T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

AÇIK KAYNAK KODLU GÖRÜNTÜ İŞLEME UYGULAMALARI

Fuat ESMERAY

Yüksek Lisans Tezi
Anabilim Dalı: Elektronik ve Bilgisayar Eğitimi
Danışman: Doç. Dr. Abdulkadir ŞENGÜR

ŞUBAT- 2014

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

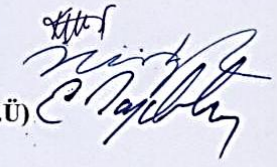
AÇIK KAYNAK KODLU GÖRÜNTÜ İŞLEME UYGULAMALARI

YÜKSEK LİSANS TEZİ

Fuat ESMERAY
101131103

Tezin Enstitüye Verildiği Tarih: 26 Aralık 2013
Tezin Savunulduğu Tarih: 10 Ocak 2014

Tez Danışmanı : Doç. Dr. Abdulkadir ŞENGÜR (F.Ü)
Diğer Jüri Üyeleri : Doç. Dr. İbrahim TÜRKOĞLU (F.Ü)
Yrd. Doç. Dr. Erkan TANYILDIZI (F.Ü)



ŞUBAT–2014

ÖNSÖZ

Bu çalışmada OpenCV kütüphaneleri, Studio.Net platformu dillerinden C#, EmguCV aracılığıyla birleştirilerek kompleks bir görüntü işleme teknolojisi oluşturulmuştur. Optimize OpenCV fonksiyonları, Studio.Net platformunun sistem kaynaklarını etkin kullanımı ve EmguCV ile birlikte yardımcı görüntü işleme araçlarının sunulduğu bu tek platformda temel görüntü işleme uygulamaları geliştirilmiştir.

Yüksek lisans çalışma süresince yaşadığım sıkıntılı süreçlerde gösterdiği anlayış ve tez çalışmamda bilgisiyle yolumu aydınlatan değerli hocam Doç. Dr. Abdulkadir Şengür'e ve çalışmalarımın her aşamasında destekleriyle beni cesaretlendiren ve motive eden çalışma arkadaşlarım Canser Kardaş ve Bilal Altan'a sonsuz teşekkürlerimle.

Fuat ESMERAY

ELAZIĞ-2014

İÇİNDEKİLER

Sayfa No

ÖZET

Sayısal görüntülerin giriş olarak alınıp amaca uygun işlemlerden geçirilerek elde edilen veriler üzerinden yeni kararlar oluşturması görüntü işleme olarak tanımlanır. Görü sistemleri görüntünün dış ortamdan algılanıp veya sistemin kendi içerisinde oluşturup hedef sonuçlara ulaşılıncaya kadar bütün süreçleri yöneten yazılımsal ve donanımsal yapıların bütününü ifade eder.

Sağlık, coğrafya, jeolojik, güvenlik, imalat sanayi gibi uygulama alanları gün geçtikçe artan görüntü işleme teknolojilerinin işlem kapasitesi oldukça fazladır. Görüntü işleme teknolojileri bu işlem kapasitesini zaman kaybını minimum seviyede tutarak yapan sistemleri geliştirmeyi amaç edinmiştir. Günümüzde bu amaçla geliştirilen donanımsal teknolojilere ek olarak, bilgisayarları veya diğer elektronik araçların görü sistemleri gibi davranmasını sağlayan çeşitli yazılımsal kütüphaneler ve programlama dilleri geliştirilmiştir.

Sunulan bu tezde EmguCV sarmalayıcı kütüphaneleri ile OpenCV fonksiyonlarının Stduio.Net platformunda C# dilinde temel görüntü işleme uygulamalarını geliştirmeyi amaçlamıştır. EmguCV aracılığıyla C# ortamında geliştirilen uygulamalar, optimize edilmiş OpenCV fonksiyonları ile C# dilinin güçlü yönlerini birleştirir. EmguCV sarmalayıcı yapısı sayesinde sistem kaynaklarının daha etkili kullanımına ek olarak görüntü verilerinin optimize edilmiş OpenCV fonksiyonları sayesinde daha hızlı işlenmesi ve daha etkili sonuçlar vermesi sağlanmıştır.

Sonuç bölümünde; EmguCV'li yazılım ortamıyla geliştirlen uygulamalar, görüntü işleme de güçlü fonksiyonlara sahip Matlab'taki aynı uygulamalar ile aynı veri grubu için hız ve performans açısından karşılaştırma yapılmıştır. OpenCV fonksiyonları ile geliştirlen uygulamaların aynı işlemi yapan birçok Matlab uygulamasından hızlı sonuçlar ürettiği ve performanslarınında daha iyi olduğu tespit edilmiştir.

Anahtar Kelimeler: Görüntü işleme, OpenCV, EmguCV

SUMMARY

Open Source Image Processing Applications

Image processing is defined as to create new decisions on the datas which obtained by being taken digital images as input and treating it in the manner suitable for the purpose. The visual systems is meaning that all hardware and software structures which managing the whole processes by detecting from the external environment of the image or by creating in the system until it reaches target results

The processing capacity of image processing technology which its application areas like health, geography, geology, security, production industry increasing day by day is pretty much. The image processing technologies aims to develop the systems that making this processing capacity by keeping the loss of time at a minimum level. Today in addition to hardware technologies that developed for this purpose, various software libraries and programming languages have been developed that making computers or other electronic tools behave like visual systems.

