kayıtları. Özellikle suçlu tespitinde büyük yararlar sağlamaktadır. (Hatta bu olgu her türlü sinema filminde de kullanılmaktadır)
- Büyük bir kurum-kuruluşun personel kayıtları

- (bir kamu kuruluşundaki çalışan kayıtları veya bir üniversitedeki öğrenci kayıtları)
- Sayısal kimlik arşivleri

Bunlara benzer birçok uygulama sıralanabilir. Sadece kişi fotoğraflarının tutulduğu bir arşiv bile bu konuda örnek olabilir. İleride yapılabilecek uygulamalara örnekler ise bilim-kurgu filmlerinde zaten tema olarak kullanılmaktadır. Bazılarını sıralayabiliriz.

- Binalara / bankalara / özel alanlara giriş-çıkışlarda kamera kontrollü geçiş izni. Bu uygulamada özel bölgeye girecek olan kişi bir kameraya yüzünü gösterir. Bilgisayar sistemi kişiyi otomatik tanıyıp yetkilerine göre geçişe izin verir ya da vermez.
- Bankamatiklerden para çekiminde Atm'de[3] işlem yapan kişinin kimlik kontrolü de yine kamera tarafından alınan bir görüntünün bilgisayar analizi ile teyit edilebilir.

Yukarıda sayılan örnekler ve sayılmamış örnekler de düşünüldüğünde söz konusu uygulamaların her biri için sayısal fotoğraf arşivleri gereklidir.

Yukarıda değinilen türde bir arşivin büyüklüğü bir örnekle anlatılabilir. Yetmiş milyon kişinin nüfus kayıtlarının tutulduğu bir sistem söz konusu olsun. Her şahıs için gerekecek fotoğraf kaydı, bilgisayar analizi yapılabilecek şekilde ayrıntılı kayıtlar için, gayet iyi sıkıştırma sağlayan jpeg/jpg[4] biçimi için bile yaklaşık yüz kilobayt[5] civarında olacaktır. Fotoğrafla birlikte saklanacak başka bilgilerin de olduğu düşünülecek olursa (örneğin parmak izi kayıtları) kişi başına tutulması gereken bilgilerin toplamı beşyüz kilobayt ve bir megabayt üstü bile olabilir. Bu durumda sadece fotoğrafların saklanması kısmıyla ilgilenilsin. Yetmiş milyon kere yüz kilobayt, yedi milyon megabayt ya da yedi terabayt veri büyüklüğü demektir. Bu çok ciddi veri saklama alanının sağlanması bir takım ciddi maliyetleri de beraberinde getirir. Yani yüz gigabayt kapasiteli

[3] ATM: Automatic Teller Machine, bankamatik
[4] JPEG: Joint Photographic Experts Group. Sayısal görüntülerin sıkıştırıldığı, standart haline gelmiş algoritma.
[5] KB: kilobayt = 2^{10} bayt, MB: megabayt = 2^{20} bayt, GB: gigabayt = 2^{30} bayt, TB: terabayt = 2^{40} bayt

sabitdisklerden[6] yetmiş adet veya DVD'lerden yaklaşık bin beşyüz adet ya da yediyüz megabayt kapasiteli CD'lerden yaklaşık on bin adet gereklidir. Bu veritabanı içinde yapılacak herhangi bir bilgisayarla arama-tarama işlemi için bütün veri depolama ünitelerinin çevrim-içi çalışıyor olması gerekir. Ancak DVD ve CD seçenekleri çevrim-içi işlemlerde kullanılamaz. Bu devasa veritabanı sistemi büyük bilgisayar sistemleri ya da bilgisayarların birbirlerine bağlı olduğu ve ortak çalıştıkları bir ağ sistemi demektir. Büyük sistemlerin gereksinimleri ve maliyetleri de büyük olacaktır. Bir diğer düşünülmesi gereken konu ise bu veritabanı üzerinde hızlı arama-tarama yapabilecek bir bilgisayar-yazılım sisteminin olması gerekliliğidir.

1.4 Görüntüdeki Belirgin Bilginin Tutulması

Verideki bilginin kalitesinden büyük ödünler vermeden veri saklama boyutlarının azaltıldığı ve hızlı arama-tarama işlemlerinin yapılabildiği çeşitli arşivleme yöntemlerinin geliştirilmesi yukarıdaki basit senaryodan da anlaşılacağı gibi çok önemlidir. Doğrudan bit-bayt bazında sıkıştırmadan ve arşivlemeden ziyade veri üzerindeki bilgiyi tutan sıkıştırma ve arşivleme teknikleri uygulanırsa sıkıştırma oranları artırılmış olur. Dolayısıyla verideki bilgiyi tutmak için gereken alan da azaltılmış olur. Önemli nokta, veri bilgi kalitesinden büyük ödünler verilmeden veri saklama boyutunu azaltmaktır. Veri bilgi kalitesini tanımlamak gerekirse; kayıplı da olsa sıkıştırılmış verinin ya da görüntünün özgün görüntüde var olan temel bilgileri bulunduruyor olması istenir. Yani, belirleyici ve ayırt edici özellikleri taşıması beklenir. Bu özellikler ne kadar belirli ise veri bilgi kalitesinin o oranda yüksek olduğunu söyleyebiliriz. Örneğin bir yüz fotoğrafında belirgin ögeler; gözler, kaşlar, burun ve dudaklardır. Bu yüz organlarını ön plana çıkaran ve bu ayrıntıları koruyan bir veri sıkıştırması veri bilgi kalitesinden de fazla ödün verilmeden yüksek oranda sıkıştırma sağlayabilir. Böylece yüzün tanınması ve tanımlanması için gerekli olan bilgi korunmuş olur.

Yüz tanıma konusu üzerinde yapılan araştırmalarla insan beyninin 'özellik çıkartımı' (feature extraction) denilen bir yaklaşımla insan yüzlerini tanımladığı

[6] HDD: Hard Disc Drive, DVD: Digital Video Disc (Standart 4,7 GB), CD: Compact Disc (Standart 700MB)

öğrenilmiştir. Yani insan beyninin algılama yöntemi, insan yüzünü, kaş, göz, burun, dudaklar gibi belirleyici organlara ayırıp vektörel biçimde tanımlama şeklindedir. Yüzdeki bu organların her birinin kendilerini diğer organlardan ayıran 'örüntü'leri (pattern) yani geometrik ve renksel olarak belirleyici görünüm tipleri-yapıları vardır. Örneğin gözler basitçe uçlarda sivrilerek daralan beyaz dolgulu bir elips içinde koyu yuvarlak iris ile, ağız ise yine uçlarında sivrilen pembe-kırmızı renkte ve kapalı haldeyken birbirlerine yaslanan iki ince uzun eliptik dudak yapısıyla belirginleşir.

Gözlerin ve dudakların belli örüntülere sahip olmalarıyla beraber farklı çeşitlerde olabilecekleri de göz önüne alınırsa bunların farklı kişilere ait olabileceği kararının verilebilmesi için sınıflandırılması da önemlidir. Örneğin patlak veya çekik gözler ya da dolgun veya ince dudaklar gibi... Ayrıca bu organların yüzdeki konumları da birbirlerine oranlı ve belirlidir.

Yukarıda anlatılanlardan temel araştırma başlıkları çıkartılacak olursa 'örüntü tanıma', 'sınıflandırma', 'sıkıştırma/kodlama' gibi konulardan bahsedilebilir. Bu temel başlıklar altında tanımları ve çeşitleri bu çalışmada verilecektir.

1.4.1 Örüntü Tanıma ve Sınıflandırma

Türk Dil Kurumunun Türkçe sözlüğünde örüntü(İngilizce karşılığı 'pattern') kelimesi 'Olay veya nesnelerin düzenli bir biçimde birbirini takip ederek gelişmesi' olarak tanımlanmaktadır. Bu tanımın bilgisayar dilindeki hatta sayısal işaret işleme alanındaki kullanımı da buna benzer bir şekilde bit'lerin bayt'ların ya da piksellerin düzenli bir biçimde birbirini takip ederek belirli bir yapı oluşturması şeklindedır. Bu 'düzenlilik' çeşitlilik göstererek farklı örüntüleri oluşturur. (Nitekim bilişim sözlüğündeki [1] 'pattern' kelimesinin tanımı da şöyledir; 1- Tanınan bir şekil, düzen, desen. 2- Bilgisayar grafiğinde bir imgeyi oluşturan pikseller. 3- Arama anahtarı olabilecek düzenli bir dizgi.) Çeşitli sayısal işaret işleme teknikleri ile bu örüntüler tespit edilerek işe yarar biçimde sınıflandırılmaya çalışılır.

Bilgisayarla örüntü tanıma teknikleri, çok geniş bir yelpazede bir çok yöntemin genelleştirilmiş adıdır. Görüntü analizi ile ilgili araştırmacıların uzun bir süredir bu konularla ilgili çalışmaları olmuştur. Literatürde tipik olarak üzerinde durulan husus, genel bir sayısal görüntü üzerinde önce insan yüzünün, ardından da yüze ait bölgelerin tespitidir.

Yüzün bölgelere bölünmesinin ardından bu bölgelerin yüzden kesilmesi ve kesilen bölgelerin daha önceden hazırlanmış bir veritabanındaki bölgelerle karşılaştırılarak arşivlenmesi yöntem olarak kullanılacaktır. Örnek verilerek konu daha kolay anlatılabilir. Bir yüz ayrıntısı olarak sağ gözü ele alınsın. Yüz görüntüsündeki (ya da imgesindeki) sağ gözün konumu çeşitli yöntemlerle tespit edilmiş olsun. Sağ göz imgesi kesilerek yüzden ayrılsın ve sağ göz veritabanındaki sağ göz imgeleriyle karşılaştırılsın. Karşılaştırma işleminde kullanılacak belirli bir yöntemin olması gereklidir. Benzerliği belirlemek için yapılan her karşılaştırmada bir sonuç çıkacaktır. Bu sonuçlara göre eldeki sağ gözün veritabanındaki hangi göze daha benzer olduğu bulunur. Sayısal görüntülerin benzerliği konusu da örüntü tanımanın ayrı bir çalışma sahasıdır. Bu sayede sınıflandırma yapılabilmektedir. Bazı durumlarda veri çeşitliliğinden dolayı sınıflandırma işlemi kesin sınırlara göre yapılamamaktadır. Kesin sınırların sınıflandırma verimini düşürdüğü durumlarda kümeleme-gruplama (clustering) yapılması sınıflandırma verimini artırır. Gruplanmış verilerin sınıflandırılması daha kolay olacaktır. Eğer küme sayısı verileri en uygun sınıflandıracak şekilde seçilirse her grup için bir temsilci merkez belirlenerek grup içindeki bütün veriler için bu merkez temsilci/örnek olarak kullanılabilir. Böylece bir tür örnekleme ve seviyelendirme yapılmış olur.

Örnekleme ve seviyelendirme basitçe sürekli(continuous) eksende kesitler alarak veri alma işlemi olarak tanımlanabilir. Yani sürekli(continuous) veri kesikli (discrete-digital) hale getirilmiş olur. Ya da sürekli veri belli seviyelere/aralıklara ayrılır ve her veri aralığı bir tek örnekle temsil edilir. Örnekleme ve seviyelendirme işlemleri zaten örneklenmiş ve seviyelendirilmiş olan veriler üzerinde yeniden de uygulanabilir. Bu durumda aşağı-örnekleme (down-sampling) işlemi yapılmış olur. Yukarıda anlatılan seviyelendirme işlemi, kuvantalama ya da nicemleme olarak da bilinir. Çok boyutlu vektörlerin

kümelenerek-gruplanarak seviyelendirilmesi genel olarak ‘vektör kuvantalama (Vector Quantization [2])’ ya da ‘vektör nicemleme’ ismiyle tanımlanır.

2. VEKTÖR NİCEMLEME (VECTOR QUANTIZATION)

2.1 Nicemleme

Nicemleme, sinyallerin (işaretlerin) sayısal olarak işlenmesi için kaçınılmaz bir adımdır. Bilgisayarlarda ve sayısal işaret işlemcilerde[7] işaret örneklerinin sonsuz hassasiyette/kesinlikte gösterilmesi mümkün değildir. Dönüştürücüler her veri örneği için sonlu hassasiyette sayısal bir değer atar. Bu nedenle sayısal değerler bilgisayarın/işlemcinin sayısal hassasiyetine göre yeniden seviyelendirilmelidir. İşaretlerin kaçınılmaz nicemlenmesinin yanında, tipik bir sayısal işaret, nicemleme ve entropi kodlaması[8] ile takip edilen bir dönüşüm /tahmin aşamasıyla sıkıştırılmış bir halde saklanır. Eğer seçeneğe bağlı dönüşüm/tahmin aşamasından geçmiş örnekler ayrı ayrı nicemlenirse buna skaler nicemleme denir. Ama örnekler vektörler oluşturacak biçimde gruplandırılırsa bunların nicemlenmesi işlemi de vektör vektör nicemleme olarak tanımlanır.

2.2 Çok Boyutlu Nicemleme

Nicemlemeyi bir boyuttan (skaler için) çok boyuta dönüştürmenin pek çok önemli etkisi vardır. Bu önemli etki daha çok iki boyutlu sayısal görüntülerin arşivlenirken sıkıştırılmasında (kodlanmasında) ve nicemlenmesinde gözlenebilir. En başta; Vektör nicemleme tam olarak veri değerlerinin basit seviyelere yuvarlanmasına karşılık gelmez. Vektör nicemleme aşaması gruplaşan örneklerin oluşturduğu vektörü temsil eden bir sayı/endeks üretir.

Çıktı endeksi, ki bu bir tamsayıdır, temsil ettiği ve gerçek veya karmaşık değerlere sahip örneklerin oluşturduğu vektörle çok az bağıntılıdır ya da aralarında hiç bir bağ yoktur.

Vektör nicemlemedeki nicemeleme terimi aslında benzer vektörlerin aynı endeks ile temsil edilmesinden gelmektedir. Böylece çok-boyutlu uzaydaki bir

[7] Sayısal İşaret İşlemci: DSP: Digital Signal Processor
[8] Entropi kodlaması, sinyal enerjisinin dağılımının en aza indirgenerek düzenlendiği bir kodlama veya sıkıştırma türüdür

çok farklı vektör bir endeks ile gösterilen tek bir vektöre nicemlenmiş olur. Her endeks önceden belirlenmiş bir vektöre karşılık gelir. Bu açıdan bakılırsa, farklı endekslerin sayısı nicemleme seviyelerini sayısını belirler. Buna göre bir veri vektörünün nicemleme endeksinin önceden belirlenmiş vektörler kümesindeki (kod-kitabı) en yakın vektöre göre seçilmesinin gerektiği söylenebilir. Örneğin, ilgili $\boldsymbol{x}$ vektörü $\boldsymbol{v_i}$ olarak tanımladığımız bir kod-kitabı elemanına en yakın olsun. Bu durumda VN çıktısı basit olarak *i*'dir. Geri çatma aşamasında yani kodlanmış bir verinin tekrar oluşturulması aşamasında, *i* endeksi $\boldsymbol{v_i}$ vektörü olarak geri çatılır ve $\boldsymbol{x}$'in $\boldsymbol{v_i}$'ye nicemlendiği söylenebilir.