In this thesis it has been aimed to develop in C# language at Studio.Net platform the basic image applications of EmguCV wrapper library and OpenCV functions. The applications that being developed in the C# *environment* through EmguCV, are assembling the functions of OpenCV that being optimized and the strenghts of C# language. In addition to more effective using of system recources thanks to the wrapper structure of EmguCV, faster processing of image data and given more effective results has been provided thanks to OpenCV functions that being optimized

In the conclusion section, a comparison was made in terms of speed and performance between the applications developed with EmguCV software environment, the same applications which having powerful functions of image processing in Matlab and the same data group. It was determined that the applications developed with OpenCV functions has produced faster results than many Matlab applications engaged in the same process and has been better than their performance.

Key Words: Image Processing, OpenCV, EmguCV

ŞEKİL LİSTESİ

Sayfa No

KISALTMALAR LİSTESİ

Bi-i	: Biologically Inspired Cellular Vision System (Biyolojik Hücresel Görü Sistemi)
CIE L*a*b	: Lightness - ±a: Red-Green - ±b: Yellow-Blue (Parlaklık- Kırmızı Yeşil, Sarı Mavi renk uzayı)
CIE L*u*v	: Lightness - ±a: Red-Green - ±b: Yellow-Blue (Parlaklık- Kırmızı Yeşil, Sarı Mavi renk uzayı)
CV	: Computer Vision (Bilgisayarlı Görü)
CIE XYZ	: X: Kırmızı, Y: Mavi, Z: Sarı renk uzayı
CUDA	: Compute Unified Device Architecture (Birleşik Hesaplama Mimarisi)
CMYK	: Cyan Magenta Yellow Blac**k** (Turkuaz Eflatun Sarı Siyah renk uzayı)
IPP	: Intel Performance Primitives (Intel Performans Mimarileri)
LTI	: Luminance Transient Improvement (Işık Geçiş İyileştirme)
RGB	: Red Green Blue (Kırmızı Yeşil Mavi renk uzayı)
HSV	: Hue Saturation Value (Renk Saturasyon Değer renk uzayı)
HSL	: Hue Saturation Luminance (Renk Saturasyon Aydınlık renk uzayı)
MLL	: Machine Learning Library (Makine Öğrenme Kütüphanesi)
OCR	: Otomatic Character Recognize (Otomatik Karakter Tanıma)
OpenCV	: Open Source Computer Vision Library (Açık Kaynak Kodlu Bilgisayarlı Görü Kütüphanesi)
VGL	: Vision Geometry Library (Geometrik Görü Kütüphanesi)
VNL	: Vision Numerics Library (Rakamsal Görü Kütüphanesi)
VIL	: Vision Image Library (Resim İşleme Görü Kütüphanesi)
VSL	: Vision Streaming Library (Akış Görü Kütüphanesi)

SEMBOLLER LİSTESİ

a	: L*a*b renk uzayında kırmızı yeşil eksen değişkeni
b	: L*a*b renk uzayında sari mavi eksen değişkeni
Cb	: Mavi renk kanalı krominansı
Cr	: Kırmızı renk kanalı krominansı
f(x,y)	: Giriş görüntü fonksiyonu
g(x,y)	: Çıkış görüntü fonksiyonu
L	: Luminance (aydınlatma) değişkeni
L	: L*a*b renk uzayında parlaklık değişkeni
H	: Hue (renk) değişkeni
S	: Saturation (yoğunluk) değişkeni
u	: L*u*v renk uzayında kırmızı yeşil eksen değişkeni
v	: L*u*v renk uzayında sarı mavi eksen değişkeni
V	: Value (değer) RGB renk kanallarında en büyük yâda en küçük piksel değer değişkeni
Y	: y türev sırası
X	: x türev sırası
$N\ x,y$	: Maksimum olmayan noktaları bastırılmış görüntü fonksiyonu
$s\ x,y$	: Gauss maskesi ile bulanıklaşan görüntü fonksiyonu
$I\ x,y$	: Gauss yumuşatıcı filtre
ζ(x,y)	: Konvülasyon matrisi
∇f	: Gradient kenar büyüklüğü
θ x, y	: Kenar oluşum yönleri
$\nabla^2 f$	: Giriş fonksiyonunun ikinci türevi

1. GİRİŞ

Teknolojinin gelişim sürecine paralel olarak bilgisayarların artan işlem hacimlerinin, hız ve zaman değişkenlerinde sağladığı ters orantı bilgisayar ile yapılacak işlem yelpazesini oldukça genişletmiştir. Bu kapsamda bilgisayarlar günümüzde, savunma sistemlerinde, medikal veri analizinde, haberleşmede, trafik kontrolünde, eğitim araştırmalarında, sosyal ve ekonomik araştırmalarda, veri bankalarının oluşturulması ve işlenmesi gibi daha birçok alanda tercih edilen araçlardır. Bilgisayarlar, tercih alanlarına ek olarak, artan işlem kapasiteleri ve destekledikleri yapay zekâ teknolojileri ile orantılı olarak günümüzde görme ve gördüğünü yorumlama gibi insansı davranışlar sergileyebilmektedirler. Bilgisayarların taklit edebildikleri bu türden davranışların başarısına bağlı olarak, hızlı veri işlemenin gerektiği trafik kontrol noktaları, medikal görüntü analizleri, harita yorumlama gibi veri işleme yönteminin çok farklı olduğu uygulama alanlarında tercih edilmelerini sağlamıştır. Gelişen teknoloji ile paralel olarak bilgisayar ile çözülmek istenen sorunun niteliğine uygun çözüm yöntemleri ve teknolojileri de artarak çeşitlilik göstermektedir. Bu kapsamda doğal ortamdan görüntü verilerini dijitalize eden, veriler üzerinde işlemler yapıp bu verilerden anlamlı sonuçlar çıkaran görüntü işleme teknolojileri de gün geçtikçe artış göstermektedir.

Görüntü işleme teknolojilerini günümüzde donanımsal seviyede görüntüyü işleyen teknolojiler ve yazılımsal görüntü işleyen teknolojiler olarak ayırmak mümkündür. Son zamanlarda, paralel çalışan iki işlemci ile görüntü verilerini sinyal seviyesinde işleyen Bi-i, Eye-Ris görü sistemleri ile Nvidia Cuda donanımsal teknolojilere örnek oluşturuyor iken, bilgisayarların birer görü sistemi gibi işlevsel olmasını sağlayan DirectX, OpenGL, OpenCV, EmguCV gibi birçok kütüphane yazılımsal görüntü işleme teknolojileri olarak sınıflandırmak mümkündür. Görüntü işleme teknolojileri, görüntü verilerinin çokluğundan dolayı temel olarak görüntü verilerini olabilecek minimum parçalara bölüp eş zamanlı işlemeyi hedef edinir.

Bu tez kapsamında EmguCV aracılığıyla görüntü işlemede yeni bir teknoloji olan OpenCV kütüphaneleri ile Studio.Net platformunun güçlü yönlerini C# dili ile birleştirilip OpenCV'nin optimize edilmiş görüntü işleme fonksiyonları ile görüntüler üzerinde temel ön operasyon uygulamaları geliştirilmiştir.

1.1. Görüntü İşleme

Günümüzde bilişim teknolojilerinin, işlem kapasiteleriyle hayata daha çok girmesi ve bununla birlikte daha çok alternatif sunmaları her gün bilgisayar ve bilişim teknolojileriyle insanoğlunun yapmak istediklerine yeni bir boyut kazandırmıştır. Bu kapsamda bilişim teknolojileri ile gerçek hayattan görüntüleri algılayıp bu görüntüler üzerinde yapılan çeşitli işlemler sonucunda görüntülerden anlamlar çıkarma, bilişim teknolojileri gelişim tarihiyle paralel bir çalışma alanı olmuştur.

Görüntülerden, dijital ortamda istenilen anlamları elde etmek için ham görüntü üzerinde çeşitli işlemler yapmak gerekir. Bu kapsamda görüntü işlemenin çerçevesi görüntünün parlaklığını değiştirme, karşıtlığını değiştirme, filtreleme gibi görüntü verilerini iyileştirmekten, işlemlerle görüntüyü anlamlandırma ve temel hedef olarak görüntü içerisindeki istenilen nesne elde edilene kadar geniş bir işlem durum yelpazesini ifade eder. Görüntüler üzerinde yapılacak işlemlerin seçimi tamamen görüntünün karakteristiğine ve ulaşılmak istenilen sonuca göre her bir görüntü için farklılık gösterebilir. Örneğin düşük çözünürlüklü renkli bir görüntüde nesne tanımyabilmek için öncelikle görüntünün çözünürlüğünü arttırmak, görüntüyü gri seviyeye dönüştürmek, histogram grafiği çıkarma ve eşikleme işleminden sonra nesne çıkarılabilir. Bu örnekten farklı olarak bir hastanın göğüs grafisini çekmek isteyen bir görsel sistemde farklı donanımlar, farklı çözüm stratejileri ve sonucunda farklı çözüm kararları oluşturulacaktır. Bu iki görsel sistemden anlaşıldığı üzere hemen her görsel sistem temel olarak görüntüleri işliyor olmaları dışında birbirlerinden tamamen farklı olabilirler.

Görüntü işleme (Image Processing); sayısal görüntü verilerini bilgisayar ortamlarında (donanımlar, yazılımlar) amaca uygun şekilde değiştirilmesidir (Kurtulmuş, 2012) şeklinde bir tanım getirmiştir. Bu yaklaşımla görüntü işleme teknolojilerinin ana çerçevesinin yazılımların ve donanımların oluşturduğu tespit edilmiştir.

İnsanın görsel algı sistemi (Vizüel[1] sistem) incelendiğinde görüntülerin dışarıdan alınmasını sağlayan göz organı ve bu organ aracılığıyla alınmış olan görüntülerden istenilen sonuçları çıkaran merkezi sinir sisteminden oluşur. Kabaca görme, insanlarda göze gelen ışığın saydam tabakada ve göz merceğinde kırıldıktan sonra dış ışınların retina bölgesindeki ışığa duyarlı almaçları uyarmasından sonra ışığın retina bölgesinde uyardığı reseptör kadar gözü kontrol eden merkezi sinir sisteminde bir görüntü oluşturulur. Oluşturu-

[1] Organizmaların görmesini sağlayan merkezi sinir sistemi parçasıdır. (URL-1,2013)

lan bu görüntü beynin diğer bölgelerinden gelen hedef sinyalleri doğrultusunda görüntü işlenerek bir görsel algı oluşturulması sağlanır. Oluşan görsel algı merkezi sinir sistemi aracılığıyla başka tepkilerin oluşturulması için kullanılır. Dijital sistemlerde sürekli insanın algılama sistemi taklit edilmiştir. Bu açıdan görsel algı sistemleri incelendiğinde, dışarıdan görüntülerin sistemin içerisine aktarılmasını sağlayan çeşitli yapılarda algılayıcılar ile alınan veriyi işleyen, sistemin yapısına göre yazılımsal ve donanımsal merkezi sistemden oluşmaktadırlar. Görüntü işleme sistemleri yapısal olarak insan görmesini taklit ettiği gibi mantıksal olarak da veriyi aynı şekilde işlemeyi hedeflemiş olsa da başarılı olunamamıştır (Avcı, 2006). Görüntü işleme; insanın görme sisteminin dijital sistemlere aktarılması olarak tanımlandığında, bu durum dijital sistemlerin öğrenmesi, ayrımlar yapması ve görsel girdiler üzerinde eylemler gerçekleştirebilmesi sonucunu gerektirir (Kurtulmuş, 2012).