Bir dizi vektöre endeksler atanmasının kodlama dışında başka sonuçları ve etkileri de vardır. $\boldsymbol{v_i}$'ye yakın olan vektörler *i* olarak ve $\boldsymbol{v_j}$'ye yakın olanlar da *j* olarak endekslendiği için, bu durum kod-kitabı vektörleri üzerindeki kümelenme bilgisini de otomatik olarak sağlar. Vektörlerin kümelenmesi genellikle sınıflandırma problemlerinin çözümünde kullanılır. Sınıflandırma ise örüntü tanımanın en önemli ögelerinden biridir. Sonuç olarak, işaret kodlamak için geliştirilmiş pek çok VN algoritmasının örüntü sınıflandırma ve tanımada kullanılan karşılıkları vardır. ISO-DATA [3], En Yakın Komşuluk (k-NN) [4] ve Kendinden-organizeli özellik haritalama (SOM) [5] gibi yaygın yöntemlerin, VN kod-kitaplarını iyileştirmek için kullanılan Max-Lloyd ve Linde-Buzo-Gray (LBG) [6] gibi algoritmalara çok benzeyen algoritmaları vardır. Bir başka yaygın VN uygulaması ise sayısal resimler için renk azaltımıdır. Pek çok görüntü alma cihazı her nokta/piksel için genellikle kırmızı yeşil ve mavi bileşenlerin (RGB)[9] sekiz bit ile gösterildiği renkli resimler üretir. Bu da her nokta/piksel için yirmidört bit demektir. Görüntüleme tampon sınırlamaları veya kayıt ortamı gereksinimlerine bağlı olarak her piksel için atanan bit sayısının azaltılması istenen bir durumdur. Bu da RGB bileşenlerinin daha az endeks sayısına nicemlenmesi ile yapılabilir. ("gif" resim biçemi[10])

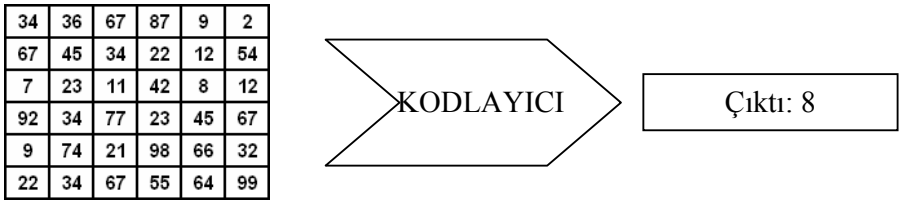
Vektör nicemlemenin anlatıldığı bölüm aşağıdaki gibi düzenlenmiştir. İlk olarak bir vektör nicemleyicinin yapısı anlatılacaktır. Burada bozulma konusu ve

[9] RGB: Red Green Blue (Kırmızı Yeşil Mavi). Her türlü sayısal görüntü piksellerinin ekranda oluşturulması için kullanılan standart veri biçimi.
[10] GIF: Graphics Interchange Format. Sayısal görüntülerin özellikle renk bazında sıkıştırılmasında kullanılan standart biçem.

çeşitli ölçütlere yer verilecektir. İkinci olarak en az bozulma ile VN'nin özellikleri ve gerekli eşitlikler verilecektir. Sonra belli bir veri kümesi için oluşturulan VN kod-kitabını iyileştiren temel algoritma verilecek ve çeşitli VN kod-kitabı tasarımı teknikleri anlatılacaktır. Son olarak da, bazı tipik VN uygulamalarına yer verilecektir.

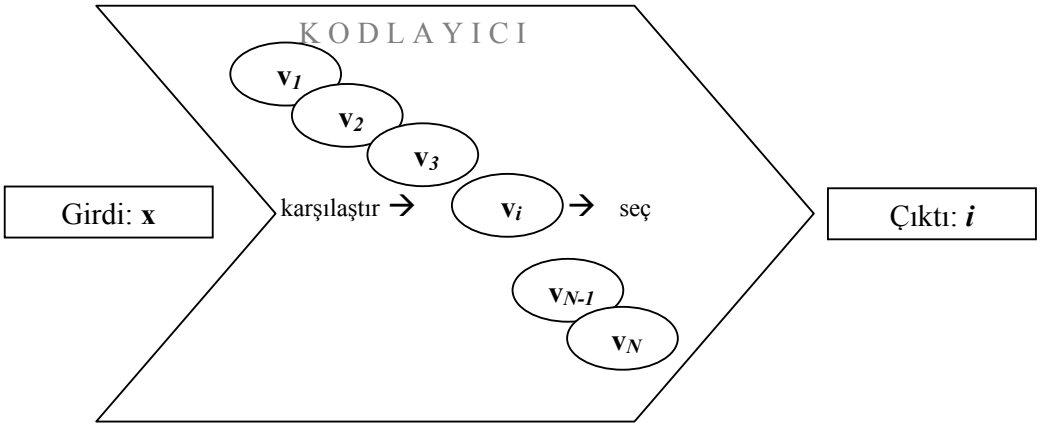
2.3 Bir Vektör Nicemleyicinin Yapısı

Bir vektör nicemleyici iki parçadan oluşur. Bunlar kodlayıcı ve kod-açıcıdır. Kodlayıcı girdi vektörüne bir endeks atayan parçadır. Örneğin **Şekil 2.1**'deki girdi vektörü otuzaltı ögeden oluşmaktadır ve kodlayıcı endeks numarasını üretir. Çıktı '8' olsun.



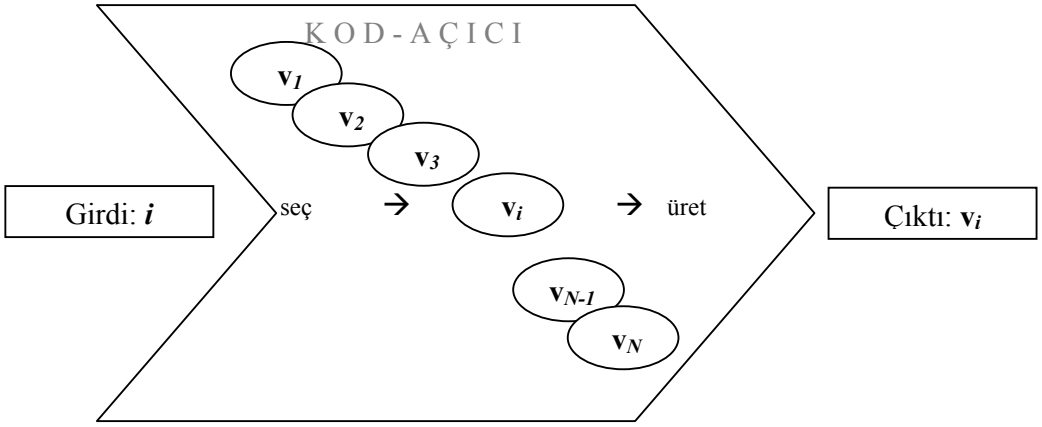
Şekil 2.1 Kodlayıcı girdi vektörüne '8' endeksini üretir.

Bu birimde kod-kitabı (***C***) olarak tanımlanan bir küme oluşturan kod vektörleri veya kod-kelimeleri olarak tanımlı birbirinden farklı bir dizi $\mathbf{v}_i$ vektörü vardır. Kodlayıcı birimi kod-kitabında girdi vektörüne en yakın olan eşleştirmeyi arar. Eğer kod-kitaptan, $\boldsymbol{x}$ girdi vektörüne en yakın kod-vektörü bazı ölçütlere göre $\mathbf{c}_i$ ise kodlayıcı çıktısı i'dir. İşlem **Şekil 2.2**'de özetlenmiştir. Kodlama sonucunda; çıktı olarak yalnızca tamsayılar elde edilir ki bu da büyük miktarda gösterim kazancı getirir (sıkıştırma).



Şekil 2.2 Girdi vektörünü kod-kitaptaki kod-vektörlerle karşılaştırma

Kodlayıcı çıktısı sadece kod-vektörlerinin bir gösterimidir. Dolayısıyla bazı kayıplara rağmen kodlanmış işareti geri elde etmek için endeks sayıları ile oluşturulmuş bu gösterim, girdi olarak tamsayıları kabul eden bir koda-açıcıya girilmelidir. Kod-açıcı çıktı olarak endeksin gösterdiği kod-vektörü üretir. Kod-açıcı aynı zamanda ters-nicemleyici olarak da adlandırılır. (**Şekil 2.3**)



Şekil 2.3 Kod-açıcıda giriş endeksine göre kod-vektörlerinin üretimi

Normalde vektör nicemleyici terimi kodlayıcı ve kod-açıcı birimlerin bileşimi olarak kullanılır. Matematiksel gösterim kullanılırsa;

$$i = \mathcal{E}(\boldsymbol{x}) \tag{2.1}$$

$$\boldsymbol{v}_i = D(i) \tag{2.2}$$

$$\boldsymbol{v}_i = Q(\boldsymbol{x}) = D(\mathcal{E}(\boldsymbol{x})) \tag{2.3}$$

Vektör nicemleyicinin iki özelliği vardır:

- k: boyut
- N: kod-kitabı uzunluğu

Tamsayı olan k değeri her vektörün eleman sayısına karşılık gelir. Böylece eğer vektör elemanları gerçek sayılarsa, bir girdi vektörü veya kod-vektörü k-boyutlu $\mathbf{R^k}$ öklit uzayında bir nokta olur. N tamsayısı ise $\boldsymbol{C}$ kod-kitabındaki vektörlerin sayısıdır. Matematiksel olarak N'ye $\boldsymbol{C}$'nin büyüklüğü/uzunluğu denir. Bu açıdan bakılırsa; Q, öklit uzayından sınırlı bir kümeye tanımlanmış bir işlemdir.

$$Q : \mathbf{R^k} \rightarrow \boldsymbol{C}$$

Kodlama amacıyla kullanılan kod-kitabı genellikle hem gönderen(kodlayıcı) hem de alıcı(kod-açıcı) kısımlarca bilinir. Böylece, sadece kodlayıcının tamsayı çıktısı (endeks) iletilir.

Diğer taraftan, kod-kitabının kendisi k-boyutlu $\mathbf{R^k}$ öklit uzayının N bölgeye işe-yarar biçimde bölümlendirilmiş halidir. Eğer kodlayıcının bir x girdi vektörüne karşılık ürettiği endeks i ise, her $\mathbf{R}_i$ bölgesi, $\boldsymbol{x}$'in $\mathbf{R}_i$ bölgesinde olduğunu gösterecek şekilde doğrudan, nicemleyici tarafından tanımlanır. Bu bölgeler *Varanoi* hücreleri olarak da bilinir. Matematiksel olarak;

$$\mathbf{R}_i = \{ \text{her } \boldsymbol{x} \in \mathbf{R}_k \mid Q(\boldsymbol{x}) = \boldsymbol{v}_i \} \tag{2.4}$$

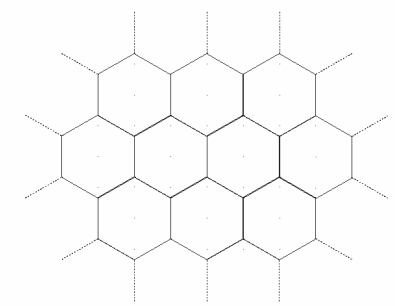
Cümlelerle ifade edersek; $\mathbf{R}_i$ kümesi $\boldsymbol{v}_i$ kod-vektörüne diğer kod-vektörlerinden daha yakın olan tüm noktaların kümesi olarak tanımlanır. Bir bölge sonlu k-boyutlu bir hacimle sınırlı (tanecikli hücre) veya sınırsız (fazla-yüklü hücre) olabilir.

Ayrıca, k-boyutlu uzayın N bölgeye ayrılması, vektör nicemleyiciye başka bir tanım yolunu da aşağıdaki gibi açmaktadır. Eğer her $\boldsymbol{x} \in \mathbf{R}_i$ ise $Q(\boldsymbol{x}) = \boldsymbol{v}_i$ olur. Dikkat edilirse, bu tanıma göre VN, gruplama veya kümeleme işlemlerinde kullanılmak için de çok uygundur. Bu amaçlarla kullanıldığında, kodlayıcı endeksleri girdinin hangi kümeye ait olduğunu hemen belirtir haldedir.

Bu bölümün başından beri kod-vektörlerden birine diğerlerinden daha yakın olma kavramını kullanıyoruz. Bu 'daha yakın olma' durumunu açıklığa kavuşturalım. Birçok uygulamada, amaç bakımından iki vektör arası öklit uzaklık,

bu vektörlerin birbirlerine ne kadar yakın olduklarının ölçümü için kullanılmaktadır. Diğer taraftan, bazı başka uzaklık ölçütlerinin de vektörlerin ne kadar yakın olduğunun tespitinde kullanılabilir. Uzaklık tanımı için tek kısıtlama, uygun bir ölçüt olması gerekliliğidir. Eğer kodlayıcı ve kod-açıcı tarafından uzaklık ölçütü olarak uygun bir metrik[11] kullanılıyorsa, $\boldsymbol{R_i}$ bölgelerinin herbirinin konveks yani dış-bükey olması bir sonuç olarak ortaya çıkar.[12]

Şekil 2.4'te görülen tipik bir 2-boyutlu VN bölge bölüntülemesidir. Merkezdeki bölgeler sınırlı hücreler ve merkezden uzak olan en kenardakiler dışa doğru uzayan (sınırlı olmayan) hücrelerdir. Bütün bölgelerin dış-bükey olması nedeniyle bu ayrımın uygun bir bölüntüleme olduğu dikkate değerdir.



Şekil 2.4 Tipik VN bölgeleri

VN'nin genel yapısıyla ilgili son bir açıklama da vektörleri oluşturmak için gruplaşan girdilere göre sıkıştırma performansını en-iyileştirebilme özelliğidir. Shannon göstermiştir ki; eğer elimizde girdi vektörlerine N tane endeksten birini en iyi kodlama performansıyla atayan bir kodlama sistemi varsa, VN de bu en iyi kodlayıcı kadar başarılı olabilmektedir [7]. bu performansa erişmenin yolu, sonraki bölümde anlatılacağı gibi, bölgelerin ve kod-vektörlerin eniyileştirilmesinden geçer. En aza indirgeme bozulma hususunda olduğu için,

[11] Metrik uzayda uygun(proper) bir metrik ($D(\cdot,\cdot)$), $\boldsymbol{a}$, $\boldsymbol{b}$ ve $\boldsymbol{c}$ vektörleri için aşağıdaki dört koşulu sağlayan uzaklık ölçütüdür.

1. Eksi olmama: $D(\boldsymbol{a},\boldsymbol{b}) \geq 0$
2. Değişebilirlik: $D(\boldsymbol{a},\boldsymbol{b}) = 0 \leftrightarrow \boldsymbol{a} = \boldsymbol{b}$
3. Simetri: $D(\boldsymbol{a},\boldsymbol{b}) = D(\boldsymbol{b},\boldsymbol{a})$
4. Üçgen eşitsizliği: $D(\boldsymbol{a},\boldsymbol{b}) + D(\boldsymbol{b},\boldsymbol{c}) \geq D(\boldsymbol{a},\boldsymbol{c})$

[12] Bir öklit bölgesinin dış-bükey olması, o bölgenin elemanı olanı herhangi iki noktayı birleştiren doğru parçasının da o bölgede olmasını gerektirir.

hepsi farklı eniyileştirme[13] sonuçları veren birtakım bozulma metrikleri formülleştirilebilir. En çok kullanılan metrikler aşağıda sıralanmıştır.

*** Minkowski metriği:**

$$d_L(x, v_i) = \left(\sum_{m=1}^{k} \left| x(m) - v_i(m) \right|^L \right)^{1/L} \tag{2.5}$$

Ve minkowski metriğinin özel durumları:

- Öklit(Euclidean) uzaklığı (MSE, $L = 2$):

$$d_E(x, v_i) = \left(\sum_{m=1}^{k} \left| x(m) - v_i(m) \right|^2 \right)^{1/2} = (x - v_i)^T (x - v_i) \tag{2.6}$$

- Manhattan uzaklığı (MAE, $L = 1$):

$$d_M(x, v_i) = \sum_{m=1}^{k} \left| x(m) - v_i(m) \right| \tag{2.7}$$

- Chebyshev uzaklığı (max, $L = \infty$):

$$d_C(x, v_i) = \max_{m=1}^{k} \left| x(m) - v_i(m) \right| \tag{2.8}$$

*** Hamming Uzaklığı:**

$$d_H(x, v_i) = \sum_{m=1}^{k})(1 - \delta(x(m) - v_i(m))) \tag{2.9}$$

*** Mahalanobis Uzaklığı:**

$$d_R(x, v_i) = (x - v_i)^T C_x^{-1} (x - v_i) \tag{2.10}$$

Burada C_x : ***x***'in özortakdeğişinti[14] matrisidir.

Yukarıda verilen metriklerin dışında, uygulamaya ve kullanışlılığa göre pek çok bozulma metriği de geliştirilebilir. Bozulma metriği $d(\cdot,\cdot)$'nin de kullanılmasıyla bütün VN sistemi **Şekil 3.9**'daki gibi yeniden ifade edilebilir.

[13] Optimization: Eniyileştirme [1]
[14] Autocovariance: Özortakdeğişinti [1]

2.4 En az bozulmalı VN

Bir vektör nicemleyicinin performansı bu bölümün başında değinilmiş olan kodlayıcı ve kod-açıcı birimlerinin en-iyiliği ile belirlenir. <performans> ve <en-iyilik> terimleri, verilen çıktı seviyesi sayısında nicemleyicinin ürettiği bozulma miktarıyla doğrudan bağlantılıdır. Bu bölümde en az bozulmalı VN için gerekli olan kodlayıcı ve kod-açıcı özelliklerini tanımlayacağız.

2.4.1 Kodlayıcı Eniyiliği[15]

Bölümün başında çıktı vektörünün girdi vektörüne en yakın olan kod-vektör olarak seçildği görülmüştü. Bu tarz vektör nicemleyiciler "en yakın komşu" nicemleyicileri olarak bilinirler. "En yakın komşu" kuralı kodlayıcının eniyiliği açısından gereklidir.