1.2. Bilgisayarlı Görü

Günümüzde güvenlik önlemlerinin önem kazanması, medikal görüntülerin işlenmesi ihtiyacı, yeryüzü haritalama, insansız hava araçları gibi farklı alanlarda artan veri işleme ihtiyacına bağlı olarak dijital dünya da çok farklı teknolojiler geliştirilmiştir. Temel olarak dışarıdan aldıkları görüntüsel verileri işleyerek yeni kararlar çıkaran bu sistemler çok farklı makinelerde karşımıza çıkabilmektedir. Günümüzde tıp alanında hastalıkların teşhisi için çok farklı görüntü işleme sistemler bulunmaktadır. Bu alanda röntgen cihazlarında x-ışınları hastanın vücudundan geçirilerek kemik yapıları, akciğer, göğüs, sindirim sistemi gibi bölgelere ait görüntüler oluşturularak hastalığın teşhisinde kullanılır. Yine tıp alanında MR (Manyetik Rezonans) teknolojisinde, mıknatıs etkisi ile hareket eden binlerce atoma ait bilgi bir bilgisayara gönderilir ve incelenen alana ait oldukça kaliteli bir resim elde edilir (URL-2,2013). Bunlardan farklı olarak kalbi ultrasonografik görüntüleyen cihaz, yani yüksek frekanslarda ses dalgalarıyla yüzeylerin görüntülerini oluşturan bir başka tıbbi görüntü işleme makinesidir. Bu örneklerden anlaşılacağı üzere görüntü işleme, çok farklı teknolojiler kullanılarak birbirinde çok farklı makinelerde karşımıza çıkabilmektedir. Bilgisayarlı görü (CV: Computer Vision); medikal görüntü analizi, insansız hava araçları, güvenlik sistemleri, uydudan yeryüzü tarama işlemi gibi insanlar tarafından yapılması zor ve oldukça maliyetli olan sistemlerin bilgisayarlı kontrol sistemini ifade eder (Bradski and Kaehler, 2008).

Bilgisayarlı görü; dışarıda bulunan görüntülerin sayısallaştırılarak bilgisayar ortamına alınması ile birlikte bu sayısal veriler üzerinde görüntü iyileştirme, nesne tanıma ve görüntüye ait özniteliklerin belirlenmesi gibi süreçlerin tamamen bilgisayar ortamlarında yapılmasını ifade eder. Bu yaklaşıma göre bilgisayarlı görüde, görüntü işleme; görüntü verisi sayısallaştırılıp bilgisayara aktarıldıktan sonra veri işleme basamaklarının hepsi bilgisayarda, yazılımsal temelli olarak gerçekleştirilir.

Bilgisayarlı görü sistemleri diğer bütün görüntü işleme sistemlerinde olduğu gibi insanın görme yeteneğini taklit etmeyi hedef edinmiştir. Bilgisayarda yerleşik olarak örüntü tanıma, otomatik odaklanma, açıklık (diyafram) kontrol sistemleri bulunmasına rağmen bilgisayarlı görü sistemi insan görü sisteminin halen çok gerisindedir (Bradski and Kaehler, 2008).

Bir bilgisayarlı görü sisteminin başarısı, oluşturulmuş çözüm algoritmalarının başarısının yanında, sisteme bağlı görsel algılayıcılar gibi sistem iskeletini oluşturan tüm donanımsal yapı ve sistemin amacına göre değişkenlik gösterebilir.

1.3. Görüntünün Yapısal İncelenmesi

Görüntü işleme, yaşam var oldukça hep söz konusu olmuştur (Bayram, 2013). Görme sistemine sahip canlıların hepsi görüntü sinyallerinin uzay zamanda bulunan analog temelli görüntüleri sürekli işlemişlerdir. Son yüz yılda teknolojide ki gelişmelerle birlikte analog görüntünün dijital ortamlarda ifade edilmesi sorunu sonucunda gerçek hayattan görüntüler çeşitli teknolojiler vasıtasıyla sayısal olarak elektronik ortamlarda oluşturulma imkânı kazanmıştır.

1.3.1. Analog Görüntü

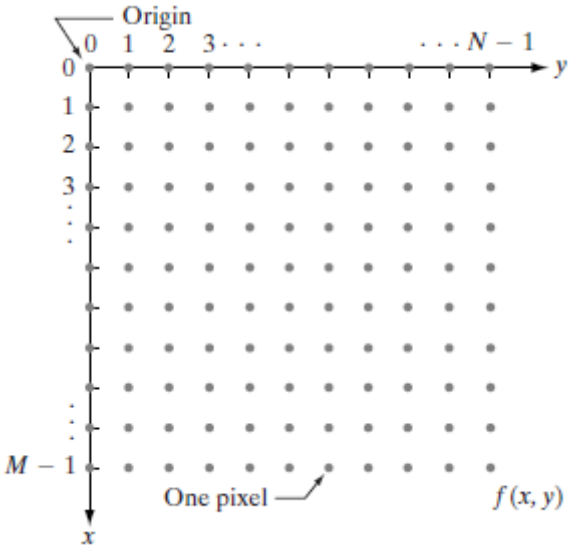
Analog (Sürekli) görüntü; etrafımızdaki nesnelere ait, uzay zamanda sürekliliği olan ve görme sistemleri ile algılanabilen sinyalleri ifade eder. Cotton ve Richard (1997) analog görüntüleri; kaynağındaki biçimiyle kaydedilebilen, saklanabilen ve iletilebilen görüntüler olarak tanımlamışlardır (Karsan, 2008). Doğada sayısal olarak ifade edilmeyen bütün görüntüler analog görüntülerdir. Bir bilgisayarın kendi görüntüsü analog görüntü iken, ekranda dijital olarak oluşturulan görüntü ise sayısal görüntüdür.

1.3.2. Sayısal Görüntü

Sayısal (Ayrık) görüntü; uzay zamanda bulunan görüntülere ait analog sinyallerin elektronik araçlar (tarayıcı, sayısal kamera, uydular vb.) aracılığıyla algılanıp bir sayı tablosu şekline dönüştürülmesiyle (sayısallaştırma/digitizing) oluşturulan dijital verilerdir. Sayısal görüntü, analog sinyalin sayısal sinyale dönüşmüş halidir (Bayram, 2013). Analog sinyaller şeklinde kamera film kasetlerine kaydedilmiş görüntülerden de çeşitli programlar aracılığıyla sayısal görüntüler oluşturulabilir. Analog görüntüleri yakalamak, saklamak, değiştirmek ve görmek üzere dijital ekipmanlar kullanılırken, bu görüntüler ilk olarak sayısallaştırma ya da tarama olarak adlandırılan bir süreç içerisinde bir dizi numaraya dönüştürülmelidir (Al-Karawi, 2012).

Elektronik aletlerin ekranlarında oluşan hareketli, hareketsiz bütün sayısal görüntüler bilgisayar grafiklerinin temelini oluştururlar. Görüntü; gerçek yaşamdaki üç boyutlu nesnelerden oluşan bir sahnenin basit iki değişkenli bir fonksiyonudur (Karakoç, 2011).

Sayısal görüntü, matematiksel olarak iki boyutlu bir f(x,y) fonksiyonu ile ifade edilir. Sayısal görüntüler analog görüntülerden farklı olarak dijital ortamda ayrık zamanda ifade edilir. Yani sayısal görüntünün temelini ifade eden pikseller arasında herhangi bir bağ yoktur. Her görüntü birbirinden farklı niteliklerdeki piksellerin uzay zamanda arka arkaya çok hızlı gösterimiyle görüntü bütünlüğü oluşturulur. Sayısal görüntüleri ifade eden f(x,y) sayısal fonksiyonunda x, y değerleri görüntü piksellerinin uzay düzlemde konumlarını işaret etmektedir. f fonksiyon ise (x,y) noktasındaki pikselin yoğunluğunu (intensity) ya da gri seviyesi olarak adlandırılmaktadır (Kurtulmuş, 2012). Gonzales ve Wood (2002) Şekil 1.1'de sayısal görüntü modelini Eşitlik 1.1'de sayısal görüntünün matrissel durumunu aşağıdaki gibi ifade etmişlerdir.



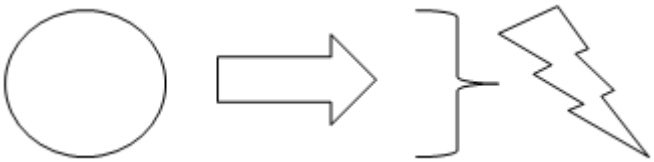
Şekil 1.1: Sayısal görüntülerin temel koordinat sistemi

Grafik koordinat sisteminde, sistemin bir orjini (0,0) ile eksenleri ve her bir eksenin artan değerlerine göre belirlenen doğrultuları vardır. Grafik koordinat sistemi, ekran üzerindeki herhangi bir konumun bir X, Y değeriyle tam noktasının belirlenmesini sağlar (Morrison, 2005).

$$f\ x,y\ = \begin{matrix} f(0.0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1.1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{matrix} \qquad 1.1$$

Görüntüler bilgisayarda; vektörel görüntüler ve raster görüntüler şeklinde ifade edilirler.

Vektörel Görüntüler: Çizim teknikleri kullanılarak oluşturulan matematiksel formüllere dayalı görüntülerdir (Karsan, 2008). Bütün nesneler her defasında koordinat sistemine dayalı matematiksel olarak yeniden ifade edildikleri için bu görüntülerde herhangi bir görüntü bozulması oluşmaz. Şekil 1.2 bazı vektörel görüntüler gösterilmiştir.

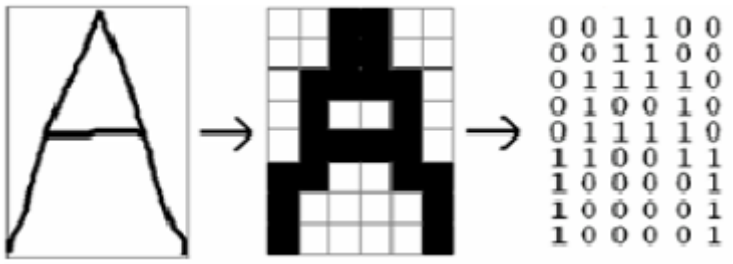


Şekil 1.2: Bazı vektörel görüntüler

Vektörel görüntülerde, nesnelerin nasıl oluşacağı matematiksel ifadeler ile tanımlandığına göre bu nesneler bellekte saklanmazlar.

Raster Görüntüler: Vektörel görüntülerden farklı olarak sayı matrisleriyle ifade edilen, her pikselin konumunun ve değerinin belli ve sabit olduğu görüntülerdir. Raster görüntülerin bilgisayar belleğinde depolanması gerekir. Raster görüntüler; slayt, diyapozitif, fotoğraf, şema, harita, grafik gibi bilgi taşıyan belgelerin yâda radar, algılayıcı gibi cihazların ürettiği sinyallerin sayısallaştırılmasıyla, dijital fotoğraf veya video kameralarından doğrudan sayısal görüntünün alınmasıyla elde edilir (Açıkgöz vd,1999).