$Q(\boldsymbol{x}) = \boldsymbol{v_i}$ sadece eğer her j için $d(\boldsymbol{x},\boldsymbol{v_i}) \le d(\boldsymbol{x}, \boldsymbol{v_j})$ ise. (2.11)

Kodlayıcının eniyiliği için en yakın komşu kuralının gerekliliğini gösterilmesi gayet basittir:

Eğer bir vektör kendisine en yakın kod-vektörü nicemlenmezse bozulma artar. Bu yüzden, eniyi kodlayıcı en küçük $d(\boldsymbol{x},\boldsymbol{v_i})$ uzaklığı için bütün kod-kitabı aramalıdır.

$$d(x,Q(x)) = \min_{i=1}^{N} d(x,v_i) \tag{2.12}$$

Bu en az bozulma, beklenen ortalama bozulmayı istatistiksel olarak en aza indirger.

$$D = \int d(x,Q(x)) f_x(x) dx \tag{2.13}$$

Burada $f_x(x)$ fonksiyonu x'in birleşik olasılık yoğunluk fonksiyonudur[16]. Sonuç olarak, $\boldsymbol{R_i}$ bölgeleri oluşur ve bölüntüleme kuralı kodlayıcıyı belirler.

[15] Optimality: Eniyilik [1]
[16] joint probability density function, joint pdf: Birleşik olasılık yoğunluk fonksiyonu [1]

2.4.2 Kod-açıcı Eniyiliği

Eniyi kod-kitabın bulunmasıyla sağlanan ikinci eniyilik şartı, kod-açıcı birim hakkındadır. Diğer bir deyişle, kümelenen bölgeler verilmişse, her bölgeyi en iyi temsil eden kod-vektörler bulunmalıdır. İstatistiksel olarak, bir R_i bölgesini temsil eden v_i kod-vektörü, R_i bölgesinde kalan herhangi bir x girdi vektörü için beklenen en az bozulmaya neden olacak şekilde seçilmelidir.

$$v_i = \arg\min_v E\{d(x,v)|x \in R_i\} \tag{2.14}$$

Eşitlik 2.14 aynı zamanda "ağırlık merkezi (centroid)" kuralı olarak da bilinir, çünkü beklenen değer R_i bölgesinin merkezine karşılık gelmektedir. MSE bozulma metriğine (**Eşitlik 2.6**) göre ağırlık merkezi, x'in ($x \in R_i$) en az MSE (MMSE) tahminine karşılık gelir.

$$v_i = cent(R_i) = E\{x|x \in R_i\} \tag{2.15}$$

Eşitlik 2.14'ün ispatı aşağıdaki gibidir:

v_i kod-vektörlerinin kümesi olarak verilmiş olan bir kod-kitap için ortalama bozulma:

$$D = \sum_{i=1}^{N} \int d(x,v_i) f_x(x) dx = \sum_{i=1}^{N} P_i \int d(x,v_i) f_{x|i}(x) dx \tag{2.16}$$

ile bulunur. P_i; x'in R_i bölgesinde bulunma olasılığı ve $f_{x|i}(x)$ ise $x \in R_i$ için x'in koşullu olasılık yoğunluk fonksiyonudur.

Beklenen değerin tanımından yola çıkılırsa;

$$E\{d(x,v_i)|x \in R_i\} = \int d(x,v_i) f_{x|i}(x) dx \tag{2.17}$$

olur. Ağırlık merkezi, **Eşitlik 2.17**'nin sağ tarafındaki beklenen değeri en aza indirgediği için sol tarafını da en aza indirgemiş olur. Böylece **Eşitlik 2.16**'daki toplam terimi yani bozulma da en aza indirgenir.

Eşitlik 2.15, bir bölgeye karşılık gelen kod-vektörün seçiminde izlenecek yolu da sağlar. Bu eşitlikten ağırlık merkezi şöyle hesaplanabilir:

$$v_i = \int_{Ri} x f_{x|i}(x) dx = \frac{\int_{Ei} x f_x(x) dx}{\int_{Ei} f_x(x) dx} \tag{2.18}$$

Son olarak, bir eniyi vektör nicemleyici aşağıdaki özellikleri de sağlar.

(a) $E\{Q(x)\} = E\{x\}$, diklik[17] koşulu olarak bilinir.

(b) $E\{x^T Q(x)\} = E\{\|Q(x)\|^2\}$

(b) $E\{\|Q(x)\|^2\} = E\{\|x\|^2\} - E\{\|x - Q(x)\|^2\}$

2.5 Girdi Verilerinin Kullanımıyla VN Kod-kitabının Tasarlanması

Eniyi VN için, kodlayıcı ve kod-açıcı birimlerin eniyilik koşullarının aynı anda sağlanması gerekir. Verilen sayıda kod-vektör için (buna N denilsin), amaç, kod-vektörlerin dolayısıyla da karşılık gelen bölgelerin seçiminde en az bozulmaya ulaşmaktır.

Özellikler ve eşitlikler Bölüm 2.4'te tanımlanmıştır. Eldeki verilerden yola çıkılarak en-iyi veya kısmî-eniyi nicemleyiciye ulaşmak için bir kaç yöntem önerilmiştir. En çok kullanılan tekniklerden biri, bir başlangıç kod-kitabından yola çıkarak kod-kitabı adım adım iyileştiren *Genelleştirilmiş Lloyd Algoritmasıdır* [6], [7]. Bu nicemleyicinin skaler sürümü skaler nicemleyici tasarımında da kullanılmaktadır. Ayrıca bu algoritma kümeleme-sınıflama alt-literatüründe k-ortalamaları[18] veya ISODATA[3] olarak da anılmaktadır.

Lloyd adımlaması iki aşamadan oluşmaktadır:

- *En yakın komşu aşaması*: Verilen bir kod-vektörler kümesi, $\boldsymbol{C} = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \cdots, \boldsymbol{v_N}\}$ için, bölgeler şöyle tanımlanmıştır:

  $$\boldsymbol{R_i} = \{\text{her } \boldsymbol{x} \in \boldsymbol{R^k} \mid d(\boldsymbol{x}, \boldsymbol{v_i}) < d(\boldsymbol{x}, \boldsymbol{v_j})\} \text{ (her } \boldsymbol{i} \neq \boldsymbol{j} \text{ için)} \quad (2.19)$$

  Eğer $\boldsymbol{x}$ verisi $\boldsymbol{R_i}$ ve $\boldsymbol{R_j}$ bölgelerinin sınırındaysa (yani iki bölgede de aynı bozulma varsa), $\boldsymbol{i}$ ve $\boldsymbol{j}$ değerlerinin küçük olanı $\boldsymbol{x}$'e atanır.
- *Ağırlık merkezi aşaması:* Verilen N tane $\boldsymbol{R_i}$ bölgesi için temsilen $\boldsymbol{v_i}$ kod-vektörleri bölge merkezi olarak atanır. Öklit uzaklık ölçütünün kullanımıyla bir bölgenin merkezi o bölgeye ait olan veri vektörlerinin

[17] Orthogonality: diklik
[18] k-means [4] : k-ortalamaları

aritmetik ortalamasına karşılık gelir. Diğer uzaklık ölçütlerine göre ise ağırlık merkezi hesabı değişiklik gösterir[19].

Bu iki aşama, genel bozulma seviyesi daha fazla azalmayana kadar, tekrar tekrar uygulanır. Dikkat edilicek olursa her tekrar, bozulma seviyesini ya azaltmalı ya da korumalıdır. Bazı durumlarda boş bölgeler oluşabilir. Böyle bir durumda ya yeni bir kod-vektör atanır ya da kod-kitap büyüklüğü, *N*, azaltılır. Lloyd adımlaması eniyileme için çok genel bir yöntemdir. Buna karşın, bazıları ara aşamalarıyla Lloyd adımlamasına dayanan, başka VN tasarım teknikleri de vardır. Bazılarının isimlerini temel fikirleriyle verelim:

(a) *Rassal Kodlama:* Bütün veri vektörlerinin içinden N tanesi rassal olarak seçilir ve kod-vektörler olarak atanır. Bu çok kabaca bir yöntem olmasına rağmen veriler kuvvetli bir şekilde birbiriyle ilintiliyse kabul edilebilir sonuçlar verebilir.

(b) *Ayıklama*[20]*:* Bu yöntemde, veri vektörleri, kod-vektörlerden herhangi birine yeterince yakın olup olmama durumuna göre sırayla kod-kitap listesine eklenirler. Eğer yeni vektör her kod-vektöre en uzak konumdaysa kod-kitaba kod-vektör olarak eklenir [10].

(c) *Parçalı yakın komşu:* Bu algoritma ağırlık merkezleri en yakın olan bölgeleri birleştirir ve birleştirmeye, bölge sayısı istenen kod-kitap büyüklüğünden fazla olduğu sürece devam eder. Başlangıçta her veri vektörü kendi bölgesini oluşturmaktadır, fakat bölgeler daha fazla veri

---

[19]Ağırlık merkezinin hesabı:

Öklit: $v_{Euc} = \frac{1}{M}\sum_{k=1}^{M} x_k$

Manhattan: $v_{Man} = \{x \mid P(x(j) > x(i) = P(x(j) < x(i))\}$

Chebyshev: $v_{Che}(i) = \{\min_{j=1}^{M} x_j(i) + \min_{j=1}^{M} x_j(i)\}/2$

Hamming: $v_{Ham}(i) = \{x_k(i) \mid P(x_j(i)) > P(x_l(i)) \forall l\}$

Mahalanobis: $v_{Mah}(i) = \left[\frac{\sum_{j=1}^{M} C_{x_j}^{-1} x_i^T}{C_{x_j}^{-1}}\right]^T$

[20] Pruning: ayıklama

vektörü içerecek şekilde adım adım büyürler [11]. İki bölgenin birleşimiyle bu bölgelerin ağırlık merkezlerinin ağırlıklı ortalamalarına karşılık gelen yeni bir ağırlık merkezi oluşur. Böylece, önceki yöntemlerden farklı olarak, kod-vektörler herhangi bir veri vektörüne karşılık gelmek zorunda değildir.

(d) *Çarpım kodları:* Eğer kod-kitap büyüklüğü $N = 2^{kR}$ ile temsil edilirse, k tane skaler 2^R seviyeli nicemleyicinin kartezyen çarpımı vektör nicemleyici olarak kullanılabilir [12].

(e) *Rasgele dengelemeli Lloyd adımlaması:* Lloyd algoritmasının her adımında, üretilen sıfır-ortalamalı gürültü ağırlık merkezlerine eklenir ve eklenen gürültünün gücü adımlar ilerledikçe değişir [13]. Örneğin sıcaklık parametresi T_m'ye göre, adım sayısı m arttıkça azalan rassal bir gürültü üretilir ki bu teknik *benzetimli tavlama*[21] olarak da anılmaktadır.

(f) *Benzetimli Tavlama:* Rasgele dengeleme algoritmasında bir ara basamak olarak, gürültü ağırlık merkezlerine eklenir (değişikliğe uğratma, sarsım[22]) ve sarsılmış ağırlık merkezi $P = e^{-\Delta H/T}$ olasılığıyla kabul edilir ki olasılığın hesabındaki ΔH parametresi adım sayısıyla beraber artan bir tür maliyettir.

(g) *Bulanık bölgeleme:* Bir veri vektörünün bir bölge ile kapsanması ikili değerlerle (0 veya 1) belirlenmez. Bunun yerine kapsanma, $\boldsymbol{x_i}$'nin $\boldsymbol{R_j}$ bölgesine üyeliğinin bir göstergesi olan $0 < S_j(\boldsymbol{x_i}) < 1$ bulanık üyeliğiyle belirlenir [21]. Bu yolla, veri vektörünün ilgili bölgeye üyeliği kısmî olur ve yeni bir bulanık bozulma tanımı kullanılır: $D_F = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N} d(x_i, v_j)\left[S_j(x_i)\right]^q$. Lloyd adımlamasında, q parametresi başlangıçta (yüksek bulanıklık belirten) büyük bir sayı olarak seçilir ve adım adım 1'e kadar azaltılır.

(h) *Linde-Buzo-Gray (ayırma-parçalama) algoritması:* Belki de Lloyd adımlamasını kullanan en genel yöntem Linde-Buzo-Gray (LBG) algoritmasıdır [8]. Bu algoritmada, normalde veri vektörlerinin ortalaması olan tek bir kod-vektörle başlanır. Sonra bu vektör, vektör uzayındaki en

[21] Simulated annealing: benzetimli tavlama [1]
[22] Perturbation: değişikliğe uğratma, sarsım [1]. Perturbed: sarsılmış

yüksek değişim yönünde büyüklük olarak daha küçük bir vektörün eklenip çıkarılmasıyla ikiye ayrılır. Bu iki yeni vektörle Lloyd adımlaması uygulanır ve büyüklüğü iki olan kod-kitap yani iki kod-vektörü elde edilir. LBG algoritması adım adım, yukarıdaki "değişikliğe uğratma"yı kullanarak her kod-vektörü ikiye ayırır ve istenen kod-vektör sayısına ulaşılana kadar Lloyd adımlamasını tekrarlar. Bu, eniyi kod-kitabın tümünün en baştan tasarımında, boş ve dengesiz bölge oluşumu riskini taşımayan çok genel bir yöntemdir.

Nicemleyici tasarım tekniğinde başka çeşitler de vardır. Örneğin birinde, girdinin genel yapısına bağlı olarak, değişmeyen yapıda bir nicemleyici yapılmak da istenebilir. Örgü ya da kafes yapılı vektör nicemleyiciler bu amaçlar için çok tutulan çeşittedir ki bölgeler sıklıkla altıgensel yapıda olan geometrik ızgaraya göre seçilir.

Nicemleyici iyileştirmeleri[23] üzerine de geçmişte çalışılmıştır. En sık kullanılan iyileştirmeler şöyle sıralanabilir:

- *Örgü (yapılı) VN:* Gerçekte, yüksek boyutlarda bu nicemleyici düzgün/tek-biçimli (uniform) bir nicemleyicidir. Her nicemleme bölgesi aynı şekle sahiptir. Buna göre bölgeler iki temel koşula uymalıdır. (1) Kesişmemelidirler, (2) Birleşimleri N-boyutlu girdi uzayının tümünü kapsamalıdır. Bu tür yapılara örgü ya da kafes denir.
- *Kazanç-Şekil VN:* Eğer girdi verisi belirgin bir dinamik değer aralığı değişimi gösteriyorsa, makul küçüklükte bir bozulma için kod-kitabın çok büyük olması gerekir. Bu duruma bir çözüm yolu olarak girdi vektörleri önce normalize edilip sonra nicemlenebilir. Normalizasyon faktörü de ayrıca kodlanmalıdır.
- *Ortalaması-silinmiş VN:* Birçok imgede, vektör parçaları benzer şekil özellikleri içerebilir, fakat yoğunluk çeşitliliğinden dolayı, uzaklık metriklerine göre birbirlerine oldukça uzak olabilirler. Bu tür vektörleri aynı kod-vektöre nicemlemek verimliliği artıracaktır. Bu da her vektörden

[23] improvement: iyileştirme

ortalamasının çıkarılarak nicemlenmesiyle mümkündür. Yukarıdaki duruma benzer biçimde ortalama değerler ayrı olarak kodlanmalıdır.