Analog sinyalden elde edilmiş bir sayısal görüntünün temel yapısı ikili (binary) sayılar matrisi şeklinde ifade edilen sayı dizisinin her bir elemanını ifade eden piksellerdir. Şekil 1.3'de analog sinyalden oluşturulmuş bir karakterin ayrık zamanlı, piksel ve ikili sayısallaştırılmış görüntüsü verilmiştir (Karakoç, 2011).



(a) (b) (c)

Şekil 1.3:(a) A karakteri, (b) raster gösterim, (c) ikili sayısallaştırma

1.4. Görüntü İşleme Teknikleri

Görüntü işleme; ölçülmüş veya kaydedilmiş olan sayısal görüntü verilerini, elektronik ortamda bilgisayar ve yazılımlar yardımı ile amaca uygun şekilde değiştirmeye yönelik olarak yapılan bilgisayar çalışmasıdır (Karabiber, 2009). Amacı, mevcut görüntü verisinden yeni bir veri elde etmek olan görüntü işleme teknolojileri, bu amaca ulaşmak için verinin özelliğine ve sistemin amacına göre günümüzde çok çeşitli algoritmalar, otomatik sistemler geliştirdiği görülmektedir.

Arslan (2011), görüntü işleme basamaklarını; girişteki görüntü üzerinde bazı temel işlemlerin uygulanıp mevcut görüntüyü bir sonraki işlem seviyesine hazırlayan düşük seviye, iyileştirilmiş olan görüntüden anlam çıkarabilmek için görüntüyü daha küçük birimlere

ayıran orta seviye ve anlamlar, kararlar çıkarılabilecek verilere dönüşmüş olan görüntü parçalarını gruplandıran yüksek seviye işlemlerdir şeklinde tanımlamıştır.

1.4.1. Görüntü İyileştirme

Sayısal bir görüntünden daha kesin sonuçlar çıkarabilmek için uygulanan düşük seviye işlemleri ifade eder. Görüntü iyileştirme aşamasında ihtiyaç duyulan teknikler tamamen sayısal görüntünün sonuç çıkarma için yeterli düzeyde veriye sahip olup olmadığıyla alakalıdır.

1.4.1.1.Parlaklık (Brightness)

Sayısal bir görüntüde siyahların tam siyah, beyazların tam beyaz olma durumlarıdır (URL-3, 2013). Parlaklık değeri sayısal görüntüde, dijital görüntüleri ifade eden f(x,y) fonksiyonunun yoğunluğuyla alakalıdır. Sayısal bir görüntü üzerinde işlemler yapıldığında görüntü içerisinde ilgilenilen nesnenin/lerin parlaklık değeri bu görüntünün işleme başarısıyla doğrudan bağlantılıdır. Gonzalez and Woods (2002) sayısal görüntülerin parlaklıklarını, pozitif veya negatif bir sayı ile toplanması durumunda parlaklık değerinin değişeceğini Eşitlik 1.2'deki gibi ifade etmiştir (Arslan, 2011).

$$g(x,y) = f(x,y) + b \qquad 1.2$$

Parlaklık değişkeni olan b'nin pozitif negatif değerlerinde oluşan sonuçlar Şekil 1.4'te gösterilmiştir.

b= -30 b = 0 b = +30

Şekil 1.4: Farklı parlaklık değerlerinde oluşan görüntüler

1.4.1.2.Karşıtlık

Sayısal görüntülerdeki en parlak alan ile en karanlık alan arasındaki farktır (URL-4, 2013). Karşıtlık (Contrast) görüntülerde alanlar arasındaki farkı belirginleştirdiğine göre görüntü işlemede, kontrast değerinin uygun seçimi sonraki seviye işlemlerin başarısı için oldukça kritik önem teşkil etmektedir. Analog sinyallerden görüntü oluşturulması sırasında kötü aydınlatma, görüntüleme sensörü dinamik aralık eksikliği, objektif sorunları nedeniyle sayısallaştırma esnasında görüntü parlaklığında veri kayıpları oluşur (Gonzalez and Woods, 2002). Bu nedenle analog sinyallerden oluşturulan bütün görüntü işleme işlemelerinde karşıtlık ayarı olabildiğince hassas yapılmalı. Gonzalez and Woods (2002) sayısal görüntünün parlaklık değerinin pozitif ya da negatif değerler alabilen bir değişkenle çarpılması ile karşıtlık değerinin değiştiğini Eşitlik 1.3 ile ifade etmiştir (Arslan, 2011).

$$g(x,y) = a*f(x,y) \qquad 1.3$$

Şekil 1.5' te aynı parlaklık değerine sahip sayısal bir görüntünün pozitif ve negatif karşıtlık değerlerinde oluşan görüntüleri verilmiştir.

a = -70 a = 0 a = +70

Şekil 1.5: Farklı karşıtlık değerinde oluşan görüntüler

1.4.1.3.Eşikleme

Eşikleme (Tresholding); gri seviyede tanımlanmış sayısal bir görüntünün 0-255 aralığındaki piksel değerlerinin belirlenen bir eşik değerin üstünde sayısal niceliğe sahip piksellerin değerini 1 beyaz, bu eşik değerinin altında niceliğe sahip piksellerin değerini 0 siyah kabul ederek görüntünün sayısal değerlerini yeniden tanımlama işlemidir. Eşikleme temel olarak gri seviyede tanımlanmış görüntüler üzerinde uygulanan bir işlem olduğu için eşik değeri 0-255 aralığında seçilmek zorunluluğu vardır. Eşikleme sonucunda görüntülerin piksel değerleri sadece 0 ve 1 değerlerini aldıklarından bu işleme görüntüyü ikilileştir-

me de denilebilir. Eşikleme işlemi farklı yoğunluktaki veya renkteki ön ve arka plan görüntülerini bölütlemede kullanılan en temel yöntemdir (Kurtulmuş, 2012). Şekil 1.6 gri seviyede tanımlanmış bir sayısal görüntünün farklı eşik değerlerindeki sonuçları gösterilmektedir.