- *Sınıflandırılmış VN:* Eğer girdi verisi, birbirinden büyük uzaysal farklar gösteren birden fazla örüntü içeriyorsa, aynı örüntü parçasından üretilen vektörler pek benzer olacağından, her örüntü için ayrı ayrı nicemleyici tasarlamak [14] ve uygun olan nicemleyiciyi vektöre uygulamak nicemleme verimini artırır [15]. Genellikle iletim ön-bilgisinde, kodlayıcının hangi kod-kitabı kullanacağını belirten bilgi bulunur.
- *Çok-aşamalı VN:* Bu yöntem kodlayıcı karmaşıklığını ve hafıza gereksinimlerini belirgin biçimde azaltır [16]. Yöntem şöyledir: Girdi ilk aşamada kabaca nicemlenir ve girdi ile onun kabaca nicemlenmiş hali arasındaki farkın nicemlenmesine adım adım devam edilir. Örneğin, elimizde üç nicemleyici, Q_1, Q_2, ve Q_3 ile $\boldsymbol{x}$ girdisi varsa

  $$\boldsymbol{y}_1 = Q_1(\boldsymbol{x})$$
  $$\boldsymbol{y}_2 = Q_2(\boldsymbol{x} - Q_1(\boldsymbol{x})) = Q_2(\boldsymbol{x} - \boldsymbol{y}_1)$$
  $$\boldsymbol{y}_3 = Q_3(\boldsymbol{x} - Q_1(\boldsymbol{x}) - Q_2(\boldsymbol{x} - Q_1(\boldsymbol{x}))) = Q2(\boldsymbol{x} - \boldsymbol{y}_1 - \boldsymbol{y}_2)$$

  olsun. Bu durumda nicemleme sonucu $\boldsymbol{y} = \boldsymbol{y}_1 + \boldsymbol{y}_2 + \boldsymbol{y}_3$ olur.
- *Uyarlanabilir VN:* Özellikleri zamanla değişen sinyallerin çevrim-içi kodlanması gibi amaçlar için, uyarlanabilir VN, durumun üstesinden gelecek bir yöntemdir. Genellikle yöntem görece büyük bir kod-kitapla başlar ve kod-kitabın o anki girdi özelliklerine uyan alt-kümesini seçer [17].
- *Trellis-Kodlamalı Nicemleme:* Bir haberleşme konusu olan trellis kodlamalı modulasyon tekniğinden [18] ilham alınarak geliştirilmiş olan bu yöntemde nicemleyici, verilen bir vektör için bir önceki VN çıktısı tarafından belirlenmiş olan kod-kitabı kullanır. Bu halde, bir veri vektörü ancak bir önceki veri vektörünün nicemleyici çıktısından sonra nicemlenebilir.

2.6 VN Uygulamaları ve Örnekler

İşaret işleme alanında vektör nicemleme yöntemlerinin kullanıldığı birçok uygulama bulunmaktadır. Bazıları aşağıda anlatılmaya çalışılmıştır.

2.6.1 Sıkıştırma

VN'nin birincil uygulama sonucu sıkıştırmadır. Muhtemelen büyük miktarlardaki girdi verilerin gösterimi için sınırlı bir kod-kitap kullanımı sonucunda kayıplı (bozulmalı) bir sıkıştırma elde edilir. Teorik olarak bozulma oranı ve bit oranı birbirleriyle ters orantılı niceliklerdir. Sıkıştırma fazla ise bozulma artar fakat bit oranı azalır.

2.6.2 Dönüşüm Nicemlemesi / Tahmini kodlama katsayıları

Normal olarak nicemleme, klasik bir sıkıştırma işleminin ikinci aşamasıdır. Sıkıştırma genel olarak aşağıdaki aşamnalardan geçerek yapılır.

(1) Dönüşüm/tahminî kodlama aşaması

(2) Nicemleme

(3) Entropi kodlaması

İlk aşama olan dönüşüm veya tahminî kodlama işlemleri girdi örnekleri arasındaki bağıntı ve ilintiyi azaltır. Genel olarak kullanılan dönüşümler Kesikli Fourier Döünşümü (Discrete F. Transform: DFT), Kesikli Kosinüs Dönüşümü (Discrete Cosine Transform: DCT, JPEG sıkıştırmada kullanılan dönüşüm), Dalgacık (Wavelet) Dönüşümleri ve girdiye özel dönüşümlerdir (Karhunen-Loeve Transform, Singular Value Decomposition). Bu dönüşümlerin ortak noktaları, girdi vektörleri arasındaki karşılıklı bağıntıyı azalmaktır. Böylece vektör enerjisinin büyük bir kısmı belli vektör elemanları üzerine toplanır. Bu işlemler, ters dönüşümler sayesinde tersinebilirdirler. Bir başka bağıntı azaltan yöntem ise tahminî kodlamadır. Bu yöntemde dizideki bir eleman ilk olarak, önceki elemanlar kullanılarak tahmin edilmeye çalışılır ve bu tahmin gerçek değerle karşılaştırılarak frak değeri çıktı olarak üretilir. Kod-açıcı, tahmin yöntemini ve bu

çıktıları kullanarak aynı tahminî değerleri üretip fark değerleri bunlara ekler ve sinyali yeniden oluşturabilir.

Bu yöntemlerin herhangi birinden sonra sinyal örnekleri çoğunlukla sıfıra nicemlenebilecek küçük değerler alırlar. Sinyal enerjisinin yoğunlaştığı az sayıdaki örnekler ise sıfırdan uzak değerler alırlar. Böylece nicemlemenin verimliliği büyük oranda artırılmış olur. Dönüşümü alınmış olan sinyal vektör nicemlenir ve sonraki aşamalara geçilir.

2.6.3 Sinyallerin Doğrudan Vektör Nicemlenmesi

Bölümün başında anlatılan teknikler herhangi bir ön işlem olmaksızın da sinyale uygulanabilir. Örneğin, 256x256'lık bir sayısal gri-seviyeli resim 8x8'lik bloklara ayrılarak bu bloklar girdi vektörleri olarak kullanılırsa yukarıda anlatılan kod-kitap tasarımı teknikleri ile bir kod-kitap oluşturulabilir. Böylece 256x256'lık bir resim 1024 adet 8x8'lik bloğa ayrılmış olur. 1024 adet girdi bloğu istenen seviyeye (32, 64, 128, ...) nicemlendikten sonra her bloğa karşılık gelen kod-vektör indisi bloğu temsil etmiş olur. Örneğin 32 bloğa nicemleme yapılmış olsun. 32 adet kod-vektör ve 1024 adet de indis kullanılmış olur. Böylece söz konusu resim 256x256 piksel verisi yerine sadece 32 adet 8x8 piksel bloğu verisi ve 1024 adet de [1, 32] aralığında tamsayı indisleri ile temsil edilmiş olur.

2.6.4 Sınıflandırma ve Kümelendirme

Vektör nicemlemenin farklı bir uygulaması da sınıflandırmadır. Bir çok örüntü tanıma uygulamasında, girdi verilerinin otomatik kümelenmesi, hangi girdi vektörü özelliğinin hangi kümede toplandığı bilgisini de sağlar. [19] Böylece farklı özellikteki girdiler birbirlerinden ayrışarak kümelenirler. Benzer özellikleri taşıyan girdiler aynı ya da birbirine yakın kümelerde bulunur. Vektör nicemlenen veriler kümelenir ve kendi kümelerini temsil eden kod-vektöre nicemlenirler. Ayrıca kod-vektörlerin birbirlerine yakınlık bilgileri de kodlayıcı ve kod-açıcı birimler tarafından bilindiğine göre otomatik kümeleme yapılmış olur. Bu da örüntü tanıma için çok değerli bilgiler sağlar.

2.6.5 Renk Azaltımı

Bir 24-bit gerçek renkli resim her pikseli için ayrı kanallarda üç farklı renk bilgisi içerir. Bu kanallar kırmızı (R: red), yeşil (G: green) ve mavi (B: blue) kanallarıdır. Her kanal verisi 8 bitle ifade edilir. Normal bir resimde, insan gözü bu kanallarda oluşabilecek birçok renk değişimini farketmez. Bu nedenle gösterim fazlalığı ortaya çıkar. Bazı bilgisayar ekran donanımları ve yazıcılar da bu renklerin çoğunu gösterebilecek ve üretebilecek kapasitede değildir. Bu gibi nedenlerle renk sayısının 2^{24}'ten belli bir sayıya örneğin $2^8 = 256$'ya indirgenmesi arzu edilen bir uygulamadır.[20] Böyle bir durumda her renk kanalı (R, G, B) bir eksenle ifade edilerek üç boyutlu bir renk uzayı oluşturulur ve bu renk uzayında renklerin kümelenme bilgisi kullanılarak nicemeleme yapılabilir. Renklerin kümelenişlerine göre uygun vektör nicemleme tekniği kullanılarak 256x256x256 farklı renk girdi vektörü 256 adet renk kod-vektörüne nicemlenebilir. GIF resim formatında bu nicemleme kullanılmaktadır [21]. Resmin renk haritası (colormap) denilen bir renk ön bilgisi ve her piksel için bu haritadaki bir renge işaret eden bir tamsayı listesi bulunur ki bu renk haritası aslında oluşturulan renk kod-kitabıdır ve [0,255] aralığındaki tamsayılar da renk kod-vektörlerinin indisleridir.

3. UYGULAMA

3.1 Önden Çekilmiş Yüz Resimleri

Çalışmada kullanılmak üzere saha çalışması yapılarak bizzat çekilmiş çok sayıda sayısal fotoğrafla bir arşiv oluşturuldu. Sayısal fotoğrafların çekiminde Canon Powershot Pro1 ve Canon Powershot A80 sayısal kameraları kullanılarak 768 x 1024 (en x boy) piksel boyutlarında jpeg biçiminde kayıtlı dosyalar kullanıldı. Çekilen resimler önden profil ve vesikalık biçimine yakındır. Çekilen sayısal resimlerde, kişinin yüzünün doğrudan kameraya dönük olduğu ve saçların kapatmadığı yüz hatlarını ön plana çıkaran nötr bir yüz ifadesi elde edilmeye çalışıldı. Bütün resimlerde yoğunluk değerlerinin (intensity) fotoğrafa mümkün olduğunca denk dağılması için çift yönlü aydınlatma ile flaş patlatılarak çekim yapılmaya çalışıldı.

3.2 Uygulamanın Aşamaları

Gerçekleştirilen uygulama, yüz üzerindeki bir takım referans bölgelerini otomatik olarak tespit ederek bu bölgeleri yüz resmi üzerinden kesip çıkararak ayrı bir yerde kodlama yapmaktadır. Bu bölgelerin bulunması ve çıkarılması için,

- Ele alınan genel yüz resmi üzerinde ilk aşamada bazı referans noktalarının (her bir göz için orta nokta, burun, ağız, vs) tespit edilmesi,
- Bu noktaların birbirlerine olan mesafe ve eğimlerine göre resmin "normalize" edilmesi (küçültme/büyütme ve döndürme-çevirme), ve
- Tespit edilen nokta civarında ortalama bir görüntü maskesi ile küçük organ resimlerinin ayrılması

gibi tespit aşamalarına ihtiyaç vardır. Bu aşamalar bu çalışmada anlatılmaktadır.

3.3 Görüntü Veri Biçimi

Geliştirilen uygulama jpeg biçimli görüntü dosyalarını açabilecek şekilde Microsoft Visual Studio 6.0 C++ MFC kod kütüphanesi ortamında ve Intel jpeg decoder kütüphanesi kullanılarak yazıldı.

Açılan jpeg resmi her piksel için 24-bit (RGB:8-bit R, 8-bit G, 8-bit B) veri bulunduran tek boyutlu dizi şeklinde bellekte tutulur. Açılan bir görüntü verisi CImage sınıfı ile oluşturulan c++ nesnesi olarak bellekte tutulur. Bu sınıf tanımlamasına göre her CImage nesnesinin 'Data', 'Height', 'Width', 'IMType' özellikleri vardır. Bu sınıf ve yazılan kodlar temelde [22] ve [23] kaynaklarındaki algoritmaların ve kodların uyarlanarak c++ ile yeniden yorumlanmasıyla oluşturulmuştur. Gerekli görüldükçe geliştirilmiş olan fonksiyonların listesi de Ekler kısmında listelenmiştir. CPIXEL, CVector ve CVQ sınıfları da kod yazılırken geliştirilmiş sınıflar olarak Ekler kısmında incelenebilir. Ayrıca yine görsel arabirim tasarımında geliştirilen sınıfların ve fonksiyonların tanımları da yine ektedir.

CImage.Data : Görüntü verisini tek boyutlu matris halinde tutan diziye göstergedir.

CImage.Width : İki boyutlu yatay genişliği piksel sayısı olarak belirtir.

CImage.Height : İki boyutlu dikey genişliği piksel sayısı olarak belirtir.

CImage.IMType : Verinin tutulduğu renk tabanını (RGB, YCbCr, vb.) belirtir.

3.4 Görüntüde Yüz Bölgesinin Tespiti İçin Bir Yöntem

İlk aşama olan ten rengine dayalı yüz bölgesi tahmini için, YCbCr renk tabanının yüz renginin tespiti ve konvolusyon, normalizasyon, ortalama değer bulma vs. piksel ve renk işlemlerinde Y kanalının sağladığı kolaylık göz önüne alınarak, eldeki RGB renk dizisi bu renk biçimine dönüştürülür. Bu dönüşüm için **Şekil 3.1'** deki dönüşüm matrisi kullanılmaktadır.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16874 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix}$$

Şekil 3.1 YCbCr ↔ RGB dönüşüm matrisleri

Y kanalının yukarıda belirtilen işlemlerde sağladığı kolaylık, dikkat edilecek olursa gri-yoğunluk değerini göstermesinden kaynaklanmaktadır. Resim üzerindeki renk verisini salt gri-yoğunluk seviyesine çevirerek geri dönüşümsüz olarak kaybetmektense Cb ve Cr kanallarında saklayarak gri-yoğunluk kanalının sağladığı kolaylıklardan yararlanılmış olunur. Cb ve Cr kanalları kabaca mavilik ve kırmızılık değerlerini belirtir. Sonuçta bu gösterim, eğer parlaklık çok baskın değilse, parlaklıktan görece bağımsız bir şekilde renk taraması yapılmasını sağlar.

Renk aralığı için [8]'deki deneysel sonuçların yanısıra çekilmiş olan resimlerin incelenmesi sonucunda yüz rengi aralığı renk uzayında üç boyutlu bir küme olarak tahmin edilerek bu YCbCr renk kümesine giren piksel değerleri işaretlendi. Resimlerin incelenmesi ile elde edilen aralıklar **Tablo 3.1**'de belirtilmiştir.

	Cb	Cr
[8]'de kullanılan değer aralıkları	[77,127]	[133,173]
Deneysel olarak seçilen değer aralıkları	[77,145]	[83,125]

Tablo 3.1 Olası ten rengi için seçilen Cb, Cr kanal aralığı

Bu işlemin daha kısa sürmesi için resmin örneklenerek küçültülmüş bir sürümü üzerinde arama-işaretleme yapılıp, tespit edilen tahmini bölge koordinatları, orijinal resim boyutlarına uyarlanır. Bu işlemin uygulandığı resimlere iki örnek **Şekil 3.2**' de görülmektedir.

Şekil 3.2 Ten rengi ile yüz bölgesinin tahmini

Yüz rengine yakın renkte başka nesnelerin de olabileceği hesaba katılmalıdır. Nitekim **Şekil 3.2**'de ikinci resim grubunda parlaklık ve saçların sarı tonlu olması yüzün işaretlenmesi işlemini olumsuz etkiler. Parlaklık değerinin yüksek olması nedeniyle renk bileşenine baskın gelmesi durumunda da renk ayrımı yapılamaz. Bu gibi nedenlerle renge dayalı bölütleme işleminin güvenilir biçimde kullanılamadığı fark edildi. Bu nedenle ilk uygulamalarda kullanıcının fare yardımıyla elle işaretleyerek gözlerin yerlerini vermesi sağlanmıştır.

3.5 Gözlerin İşaretlenmesi

Gözlerin işaretlenmesi işlemi kullanıcının görüntü üzerinde iki nokta seçmesiyle gerçekleştirilmektedir. Göz yerlerinin tespitinde kullanıcı yardımı alınmış olması yüzün tanınması için gereken bilgisayara ait işlem zamanını büyük ölçüde kısaltmaktadır. Belirtilmiş olan iki nokta civarında aşağıdaki ön işlemler gerçekleştirilerek göz merkezlerinin belirginliği artırılmaya çalışılmıştır.

Tıklanan noktanın koyu olan göz irisi içinde olduğu kabul edilerek noktanın ya da irisin iki yanındaki göz aklarına ulaşılmaya çalışılmıştır. Göz akı olabilecek pixellerin işaretlenmesi ise RGB tabanında yapılan beyaz(255,255,255) renge çevresindeki piksellere oranla yakınlıklarına göre yapılmıştır. Salt beyaz renge yakınlık:

$$sby = \sqrt{(R-255)^2 + (G-255)^2 + (G-255)^2} \qquad (3.1)$$

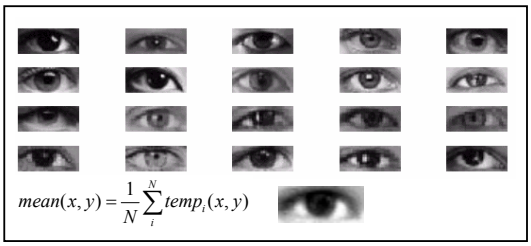
Ancak göreceli karşılaştırma yapıldığı için bu uzaklık metriği duruma uyarlanmış ve 255 değeri yerine noktanın belirli komşuluğundaki (yaklaşık 30 piksellik alandaki) noktaların gri-yoğunluk ortalamaları konularak karar verilmiştir. Bu

sayede çevresinden belli bir eşik değerini aşacak şekilde parlak olan pikseller göz akı olarak işaretlenmiştir. Bundan sonra noktanın dikeyde beş piksel uzaklığına kadar bakılarak iki ak bölge arasının en fazla olduğu satırın orta noktası yeni merkez olarak alınmıştır. Burada amaç, göz aklarının birbirlerine en uzak olduğu satırı yani göz yuvarlağının en geniş olduğu satırı bulmaktır.