Şekil 1.6:(a) gri seviye görüntü, (b) 25 eşiğine göre ikili görüntü, (c) 50 eşiğine göre ikili görüntü.

1.4.1.4.Görüntü Negatifi

Sayısal görüntülerin piksel yoğunluk değerlerinin tersini ifade eder. Negatifleştirme işlemi (Negation); ikilileştirilmiş görüntü de beyazları siyaha, siyahları beyaza çevirir. Negatifleştirme işlemini doğadaki renklerin terslerini ya da tümleyenlerini verir. Negatifleme renkli, gri ve ikili görüntülere uygulanabilir. Negatifleştirme işlemi temel olarak bir görüntünün karanlık kısımlarında saklanmış beyaz ya da gri seviyeli ayrıntıları ortaya çıkarmak için kullanılmaktadır (Kurtulmuş, 2012). Negatifleştirme aynı zamanda olumsuzlaştırma olarak ta ifade edilmektedir. Şekil 1.7 de sırasıyla gri ve ikilileştirilmiş sayısal görüntülerin negatifleri verilmiştir.

(a) (b)

(c) (d)

(e) (f)

Şekil 1.7: (a) renkli görüntü, (b) renkli görüntünün negatifi, (c) gri görüntü, (d) gri görüntü negatifi, (e) ikili görüntü, (f) ikili görüntü negatifi

1.4.1.5.Sayısal ve Mantıksal İşlemler

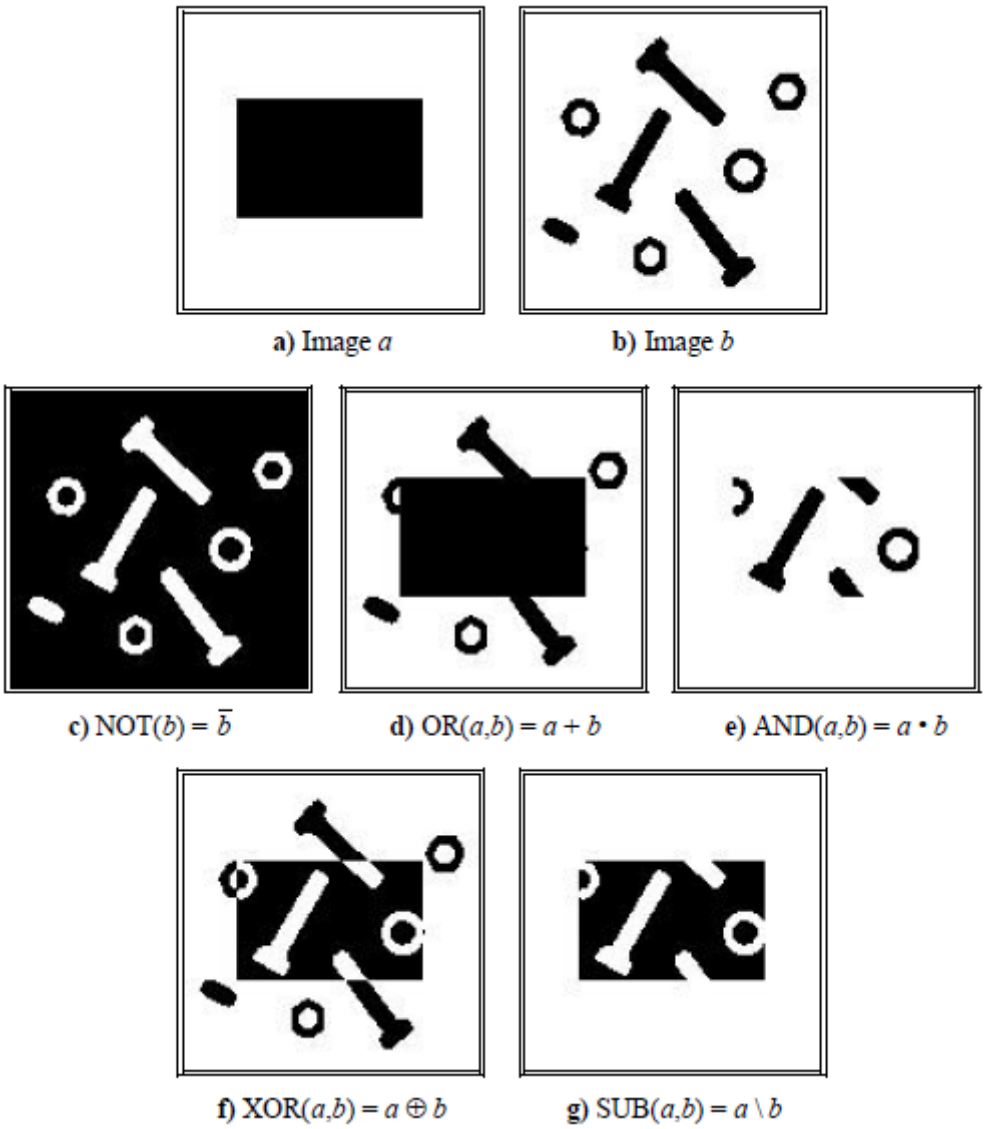
Aritmetik ve mantıksal işlemler temel olarak görüntü iyileştirmekten ziyade görüntü içerisinde bulunan hedef alanların tespitine yönelik ön operasyonlardır. Çalışmalar incelendiğinde sayısal görüntülerde toplama ve çıkarma işlemleri daha çok iki sayısal görüntüyü karşılaştırıp iki görüntü arasındaki farkı ortaya koymayı amaç edinir. Şekil 1.8 orijinal görüntüde yer alan bozuk paraların yeri değiştirildikten sonra fark işlemiyle iki görüntü arasındaki farkı ortaya koyar (Arslan, 2011).