Göz merkezleri belirginleştirildikten sonra sıra iki merkezin aynı yatay hizaya getirilmesine gelir. İki merkezin ortası referans kabul edilerek resim açısal döndürme işlemine tabi tutulur. Bu küçük ayardan sonra aynı orta nokta kullanılarak yüz oranlarına uygun biçimde üzerinde işlem uygulanan resim dikdörtgensel biçimde kırpılır. Yani yüz bölgesi içeride kalacak şekilde resim kenarlarından kesilir. Bu kesme işleminden sonra yine yüz oranlarının kullanımıyla (iki merkez arasındaki uzaklık kullanılarak) göz alanları daraltılır. Bu işlem bize sadece gözleri ve yakın çevresini içine alan (sağ ve sol göz için ayrı ayrı) iki dar dörtgen alanı verir. Bu dörtgenler ileride şablon olarak kullanılmak üzere ayrılarak saklanacaktır. Gözlerin kesilip ayrılmasından sonra sıra kaşların, burnun ve ağzın tespit edilerek kesilip ayrılmasına gelir. Kaşlar gözlerin hemen üstünde ten renginden farklı olarak koyulaşan bölgeler olarak kolayca tespit edilip ayrıştırılabilirler. Burnun tespitinde kenar bulma algoritmaları devreye girer. Burada karar verme mekanizması göz merkezlerinin orta noktasından aşağıya doğru inerken karşılaşılan keskin kenarlara göre belirlenir. Kabaca ilk keskin kenar burnun alt sınırına daha aşağıdaki ikinci kenar ise dudakların başlangıç noktasına denk gelecektir. Bu karar mekanizmalarının uygulanmasıyla burun bir çokgen içine alınır ağız ise dudakları içine alan bir dörtgen ile sınırlandırılır.

3.6 Gözler İçin Şablon Kullanımı

Şablon eşleme sırasında uygulanan yöntem şöyledir: daha önceden manüel olarak işaretlenerek bölütlenmiş ve kesilmiş olan eğitim gözleri kullanılarak ortalama göz şablonları oluşturulur. **(Şekil 3.3)** göz şablonlarının birkaç tane (karar verilen kod vektör sayısına ulaşana kadar) olması taramanın güvenirliğini artırır. Ancak şablon taram işleminden önce yine göz bölgelerinin tıklanması istenmektir.



Şekil 3.3 Elde edilen göz resimlerinden ortalama şablon elde etme.

3.6.1 İşaretlenen Alanın Taranması

Göz şablonu ile seçilmiş olan iki nokta civarındaki sınırlı alan korelasyon işlemine alınır. Şablon yeniden ölçeklenerek ve küçük açılarla (<5°) saat yönünde ve tersinde çevrilerek her ölçekleme ve çevirme sonrasında konvolusyon tekrarlandı. Konvolusyon öncesi taranacak alanın ve şablonun dc değerlerinin sıfırlanması ve yoğunluk değerlerinin normalize[24] edilmesi korelasyon kalitesini artırdığı için uygulanan ara işlemlerdir. Yapılan taramalar sonucunda belli bir korelasyon eşik değerini aşmış olan noktalar işaretlenerek gözün tam yeri tespit edilmeye çalışılır. **(Şekil 3.4)**

Şekil 3.4 Solda elle işaretlenmiş gözler, sağda ise tespit edilen noktalar görülmektedir

Sağ göz için yapılan işlemler benzer biçimde sol göz için de şablon oluşturularak yapılır ve gözlerin tam konumları tespit edilir. Birbirine yakın olan noktalar kümesi olarak iki farklı kümelenme ortaya çıkmaktadır. Bunlardan biri sağ göze ait tahmin kümesi diğeri ise sol göze ait tahmin kümesidir. Bu

[24] Normalizasyon: Bir sayısal veri dizisindeki bütün değerlerin mutlak [0,1] aralığında olacak şekilde yeniden ölçeklenmesi. Dizideki en büyük değer 1'e ölçeklenir.

tahminlerin yüz resimlerinde gerçekte gözlerin yerlerini verdiği yapılan denemelerle gözlerin tespitinin büyük oranda yapıldığı görülmektedir.

3.7 Çevirme ve Normalizasyon

Göz merkezlerinin tespitinden sonra bütün resim, göz merkezlerinin orta noktası referans alınarak, gözler yatayda aynı hizaya gelecek biçimde çevrilir.

Şekil 3.5 İki göz merkezi arasına çizilmiş çizgi ve resmin yataya hizalanmış hali

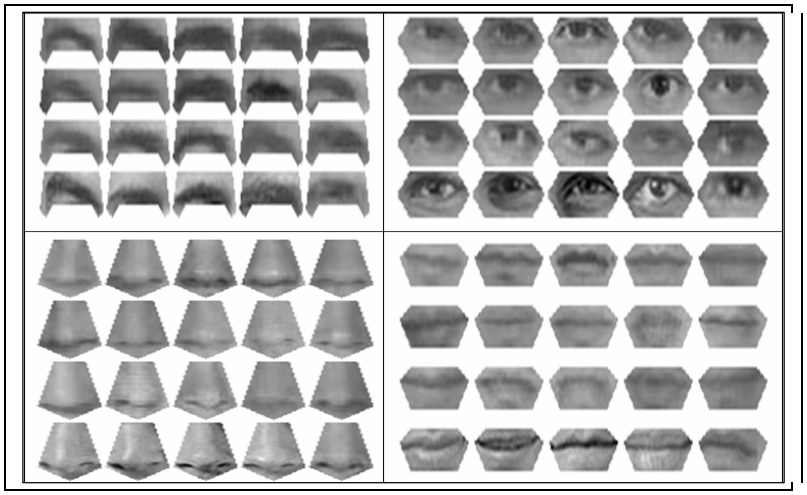
Sonraki işlem bu iki göz merkezi arasındaki düz çizgiyi normalize etmektir. Taranan bütün yüz resimlerinde bu çizgi eş uzunlukta olacak biçimde yeniden boyutlandırma/ölçeklendirme yapılır. Böylece resimler yüze uygulanacak olan maskeye de uygun hale gelmiş olur.

3.8 Maske Uygulanması

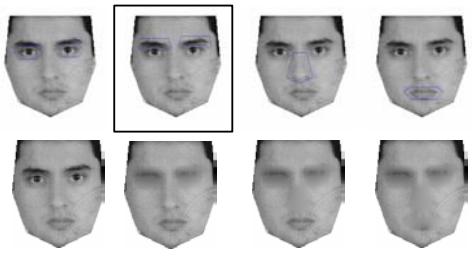
Maske gözleri referans alarak yüzün geneline uygulanan iki boyutlu bir katmandan ibaret olarak görüntünün üstüne oturtulur ve gözlerin yerlerine bağlı olarak diğer yüz elemanlarının istatistiksel yerlerine gore bölütleme yapar. Burada kullanılan maske insan yüzünün simetrisi ve yüz organlarının birbirlerine olan doğal oranları kullanılarak oluşturulmaktadır. Sonuçta birkaç farklı maske formu kullanılarak uygun maske seçimiyle sadece gözlerin tespiti neticesinde kaş, burun ve ağız da bölütlenmiş olmaktadır. Açıkta kalan bölgeler kaşlar, burun ve ağız olarak kesilir.

Şekil 3.6 Sadece gözlerin tespiti yüzün diğer ögelerini saptamaya yeterlidir.

Bu maske uygulandıktan sonra kesilen yüz parçalarından gri-seviye bazı örnekler **Şekil 3.7**'de görülmektedir.



Şekil 3.7 Kesilen öge dizinleri



Şekil 3.8 Parçaların kesilmesiyle kalan kısmın doldurulması

Daha sonra geri kalan yüz resminin (**Şekil 3.8**'de ikinci sıradaki son resim) entropisinin düştüğü gözlenerek ayrı bir yöntemle kodlandı. Ayrıca yüz

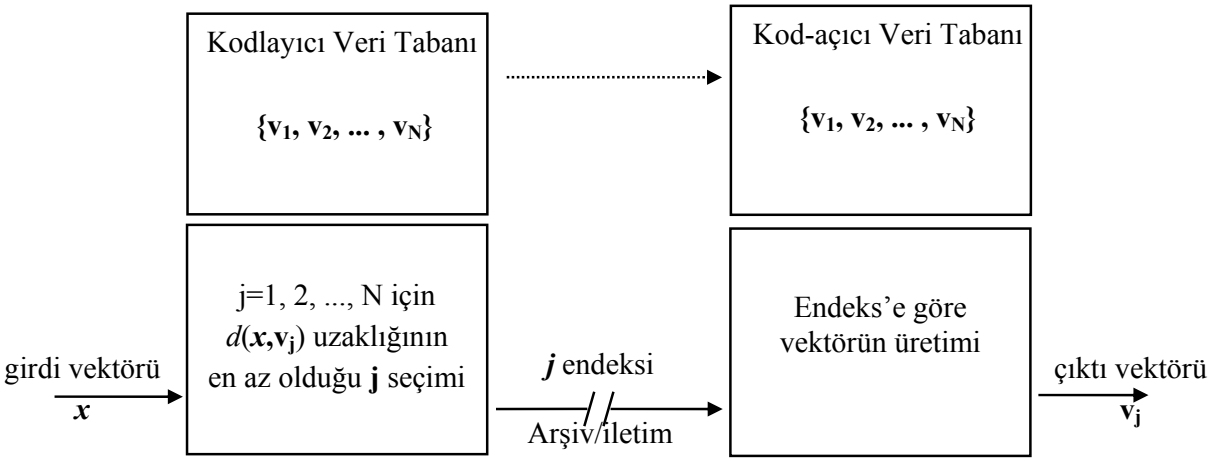
bölütlerinin ayrı ayrı kodlanması kolay bir görüntü geri elde etme sistemine ulaşılmasını sağladı. Yüz resmindeki belirleyici özellikler olan organların vektör nicemlenmesiyle yüz benzerlikleri bilgisine karşılık gelen dizinler elde edildi.

3.9 Arşivleme (VN kullanımı)

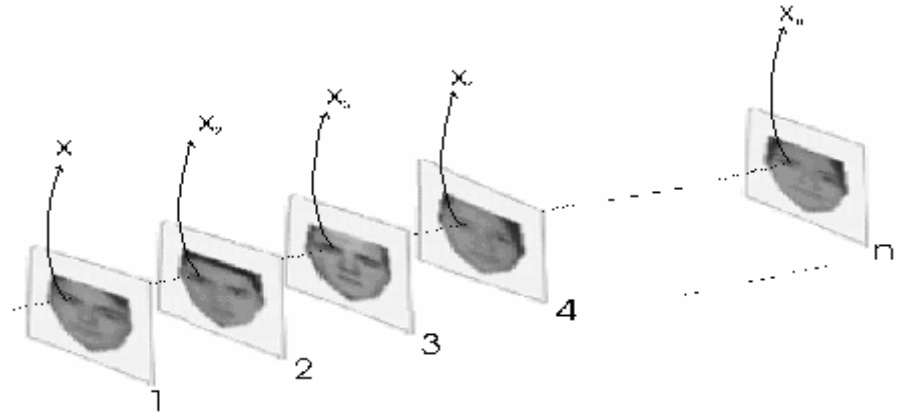
Bölütlenen ve yüzden çıkarılan yüz kısımlarının sıkıştırılmasında vektör nicemleme [9] yöntemleri kullanıldı. Kod-kitabı oluşturma aşamasında altı farklı kod kitabı düzenlendi.

- Sağ gözler kod-kitabı
- Sol gözler kod-kitabı
- Sağ kaşlar kod-kitabı
- Sol kaşlar kod-kitabı
- Burunlar kod-kitabı
- Dudaklar kod kitabı

Arşive alınacak yüz resmi sisteme girildiğinde sıkıştırılmış organsız yüz parçasıyla birlikte yukarıdaki kod-kitapların endekslerini bulunduran bir veri yapısında arşivlenmiştir. Bu durumda entropisi düştüğü için yüksek oranda sıkıştırılmış organsız bir yüz parçasıyla birlikte altı endeks sayısı büyük bir sıkıştırma kazancı getirmektedir.



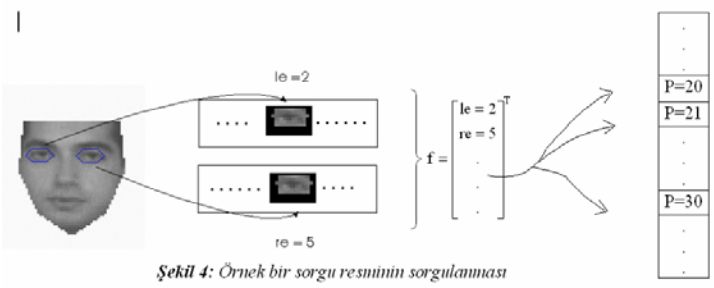
Şekil 3.9 Vektör Nicemleme işleminin kodlama ve kod çözme aşaması.



Şekil 3.10 VN sırasında mesafeleri ölçülen ve ortalamaları alınan piksel örnekleri (sağ gözler).

3.10 Arşivden Geri Çatım ile Yüzün Oluşturulması

Her organ resminin, arka plan resmi ile birleştirilmesi sonucunda kişiye ait yüz resmine geri ulaşılabilmektedir. Öte yandan bu organ resimlerinin her yüz resmi için tek tek kodlanması, hem sıkıştırma verimliliği açısından, hem de kodlanmış resmin kime ait olduğu bilgisinin elde edilebilmesi açısından zorluk çıkarmaktadır. Bu nedenle, her yüz için aynı organa karşılık gelen resimlerin gruplanarak kodlanması daha uygundur. Bu gruplama sırasında Vektör Kuvantalama kodlayıcısı kullanarak hem az sayıda "bit" ile gösterim imkanı elde edilmiş, hem de kodlanmış yüz resmi bir takım katalog VN indeks numaralarından oluştuğu için, resimlerin birbirlerine benzer olup olmamaları açısından veri çıkarımına olanak sağlamıştır (**Şekil 3.11**).



Şekil 4: Örnek bir sorgu resminin sorgulanması

Şekil 3.11Yeni bir yüz resminin organ kısımlarının mevcut resim veri tabanı içinde sorgulanması

VN indeksi veri tabanından alınan yüze ait herhangi bir organ görüntüsü için geri çatma işlemi, geri plan yüz resmi üzerine bu VN kod-kitabı resmini organa ait fiziksel koordinatlarda yapıştırmaktan ibarettir. Bu yapıştırma sırasında uç kenarların örtüştürülerek birbirlerine kaynaşacak şekilde yumaşatılması düzgün bir görüntü için gereklidir.

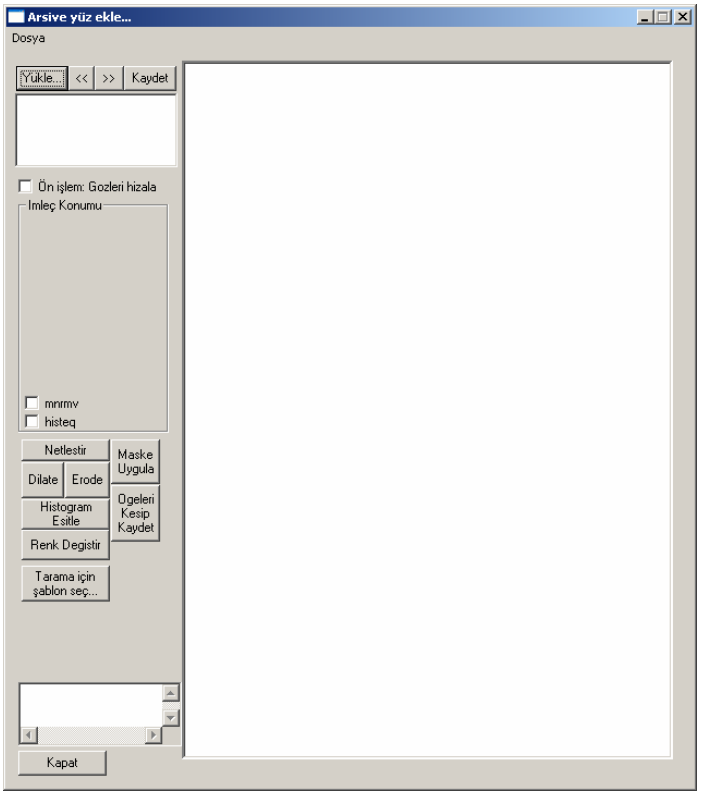
4. YAZILIM VE SONUÇLAR

4.1 Yazılım Görünümü ve Aşamalar

Programın ana penceresi (**Şekil 4.1**) bir araç çubuğu şeklinde tasarlanmıştır.

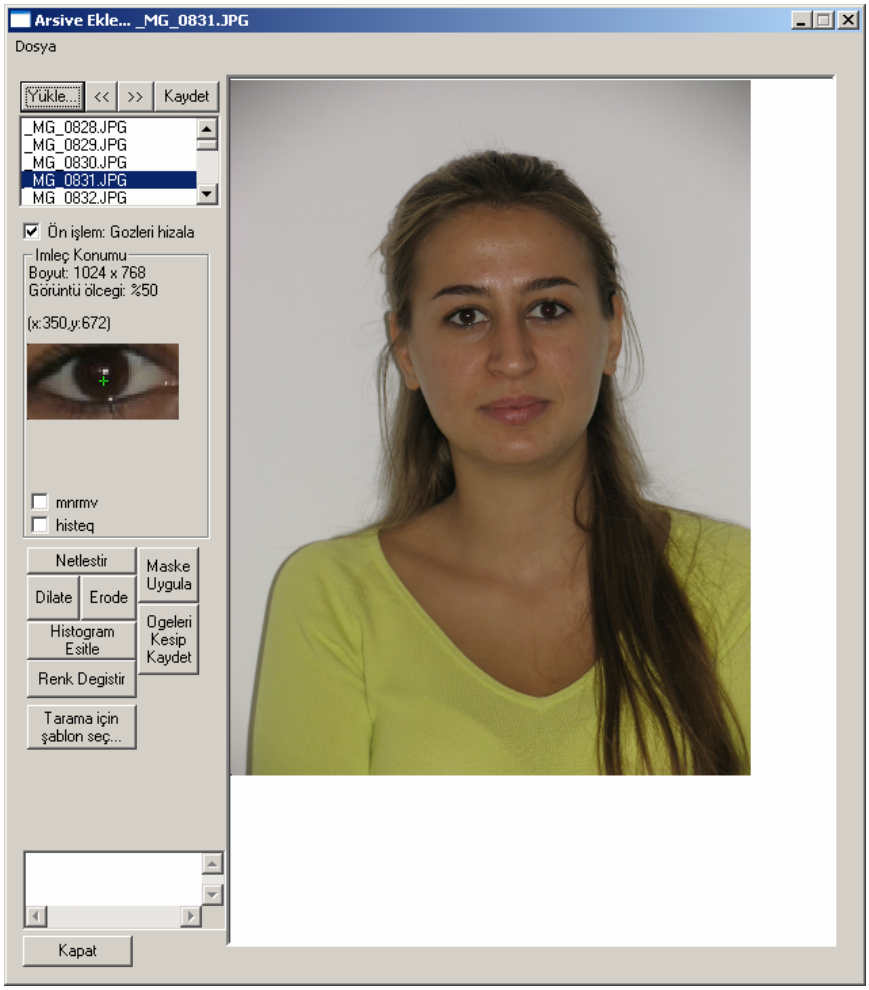
Şekil 4.1 – Ana pencere

'Basla...' butonuna basıldığında ekrana yeni bir fotoğrafın seçilmesi ve arşive eklenmesi için gerekli adımların yapıldığı 'Arsive yüz ekle...' penceresi gelir.



Şekil 4.2 'Ekle' penceresi

Arşive yeni bir yüz eklemek için resim dosyasının seçilip yüklenebileceği ‘Resim dosyası seç...’ butonuna basılır. Gelen seçim penceresinden yüz resmi dosyası bulunarak seçilen resim yüklenerek pencerede görüntülenir.



Şekil 4.3 ‘Ekle’ penceresinde yüklenmiş resim

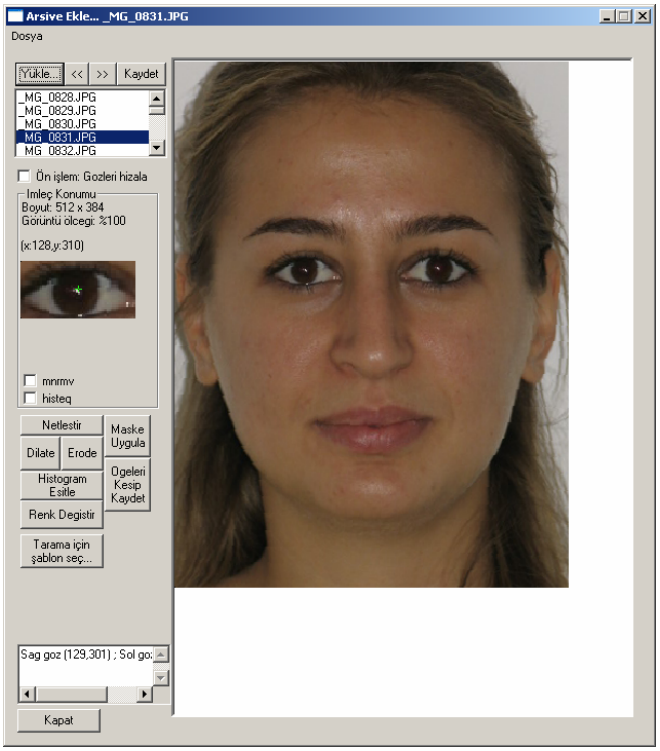
Bir sonraki aşamada görüntülenen yüz resmindeki gözlerin kullanıcı tarafından fare yardımıyla işaretlenmesi gereklidir. Bu işlem sağ ve sol göz için ayrı olarak yapılır. Bu uygulama penceresinde soldaki ‘İmleç Konumu’ bölümünde fare imlecinin üzerinde olduğu resim dörtgeni büyütülerek ayrıntılandırılır. Sağ gözün üzerinde sağ fare tıklaması yapılırsa sağ-tık menüsü açılarak **Şekil 4.4**’teki gibi görüntülenir.

Şekil 4.4 Sağ gözün fare yardımıyla işaretlenmesi

Sağ gözün işaretlenmesinden sonra sol göz için de aynı işlem adımları yapılır.

Şekil 4.5 Sol gözün fare yardımıyla işaretlenmesi

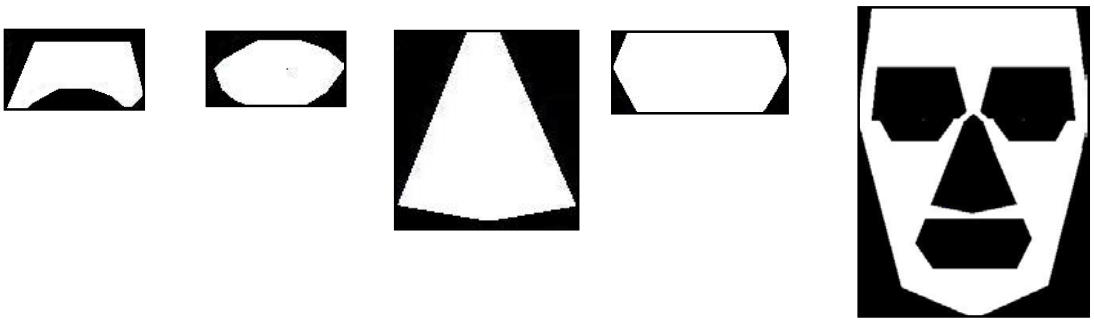
Gözler işaretlenmeden önce gerekiyorsa gözlerin hizalanması için yüzün açısal olarak çevirilmesi ön işlemini yaptırmak amacıyla "İmleç Konumu" bölümünün hemen üzerindeki kutucuğun işaretlenmesi uygun olacaktır. Sağ-tık menüsüyle sağ ve gözlerin işaretlenmesi ile, herhangi bir ek işlem yapılmaksızın, resim işaretlenmiş olan göz konumları aynı hizaya getirilir ve bu göz hizalama işlemiyle birlikte belirli yüz oranları da kullanılarak yüz genel hatlarıyla büyük resimden ayrılır ve ekranda görüntülenir. **Şekil 4.6**'de görülen yüz özgün resmin saat yönünde yaklaşık dört derece çevrilmesi ile alınmıştır.



Şekil 4.6 Çevirme ve işlem alanının yüze doğru daraltılması

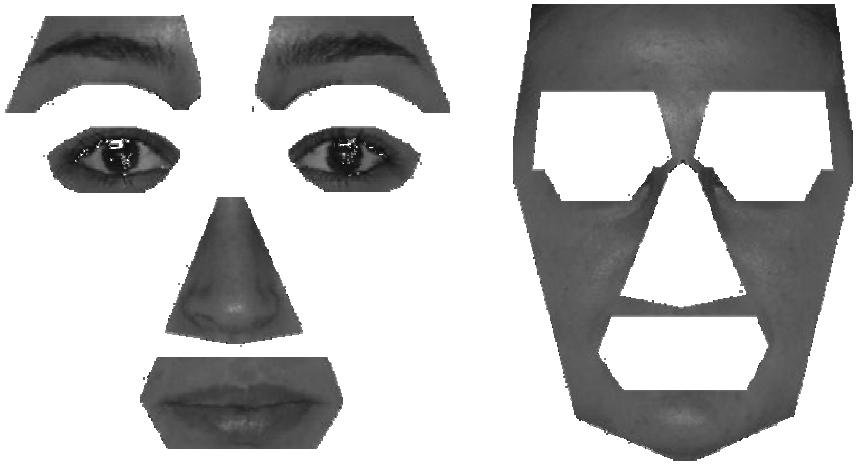
4.2 Maskeleme Aşaması

Gözlerin hizalanması ve yüzün genel hatlarıyla özgün görüntüden ayrılması aşamasından sonra zaten bilinen göz konumları kullanılarak daha önceden arşiv resimleri incelenerek deneysel olarak tespit edilmiş oranların kullanılmasıyla gözlerin ve diğer organların yüzden kesilmesi işlemi gerçekleştirilir. Bu kesme ve ayırma işlemleri için uygun maskeler (**Şekil 4.7**) tasarlanıp kullanılmaya çalışılmıştır.



Şekil 4.7 Yüz parçalarını ayırmak için kullanılan maskeler

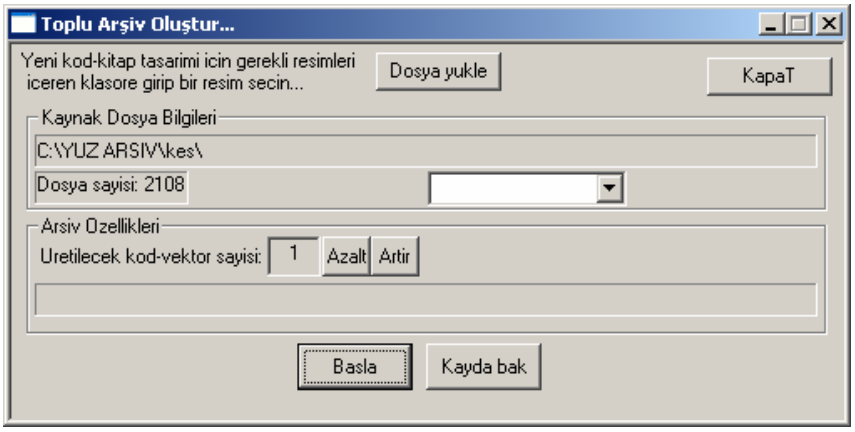
Bu maskeler deneysel yöntemler kullanılarak arşivdeki yüz resimlerinin incelenmesi yoluyla tasarlanmıştır. Yukarıdaki uygulama pencerelerinde üzerinde işlem yapılan "_MG_0831.jpg" dosyası ön işlemler ve maskeleme sonucunda yedi ayrı parçaya ayrılmıştır.



Şekil 4.8 '_MG_0831.jpg' dosyasındaki yüz görüntüsünün parçalarına ayrılmış hali

4.3 Toplu Arşiv Aşaması

Önceki bölümlerdeki göz konumlama ve yüz kesme adımlarıyla arşivden seçilmiş 300 adet yüz görüntü yüz organlarına ayrılmış ve oluşan 2100 adet organ parçası nicemleme işlemlerine girdi olarak kullanılmıştır. Bu kesilmiş olan parçalar ve arşiv dosyaları geliştirilmiş olan yazılımla birlikte CD içeriğinde mevcuttur.



Şekil 4.9 Kod-kitap tasarım penceresi

Arşivde bulunan yüz görüntülerinden oluşturulan organ parçaları veri tabanını kullanan ve uygulama ana penceresindeki 'Toplu Arsiv' butonuyla çağırılan alt birimin kullanılmasıyla da kod-kitapların ve kod-vektörlerin üretilmesine çalışılmıştır. Oluşturulan herbiri 300 adet olan sağ kaş, sol kaş, sağ göz, sol göz, burun, dudaklar ve geri kalan kısım ile vektör nicemleme ile 32, 64 ve 128 seviyeli kod-kitapların tasarlanma işlemlerinde uzaklık metrikleri olarak öklit uzaklığı kullanılmıştır ve uzaklık hesaplaması yapılmadan önce oluşturulan girdi vektörlerinin ortalama değerleri çıkarılarak ayrı şekilde saklanmıştır. Ortalama değerlerin görüntülerden çıkarılması uzaklık metriklerinin daha etkili ve verimli çalışmasını sağladığı için tasarlanan kod-kitaplar optimuma yaklaştırılmıştır.

4.4 İleri Çalışmalar

Bu uygulama ve anlatılan yöntemlerle, yüz görüntülerinin arşivlemeye uygun bir şekilde sıkıştırılmasına yönelik bir proje uygulama ile hayata geçirilmiş önceki çalışmalarla birleştirilen proje birikimi ile

1. Görüntü veri tabanı oluşturulmuş ve
2. Simülasyon çalışmaları büyük ölçüde gerçeklenmiştir

Uygulamanın geliştirilerek daha gürbüz bir biçime getirilmesi ve otomatik çalışır hale getirilmesi ileri araştırma ve çalışmalar olarak önerilmektedir.

İleri çalışmalar için web sayfası:
http://www.mm.anadolu.edu.tr/eee/phd_users/sgorgulu/yltez/

KAYNAKLAR

[1] Sankur B. ve Güleryüz H., *"Bilişim Sözlüğü 2005"*, www.bilisimsozlugu.com, 2005

[2] Gerek Ö. N. ve Çetin A. E., *"Vector Quantization"* yayınlanacak, Wiley Encyclopedia of Biomedical Engineering, 2006

[3] Ball G. H. ve Hall D. J., *"Isodata, a novel method of data analysis and pattern classification"*, Stanford Research Institute Report, Stanford, CA, A.B.D., 1965

[4] Patrick E.A. ve Fischer III F.P., *"A generalized k- nearest neighbor rule"*, Information and Control, **16(2)**, 128-152, 1970

[5] Kohonen T., *Self Organization and Associative Memory*, 3. Baskı, Springer-Verlag, Berlin, Almanya, 1989

[6] Linde Y., Buzo A. ve Gray R. M., *"An algorithm for vector quantizer design"*, IEEE Trans. on Communications, **28**, 84-95, Ocak 1980

[7] Shannon C. E., *"Coding theorems for a discrete source with a fidelity criterion"*, IRE National Convention Record, **4**, 142-163, 1959

[8] Ngan K. N. ve Chai D., *"Face segmentation using skin color map in video phone applications"*, IEEE Trans. On Circuits and Systems, **9(4)**, 1999

[9] Gerek Ö. N. ve Çınar H., *"Segmentation based coding of human face images for retrieval"*, Signal Processing, **84(6)**, 1041-1047, Haz. 2004

[10] Jain A. K. ve Dubes R. C., *Algorithms for clusterin data,* Prentice-Hall Inc., A.B.D., 1988

[11] Equitz W. H., *"A new vector quantization clustering algorithm"*, IEEE Trans. on A.S.S.P., 1568-1575, Ekim 1989

[12] Sabin M. J. ve Gray R. M., *"Product code vector quantizers for waveform and voice coding"*, IEEE Trans. on A.S.S.P., **32**, 474-488, Haz. 1984

[13] Zeger K. ve Gersho A., *"A stochastic relaxation algorithm for improved vector quantiser design"*, Elecronics Letters, **25**, 896-898, Tem. 1989

[14] Gray R. M., *Source Coding Theory*, Kluwer Academic Press, Boston, A.B.D., 1990

[15] Ramamurthi R. ve Gersho A., *"Classified vector quantization of images"*, IEEE Trans. on Communications, **34**, 1105-1115, Kasım 1980

[16] Juang B. H. ve Gray A. H., *"Multiple stage vector quantization for speech coding"*, Proc. Intl. Conf. on A.S.S.P., IEEE, 597-600, Nisan 1982

[17] Panchanathan S. ve Goldberg M., *"Adaptive algorithm for image coding using vector quantization"*, IEEE Signal Processing: Image Comm., **4**, 81–92, 1991

[18] Viterbi A. J. ve Omura J. K., *Principles of Digital Communications and Coding,* McGraw Hill, New York, A.B.D., 1979

[19] Fraley C. ve Raftery A. E., *"How many clusters? Which clustering method? - Answers via model based cluster analysis",* Computer Journal, **41**, 578–588, 1998

[20] Heckbert P. S., *"Color image quantization for frame-buffer display"*, ACM Computer Graphics (ACM SIGGRAPH '82), **16(3)**, 297- 307, 1980

[21] Murray J. D. ve VanRyper M., *Encyclopedia of Graphics File Formats,* O'Reilly and Associates, Inc., Sebastopol, CA, A.B.D., 1994

[22] Myler H. R. ve Arthur R., *The pocket handbook of image processing algorithms in C*, Prentice Hall/PRT, A.B.D., 1993

[23] Pitas I., *Digital Image Processing Algorithms and Applications*, WILEY Yayıncılık, A.B.D., 2000

EKLER

EK 1: GELİŞTİRİLEN YAZILIM

EK 1.1 CImage C++ Sınıfı ve Üye Fonksiyonların Tanımları

Image.h: interface for the Image class.

```
#include "Basic.h"
#include <malloc.h>
#include "PIXEL.h"

#ifndef COLORSPACE_RGB
#define COLORSPACE_RGB          (BYTE)0
Image verisinin renk tabanını RGB olarak belirten IMType seçeneği.
#endif

#ifndef COLORSPACE_GRAY
#define COLORSPACE_GRAY         (BYTE)1
Image verisinin renk tabanını Gri olarak belirten IMType seçeneği.
#endif

#ifndef COLORSPACE_YCbCr
#define COLORSPACE_YCbCr        (BYTE)2
Image verisinin renk tabanını YCbCr olarak IMType seçeneği
#endif

#ifndef COPYAS_BLANKLAYER
#define COPYAS_BLANKLAYER       (BYTE)0
İmage Kopyalama fonksiyonlarının nesne verisiyle eş-boyutlu sıfır katmanı
oluşturmak için kullandıkları seçenek.
#endif

#ifndef COPYAS_IMAGE
#define COPYAS_IMAGE            (BYTE)1
İmage Kopyalama fonksiyonlarının nesne verisini aynen kopya için kullandıkları
secenek.

#endif
#ifndef COPYAS_WHITELAYER
#define COPYAS_WHITELAYER       (BYTE)255
İmage Kopyalama fonksiyonlarının nesne verisiyle eş-boyutlu beyaz katman
oluşturmak için kullandıkları seçenek.
#endif

Mantıksal işlem fonksiyonu (Logic_op) için seçenekler (OPTION)
```

```
#define OP_OR        (BYTE)0
#define OP_PIX_OR (BYTE)4
#define OP_AND     (BYTE)1
#define OP_ADD     (BYTE)2
#define OP_SUB     (BYTE)3


class Image
{
        friend class PIXEL;

public:

        BYTE IMType;
 // COLORSPACE _RGB, _GRAY, _YCbCr
```
İmage nesnesinin renk tabanını belirten değişkendir.

```
        UINT Height;
```
Nesnenin 2-boyutlu yüksekliğini nokta olarak belirtir.

```
        UINT Width;
```
Nesnenin 2-boyutlu genişliğini nokta olarak belirtir.

```
        BYTE *Data;
```
*İmage verisinin nokta başına 24-bit olarak (2-boyutlu görünüme göre) sol-alt köşeden itibaren satırlar halinde 1-boyutlu dizi olarak tutulduğu hafızayı gösterir. Hafıza alanı byte olarak 3*Width*Height kadardır.*

```
//      BYTE *Layer;
```
İmage verisindeki her noktaya(pixel) karşılık gelen katman bilgisinin tutulduğu alanı gösterir.
*Hafıza alanı Width*Height kadardır.*

```
        /*Constructor*/
        Image(BYTE *DataBuffer,             /* Data Buffer */
                BYTE ImageColorType,        /* Image Type */
                UINT ImageHeightInPixels,  /* Rows */
                UINT ImageWidthInPixels );/* Columns */
```
Nesne oluşturucu: Olusturulan image nesnesi icin gerekli hafıza alanını ayırır.Veri göstergesi ile image tipini belirterek olusturabilir.

```
        Image( UINT ImageHeightInPixels,  /* Rows */
                        UINT ImageWidthInPixels );/* Columns */
```
Nesne oluşturucu: girilen yükseklik ve genislikte bir veri alanı ayırır. Ve nesne göstergelerini o alana yöneltir.

```
        Image();
```
Nesne oluşturucu: sıfır nokta değerli image nesnesi oluşturur.

```
virtual ~Image();
```
Nesneyi hafızadan siler.

```
int HiLightRectange(UINT rsx,UINT rsy,
                    UINT rwidth, UINT rheight,
                    UINT OPTION);
```
image verisinde başlangıç noktası (rsx,rsy) ile genişlik ve yüksekliği (rwidth, rheight) verilen dörtgen alanını seçeneğe (OPTION) bağlı olarak belirginleştirir.

```
int HiLightRectange(UINT *Rect, int OPTION);
```
*image verisinde belirtilen dörtgene (*Rect) ve seçeneğe bağlı olarak seçili alanı belirginleştirir.*

```
int QuantizeIntensity(int qlevels);
```
(YCbCr renk tabanında işlem yaparak) Y değerini qlevels kadar seviyeye quantalama yapar.

```
int NormalizeTo(int Nval);
```
(YCbCr renk tabanında işlem yaparak) en yüksek Y değeri Nval olacak şekilde doğrusal bir döüşüm gerçekleştirir.

```
int RemoveDCval();
```
(YCbCr) image verisinde yoğunluk (Y) değerinin dc değerini sıfırlar.

```
int RemoveMean();
```
(YCbCr) ortalama Y değerini sıfırlar.

```
Image * Set2Draw(   double *scale,
                    const int *DrawingZone,
                    int *DrawingRect);
```
İmage verisini Ekran çizim alanına göre yeniden boyutlar ve çizim alanındaki resim dörtgenini belirler. Boyutlanmış veriyi döndürür.

```
static int* FacialRect(Image* myimage);
```
*nesne bağımsız olarak *myimage verisindeki yüz alanını tahmin eder ve dörtgen değerlerini döndürür.*

```
int* FacialRect();
```
bir üstteki fonksiyonun üye sürümü.

```
int MeanOfRect(     POINT *P1,
                    int width, int height);
```
*image verisinde başlangıç noktası (*P1) ile genişlik ve yüksekliği (width, height) verilmiş dörtgensel alanın ortalama Y değerini döndürür.*

```
void dilate( CProgressCtrl *progressbar);
```
image verisini Y değerine göre gri seviye genleştirir.

```
        void erode( CProgressCtrl *progressbar);
```
image verisini Y değerine göre gri seviye darlaştırır.

```
        bool IsValid();
```
image verisinin tutarlı olup olmadığını döndürür.

```
        int Mean();
```
Y değerine göre aritmetik ortalama
```
        int Min(); // non-zero minimum
        int Max(); // non-255 maximum
```

```
        bool FlipHorizontal();
```
image verisini iki boyutlu yatayda ters çevirir.

```
        bool FillBordersFrom(Image* IMAGE, int bordersize);
```
*image verisinin iki boyutlu kenarlarını *IMAGE verisinden bordersize değerine göre kopya eder. (Dilation ve erosion işlemleri için kullanılır)*

```
        bool SetPixel(PIXEL *mpix);
```
*image verisinde *mpix noktasının konumuna göre *mpix renk değerlerini atama yapar.*

```
        bool SetPixel(UINT X, UINT Y, BYTE ValGray);
```
image verisinde (X, Y) konumuna ValGray gri değerini atama yapar.

```
        bool SetPixel( UINT X, UINT Y,
                        BYTE ValB1, BYTE ValB2, BYTE ValB3);
```
image verisinde (X, Y) konumuna 24bit (ValB1, ValB2, ValB3) renk değerlerini atar.

```
        void MoveBuffersFrom(Image* myimage);
```
*image verisini *myimage verisinden kopya eder.*

```
        PIXEL* Averageof9Pixels(CPoint *myp);
```
**myp noktasının komşuluğundaki noktalarla birlikte ortalama renk değerlerini döndürür.*

```
        void Resize(double ratio);
```
image verisini ratio boyutlama oranına göre iki boyutlu yeniden boyutlar.

```
//      void InterpolateUsingLayer();
//      void DrawMode(Image *ofimage);
```

```
        void Resize(UINT ToWidth, UINT ToHeight);
```
image verisini belirtilen genişlik ve yüksekliğe (ToWidth, To Height) yeniden boyutlar.

Image* Resize2(double ratio);
image verisini ratio boyutlama oranına göre iki boyutlu yeniden boyutlanmış olarak döndürür.

Image* Resize2(UINT ToWidth, UINT ToHeight);
image verisini belirtilen genişlik ve yüksekliğe (ToWidth, To Height) yeniden boyutlanmış olarak döndürür.

static float** MatricalArrayF(int M, int N);
image nesnesinden bağımsız olarak hafızada (M,N) boyutlu float tipinde iki boyutlu alan ayırır ve gösterge değerini döndürür.

static int** MatricalArrayI(int M, int N);
image nesnesinden bağımsız olarak hafızada (M,N) boyutlu int tipinde iki boyutlu alan ayırır ve gösterge değerini döndürür.

static void DeleteMatricalArray(float **pMASK, int M);
image nesnesinden bağımsız olarak hafızada ayrılmış pMASK göstergeli, (M,-) iki boyutlu float tipinde alanı serbest bırakır.

static void DeleteMatricalArray(int **pMASK, int M);
image nesnesinden bağımsız olarak hafızada ayrılmış pMASK göstergeli, (M,-) iki boyutlu int tipinde alanı serbest bırakır.

void Destroy(bool BuffersOnly);
image verisinin tuttuğu hafizayı serbest bırakır, yani siler :) sadece veriyi silme seçeneği var

static BYTE* Get(bool DATAis1_LAYERis0, Image *ofimage);
*image verisinden bağımsız olarak *ofimage verisinin seçeneğe bağlı olarak renk verisini veya katman verisini ham (BYTE dizisi) olarak döndürür.*

Image& operator =(Image &img);
Kopyalama işletmeni

bool GetPixelVals(PIXEL *Pix);
*image verisinden *Pix'in konum bilgisini kullanarak renk değerlerini *Pix'e atama yapar.*

```
/*      static float MASKo[5][5];
        static float MASK[5][5];
        static float MASK1f[5][5];
        static int MASK1i[5][5];*/
```

static void RotateImage(Image *myimage, double angle);
**myimage verisini warp işlemi kullanarak iki boyutlu olarak angle radyal açısıyla döndürür.*

```
        void ResizeImageBuffer(int *Wx, int *Wy);
```
image verisinde warp işlemi uygulanırken yeni boyutlara göre veri için tutulan hafıza büyüklüğünün yeniden boyutladırılmasını gerçekleştirir

```
        void AlingEyes(CPoint *RE, CPoint *LE);
```
sağ göz konumunu (RE) ve sol göz konumunu (LE) aynı yatay hizaya getirmek için açısal warp uygulayarak image verisini iki boyutlu döndürür.

```
        void RelocateCornersWithAngle(CPoint *CRP, double q, int *Wx, int
*Wy);
```
*warp işlemi uygulanırken kullanılan, referans noktasını (*CRP) sabit tutarak köşe değerlerini (Wx[4],Wy[4]) yeniden hesaplar.*

```
        bool RGB2Gray();
```
image verisinde RGB renk tabanından gri tabana geri dönüşsüz dönüşüm

```
bool RGB2YCbCr();
```
// color transformation YCbCr <-> RGB , in-place
```
bool YCbCr2RGB();

void InvertColors(BYTE Option);
```
image verisinde renklerin 255'e göre terslerini seçeneğe bağlı olarak alır
```
        void InvertColors();


        Image* DetectFaceArea(CProgressCtrl *progressbar);
```
İmage verisinde yüz alanını tahmin ederek işaretlenmiş yüzü içeren 2-seviyeli yeni veriyi gösterge olarak döndürür.

```
static Image* IsCorrelationOf(Image *Image1, Image *Template1);
```
**Image1 ile *Template1 verilerini iki boyutlu konvolusyon işlemine alır ve korelasyon değerlerini yeni image olarak döndürür.*

```
static Image* IsCorrelationOf(Image *Image1,            //sg: input image
                              UINT *Area2searchInImage1,
                              Image *Template1);        //sg: template image
```
**Image1 verisinde belirtilen 2-boyutlu (Area2searchInImage1[8]) alana sınırlı olarak bu alanı *Template1 verisiyle 2-boyutlu konvolusyon işlemine alır ve korelasyon değerlerini yeni image olarak döndürür.*

```
static POINT* IsCorrelation2Of(     int threshold,
                                    Image *Image1,
                                    UINT *Area2searchInImage1,
                                    Image *Template1,
                                    CProgressCtrl *progressbar);
```
**Image1 verisinde belirtilen 2-boyutlu (Area2searchInImage1[8]) alana sınırlı olarak bu alanı *Template1 verisiyle 2-boyutlu konvolusyon işlemine alır ve*

korelasyon değerlerini eşik değerinden(threshold) süzerek yeni image olarak döndürür. Progressbar ilerletilir.

```
static void Logic_Op( const Image *Image1,//sg: input image 1
                      const Image *Image2,//sg: input image 2
                      Image *OutImage,        //sg: output image
                      BYTE Operation);        //sg: determines which LOGIC
                                                  operation will be
                                              applied
```

*image verisinde mantıksal işlemler yapmak icin bir fonksiyon. *Image1 ile *Image2 verilerini Operation seceneğine göre AND, OR, gibi işlemlere sokar ve sonuc *OutImage olarak döner.*

```
Image* Eroded(bool inRGB, CProgressCtrl *progressbar);
```

İmage verisini inRGB secenegine gore renkli veya gri-taban aşındırır. Progressbar ilerletilir. Ve sonuç image verisi gösterge olarak döndürülür.

```
Image* Dilated(bool inRGB, CProgressCtrl *progressbar);
```

İmage verisini inRGB secenegine gore renkli veya gri-taban genleştirir. Progressbar ilerletilir. Ve sonuç image verisi gösterge olarak döndürülür.

```
Image* DilationColor(float **MASK, int N, CProgressCtrl *progressbar);
```

*İmage verisini **MASK 2-boyutlu filtresi ile süzerek (YCbCr tabanında Y kullanılarak) genleştirir. Progressbar ilerletilir. Ve sonuç image verisi gösterge olarak döndürülür.*

```
static void DilationGray(     const Image *IMAGE,
                              float **MASK, int N,
                              Image *DILATED);
```

*nesneden bağımsız olarak IMAGE verisini **MASK 2-boyutlu filtresi ile süzerek (YCbCr tabanında Y kullanılarak) genleştirir. Progressbar ilerletilir. Ve sonuç verisi DILATED göstergesiyle döndürülür.*

```
Image* ErosionColor(float **MASK, int N, CProgressCtrl *progressbar);
```

*İmage verisini **MASK 2-boyutlu filtresi ile süzerek (YCbCr tabanında Y kullanılarak) aşındırır. Progressbar ilerletilir. Ve sonuç image verisi gösterge olarak döndürülür.*

```
static void ErosionGray(      const Image *IMAGE,
                              float **MASK, int N,
                              Image *ERODED);
```

*nesneden bağımsız olarak IMAGE verisini **MASK 2-boyutlu filtresi ile süzerek (YCbCr tabanında Y kullanılarak) aşındırır. Progressbar ilerletilir. Ve sonuç verisi ERODED göstergesiyle döndürülür.*

```
void PutPlus(CPoint *Pxy);
```

*image verisinde *Pxy noktası ile belirtilen konum artı işareti olacak şekilde değiştirilir.*

Image* GetRect(CPoint *P1, int width, int height);
*İmage verisinde *P1 başlangıç noktası ile genişlik ve yüksekliği belirtilen 2-boyutlu veri yeni image nesnesi olarak gösterge biçimiyle döndürülür.*

Image* GetRect(CPoint *P1, CPoint *P2);
*İmage verisinde sol üst köşesi *P1 noktası ve sağ alt köşesi *P2 noktası olan 2-boyutlu veri yeni image nesnesi olarak gösterge ile döndürülür.*

static void Copy(const Image *Res, Image *Target, BYTE GenLayer);
*nesneden bağımsız olarak kaynak (*Res) verisi hedefe (*Target) kopyalanır. GenLayer seçeneği pixel renk verisinin kopyalanıp kopyalanmayacağını belirtir. Katman olarak eş boyutlu fakat boş bir nesne oluşturmak için GenLayer=1 olarak girilir.*

```
BYTE* Filter2D(float *MASK);
        ////////////////////////////////////////////////////////////
        // Warp an image (Complex rotation)
        // BASIC WARP ALGORITHM
        // Using bilinear Interpolation

void Warp(      Image *Out,
                int *Wx, int *Wy,
                bool RotationIs1_ResizingIs0,
                CProgressCtrl *progressbar);
/*      int *Wx,            // output coordinate pairsX
                            // as pointers to two integer arrays
        int *Wy);           // output coordinate pairsY
                            // as pointers to two integer arrays */

void Warp(      Image *Out,
                CPoint *SrcPS, CPoint *SrcPE,
                int *Wx, int *Wy,
                bool RotationIs1_ResizingIs0,
                CProgressCtrl *progressbar);
/*      int *Wx,            // output coordinate pairsX
                            // as pointers to two integer arrays
        int *Wy);           // output coordinate pairsY
                            // as pointers to two integer arrays
*/
```

bool SaveJPGtoFile(CString FileName, BOOL Color);
image verisini girilen dosya adıyla (FileName) JpegLib kütüphanesini kullanarak jpeg formatında kaydeder. Dosya adı dosya yolunu (path) da içerir. Color değişkeni ise JpegLib kütüphanesi tarafından RGB veya Gri-seviye secimi icin kullanılır.

bool LoadJPGfromFile(CString FileName);
Dosya yoluyla beraber adı girilen dosyayı JpegLib kütüphanesi ile jpeg formatında okuyup image nesne verisi olarak kaydeder.

};

EK 1.2 CPIXEL C++ Sınıfı ve Üye Fonksiyonların Tanımları

PIXEL.h: interface for the PIXEL class.

```
#include "../base/Image.h"

#define Pixel PIXEL

class PIXEL
{
        friend class Image;
```

Image sınıfıyla birlikte çalışır.

```
public:

        UINT x;
```

PIXEL nesnesinin yatay koordinat konum bilgisi

```
        UINT y;
```

PIXEL nesnesinin dikey koordinat konum bilgisi

```
        BYTE ValB1;
```

PIXEL nesnesinin birinci renk bileşeni

```
        BYTE ValB2;
```

PIXEL nesnesinin ikinci renk bileşeni

```
        BYTE ValB3;
```

PIXEL nesnesinin üçüncü renk bileşeni

```
        BYTE type;
```

PIXEL nesnesinin renk tabanı-biçimi-formatı-tipi (GRAY, RGB, YCbCr)

```
        PIXEL();
```

Boş (sıfır) Nesne oluşurucu

```
        PIXEL(UINT X, UINT Y);
```

(X, Y) koordinatlı nesne oluşturucu

```
        PIXEL(UINT X, UINT Y, BYTE VALB1);
```

Nesne oluşturucu (gri-seviye)

```
        PIXEL(UINT X, UINT Y, BYTE VALB1, BYTE VALB2, BYTE
VALB3);
```

Tam verili nesne oluşturucu

virtual ~PIXEL();
nesneyi hafızadan siler.

void YCbCr2RGB();
PIXEL nesnesinin renk biçimini RGB'ye dönüştürür.

void RGB2YCbCr();
PIXEL nesnesinin renk biçimini YCbCr'a dönüştürür.

CString ToString();
PIXEL nesnesinin nokta koordinatlarını "(XXX, YYY)" metni şeklinde döndürür.
static CString ToString(CPoint point);
Nesneden bağımsız olarak CPoint nokta koordinatlarını "(XXX, YYY)" metni şeklinde döndürür.

void GetColorV(COLORREF crColor);
32-bit formatında verilmiş olan (crColor) RGB renk değerlerini ayırır ve PIXEL nesnesine kaydeder.

COLORREF SetColorV();
PIXEL nesnesinin renk değerlerini RGB formatında birleştirip 32-bit COLORREF formatında döndürür.

bool GetPixelV(CDC *dc, int X, int Y);
**dc ekran nesnesindeki (X, Y) koordinatlı noktanın renk değerlerini ve konumunu PIXEL nesnesine kaydeder ve işlem hatasız ise 1 döndürür.*

bool SetPixelV(CDC *dc);
*PIXEL nesnesinin konum bilgisine göre *dc ekran nesnesindeki ilgili koordinattaki renk değerlerini değiştirir. Hatasız işlem sonucunda 1 döndürür.*

bool IsValidFor(const Image* image);
*PIXEL nesnesinin konum bilgisinin *image nesnesinde tanımlı olup olmadığını döndürür. (nokta koordinatları resim sınırları dışındaysa 0 döndürür)*

PIXEL& operator =(CPoint &point);
CPoint (point) nesnesinin konum bilgisini PIXEL nesnesine kopyalar.

bool SetPixelVals(Image *ofimage);
**ofimage nesnesindeki ilgili (PIXEL nesnesinin konum bilgisine uyan) nokta renk değerlerini değiştirir.*

bool GetPixelVals(const Image *fromimage);
**fromimage nesnesindeki ilgili (PIXEL nesnesinin konum bilgisine uyan) nokta renk değerlerini PIXEL nesnesine kopyalar.*

bool GetAsAverageofNearest9Pixels(const Image *fromimage);

*nesne koordinatlarına uyan *fromimage resmindeki noktanın komşuluğundaki 8 noktayla beraber ortalamalarını PIXEL nesnesine alır. Hatasız işlem 1 döndürür.*

};
)

EK 1.3 MFC Arayüz Sınıfları ve Üye Fonksiyonların Tanımları

// Microsoft Windows icin gelistirilmekte olan MFC tabanlı arayüzün yazılım kodları
Bu metodlar/fonksiyonlar diğer class'larda geliştirilmiş olan metodların birleştirilerek uygulandığı karma metodlar içeren gelişmiş fonksiyonlar olarak tanımlanabilir.

YTASv3Dlg.h : header file

```
#include "MyMenu.h"
#include "Controls.h"
#include "EkleWindow.h"
#include "CoderWindow.h"

class CYTASv3DlgAutoProxy;

/////////////////////////////////////////////////////////////////////////////
// CYTASv3Dlg dialog

class CYTASv3Dlg : public CDialog
{
	DECLARE_DYNAMIC(CYTASv3Dlg);
	friend class CYTASv3DlgAutoProxy;

// Construction
public:

	CYTASv3Dlg(CWnd* pParent = NULL);	// standard constructor
	virtual ~CYTASv3Dlg(); // standard destructor
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
protected:
	CYTASv3DlgAutoProxy* m_pAutoProxy;
	HICON m_hIcon;
	HWND m_hmenu;
	BOOL CanExit();
	virtual BOOL OnInitDialog();
		Başlangıç tanımlama fonksiyonu
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnClose();
		Çıkış
```

```
        virtual void OnOK();
        virtual void OnCancel();
        afx_msg void OnButton1();
                Ekle Penceresini çağıran fonksiyon

        afx_msg void OnMenu3Sag();
                Sağ-tık menusunde sağ göz işaretleme işlemini yapan modül

        afx_msg void OnMenu3Sol();
                Sağ-tık menusunde sol göz işaretleme işlemini yapan modül

        afx_msg void OnMenu3Gozhiza();
                Sağ-tık menusunde göz hizalama işlemini yapan modül

        afx_msg void OnBtoplu();
                Toplu Arşiv Penceresinin çağırılarak ekranda gösterilmesini
sağlayan fnk.
private:
        CMyMenu *pMyMenu;
                Sağ-tık menusune gösterge
public:
        int namecount;
                işlenen dosya sayısını tutmak için değişken

        CEkleWindow EkleDlg;
                EkleWindow nesnesi,

        CCoderWindow CoderDlg;
                TopluWindow nesnesi
        int first;
                EkleWindow için gerekli bir değişken
}
```

EkleWindow.h : header file

```
#include "CImage.h"
/////////////////////////////////////////////////////////////////////////////
// CEkleWindow dialog

class CEkleWindow : public CDialog
{
// Construction
public:

        CEkleWindow(CWnd* pParent = NULL);   // standard constructor

        CListBox        m_cfilelist;
```

Dosya listesini tutan nesne

CButton m_ccheckgh;
Göz hiza kontrolü

CButton m_ccheckmr;
MeanRemove kontrolü

CButton m_ccheckhe;
Histogram Eq. kontrolü
CStaticm_ccurrentrectarea;
Büyüteç alanı

CEdit m_board;
Son bilgilerin ekrana yazdırıldığı kutucuk

CStaticm_cSampleDraw;
Örnek çizim alanı

CStaticm_cDrawArea;
Görüntü çizim alanı

CString m_boardtext;
Bilgi metni nesnesi

CString m_statusds;
Görüntü oranını belirtir label

CString m_statusxy;
Yatay dikey konumu belirtir label

BOOL m_bcheckedhe;
Histogram eşitleme bayrağı (flag)

BOOL m_bcheckmr;
Ortalama değer silme bayrağı

BOOL m_bcheckgh;
Göz hiza bayrağı

public:

void callnext();
Bir sonraki görüntüyü getiren fonksiyonu dışarıdan (class dışı) çağırmak için

void callcut();
Bir görüntüyü parçalayan fonksiyonu dışarıdan (class dışı) çağırmak için

CString filename;
Açık dosya adı

CString filepath;
Açık olan klasor yolu

bool m_bgozhiza;
goz hizalama ile ilgili bayrak

int currenty;
fare imleci konum belirteci

int currentx;
fare imleci konum belirteci

CRect currentrectarea;
Büyüteç alanı dörtgen bilgisi

void calcdrimagearea();
çizim alanındaki görüntü boyutlarını hesaplar

CRect drimagearea;
Çizim alanını köşe noktaları olarak tutar

double calcdispscale(CRect *DR, CImage *myimage);
çizim alanı için görüntünün boyutlandırılmasını sağlar

void CalcDrawRect(CRect *DR);
çizim alanını hesaplar

double dispscale;
ekran oranlama ayarını tutar

void ResetAll();
menuler dahil, bütün değişkenleri sıfırlar

void ResetMenus();
menulerin kilitlerini kaldırır

void UpdateImageDisplay();
görüntüyü yenileme işlemi

CMenu *popupmenu;
Sağ-tık menu göstergesi

CMenu rightclickmenu;
Sağ-tık menusu

```
        CImage displayedimage;
                Görüntülenen CImage nesnesi

        CDialog* pdialog;
                this göstergesi

        CPoint ClickedPoints[2];
                Sağ-tık ile işaretlenen göz konumlarının koordinatlarını tutar

        void myfunc(CImage *image, CPoint *point);
                göz beyazının tespitinin denenmesi için yazılmış bir fonk.

        CRect m_cSampleDrawRect;
                Örnek görüntü alanının köşe noktaları

        CImage m_loadedimage;
                Yüklenen orjinal CImage verisi

        CRect m_cDrawAreaRect
                Çizim alanının köşeleri

        int namecount;
                dosya sayacı
protected:
        afx_msg void OnBrowse();
                klasor ve dosya seçimini sağlayan Yükle butonuna fnks.

        afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
                sağ-tık menusu

        virtual void OnCancel();
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
                fare imlecinin hareketine tepki veren fonk.

        afx_msg void OnMove(int x, int y);
                pencere taşıma işlemine tepki veren fonk.

        afx_msg void OnPaint();
        virtual BOOL OnInitDialog();
        afx_msg void OnButtonCorr1();
                konvolusyon dugmesi kontrolu

        afx_msg void OnBhist();
                histogram eşitleme butonunu işletir

        afx_msg void OnBkayd();
                dosyayı kaydetme butonuna map
```

```
        afx_msg void OnBrenk();
                renk değiştirme butonuna...

        afx_msg void OnBmask1();
                netleştirme butonuna...

        afx_msg void OnBdilate();
                dilate butonuna...

        afx_msg void OnBerode();
                erode butonuna...

        afx_msg void OnCheckhe();
                histogram eşitleme kontrolu

        afx_msg void OnCheckmr();
                ortalama değer kontrolu

        afx_msg void OnBmask();
                maskeleme kontrolu

        afx_msg void OnBcut();
                goruntuyu parçalara ayırıp kaydetme kontrolu

        afx_msg void OnBbirsonra();
                klasordeki bir sonraki dosya acilir ve goruntulenir

        afx_msg void OnBbironce();
                klasordeki bir önceki dosya acilir ve goruntulenir

        afx_msg void OnCheckgh();
                göz hizalama kontrolu
};
```

CoderWindow.h : header file

```
#include "CImage.h"
/////////////////////////////////////////////////////////////////////////////
// CCoderWindow dialog

class CCoderWindow : public CDialog
{
// Construction
public:
        int m_ifilecount;
                dosya sayacı
```

```
        CListBox        m_cfilelist;
                Dosya liste nesnesi

        CComboBox   m_ccombotypes;
                Combo kontrolu

        CProgressCtrl m_cprogressbar1;
                İlerleme çubugu kontrolu

        int             m_icodewordnum;
                kod-vektor sayısını belirler

        CString         m_sfilecount;
                Dosya sayısı metni

        CString         m_scombotypes;
                Combo metni

        CString         m_sfilepath;
                Dosya klasor yolu

        CString         m_storepath;
                Kod-kitap kayıt yolu
protected:
        afx_msg void OnBbasla();
                kitap tasarimina baslamak için buton...

        afx_msg void OnBcodewndown();
                kod-vektor sayısını azatlan buton

        afx_msg void OnBcodewnup();
                kod-vektor sayısını azatlan buton

        virtual void OnCancel();
                çıkış

        afx_msg void OnSpath();
                klasor yolunu değiştirmek için...

        afx_msg void OnBbrowse();
                klasor yolunu değiştiren buton

        virtual BOOL OnInitDialog();
        afx_msg void OnBopenlog();
                kayıt defterinin görüntülemek için...
};
```

EK 1.4 CVector C++ Sınıfı ve Üye Fonksiyonların Tanımları

Vector.h: interface for the CVector class.

```
//////////////////////////////////////////////////////////////////////
#include "VQ.h"

class CVector
{
        friend class CVQ;
public:
        int loadfromimagefile(CString filename);
                filename ile gosterilen dosya bilgisini dbuffer'a yukler

        void free_dbuffer();
                dbuffer göstergesini serbest bırakır

        int loadimage2dbuffer();
                dosya bilgisine gore CImage bilgisini double buffer'a yukler

        int reallocateDbuffer(int rewidth, int reheight);
                dbuffer alanını degistirir

        void saveDbufferAsImage(CString filename, bool withmean);
                dbuffer verisini jpg olarak kayit yapar

        CString data2string();
                Veri bilgisini metne donusturur

        void reset();
                bilgileri sıfırlar

        void disloadimagedata();
                CImage alt nesnesini siler

        void removemean(bool updateold);
                ortalama değeri temizler ve imagemean değerine atar

        int height;
                goruntu vektorunun piksel yuksekliği

        int width;
                goruntu vektorunun piksel genişliği

        CString file;
                Dosya yolu
```

```
        int region;
                voronoi bolge numarası

        double distance;
                centroid'e uzaklık

        int imagemean;
                ortalama değer

        double *dbuffer;
                double buffer tampon göstergesi

        CImage *image; // should be grayintensity and mean removed
        CVector();
        virtual ~CVector();
};
```

EK 1.5 CVQ C++ Sınıfı ve Üye Fonksiyonların Tanımları

VQ.h: interface for the CVQ class.

```
//////////////////////////////////////////////////////////////////////
#include "CImage.h"  // Added by ClassView
#include "Vector.h"

class CVQ
{
        friend class CVector;
public:

static void calcregions(CVector *vectors, int numofvecs,CVector *centroids, int
numofcents);
        vektorler dizisini kullanarak istenen sayıda merkez hesaplar ve vektorlerin
bu      merkezlere uzaklıklarını ve bölgelerini de vektör bilgilerine kaydeder

static void calccentroid(CVector *vectors, int numberofvectors, CVector
*rcentroid);
        girilen sayıda vektorun ortalama değer vektörünü hesaplar

static double calcdistance(CVector *vector1, CVector *vector2);
        iki vektor arası uzaklığı hesaplar

        CVQ();
        virtual ~CVQ();
};
```