(a) Orijinal görüntü (b) Değiştirilmiş görüntü (c) Fark görüntüsü

Şekil 1.8: İki görüntünün farkı

Toplama ve çıkarma işlemelerinden farklı olarak çarpma ve bölme işlemleriyle genel olarak görüntü piksellerinin yoğunluklarını değiştirerek parlaklık kontrast gibi görüntü iyileştirme işlemlerinde tercih edilirler.

Mantıksal işlemler, sayısal işlemlerden farklı olarak sayısal verilerin morfolojik özellikleriyle ilgili değişiklikler yapar. Görüntü işlemede temel AND, OR ve NOT işlemleri kullanılır, diğer mantıksal işlemler bu üç işlemin belli bir kombinasyonu ile elde edilirler. NOT işlemi; renkli, gri veya ikili bütün görüntülerin piksel değerlerinin tersini alır. Bir bakıma negatifleme işlemidir. AND ve OR işlemleri ise görüntülerden hedef görüntüleri ayırmak için maskeleme sağlarlar. Bu işlemler amaca ve görüntünün yapısına göre arka plan verisi ya istenilen nesnenin kendisini tek bir sayısal veri ile maskeler. Böylelikle üzerinde çalışılmak istenen görüntü alanı belirginleşmiş olur. Young vd (1998) ikili tanımlanmış iki görüntü üzerindeki mantıksal işlemleri Şekil 1.9'daki gibi vermiştir.



Şekil 1.9: İkili tanımlanmış görüntüler üzerinde çeşitli mantıksal işlemler

1.4.1.6.Filtreleme

Analog sinyallerin sayısallaştırılması esnasında yetersiz ışık gibi çevre koşulları, algılayıcıdan, görüntünün kendi fiziksel bütünlüğünden kaynaklanan durumlara bağlı olarak görüntü içinde işlem yapmayı güçleştirecek görüntü kayıpları veya gürültüler bulunabilir. Görüntüyü işlem yapmaya uygun hale getirmek için kayıt esnasında istenmeyen türden oluşan bu görüntüleri elemine etmek veya görüntü kayıplarını telafi etmek için filtreler kullanılır. Görüntüye hangi veya nasıl bir filtrenin uygulanacağı tamamen sayısal görüntünün bütünlüğüne, istenilen amaca uygunluğuna göre değişir.

Filtreleme, görüntü üzerinde bir filtre varmış gibi görüntü piksellerinin sayısal değerlerinin yeniden hesaplanmasıdır (Siyah, 2013). Filtrelemeler aracılığıyla görüntüleri amaca uygun netleştirme, yumuşatma, keskinleştirme ve kenar bulma işlemleri uygulanır. Filtreler görüntü pikselleri üzerinde doğrusal ve doğrusal olmayan türden işlemler yaparlar. Teknik olarak doğrusal filtreler konvolüsyon veya fourier dönüşümleri kullanılarak uygulanır ve doğrusal filtrelerin yetersiz kaldığı durumlarda doğrusal olmayan filtreler kullanılır (Khurshed, 2009). Filtreler tektip (kare, dikdörtgen, dairesel) veya üçgensel (piramid, koni) olarak tanımlanabilirler.

1.4.2. Kenar Belirleme İşlemleri

Sayısal görüntü işleme sürecinin orta düzey işlemlerini ifade eder. Görüntü iyileştirme operasyonlarıyla içerisindeki nesne sınırları optimize edilen görüntü, kenar belirleme algoritmalarıyla sınırları belirlenerek yüksek seviye görüntü işleme için hazır hale gelir. Görüntüleri sınıflandırma ve örüntü eşleştirme için kenar belirleme, görüntü işleme sistemlerinin kritik aşamalarındandır. Görüntüde, birbirinde oldukça farklı gri seviye ya da renk yoğunluklarına sahip bölgeler arasındaki geçişler kenar olarak tanımlanmaktadır (Çulha, 1996). Kenar tabanlı bölütleme algoritmaların amacı özellik değerlerinde görüntünün yüksek değişime sahip olduğu noktalarda nesne sınırlarının yerini belirlemektir (Karabatak, 2010). Renkli, gri ya da ikili görüntülerin pikselleri arasındaki anlamlı süreksizlikler belirlemede genel olarak Gradient, Canny ve Laplacian tabanlı kenar bulma algoritmaları tercih edilmektedir. Kenar tabanlı algoritmaların çoğu görüntüdeki yerel kenar bilgisini incelemede uzamsal bilgiyi kullanır (Karabatak, 2010). Gradient tabanlı algoritmalar görüntünün 1. türevinin maksimum ve minimum değerlerine kenar bulma işlemini uygular. Laplacian görüntünün 2. türevindeki 0 geçişlerine göre kenarları çıkarır (Kazan, 2013). Bunlardan farklı olarak Canny, görüntü üzerinde daha hassas işlem yaparak görüntünün kenarlarını dört adımda çıkarmıştır.

1.4.2.1.Sobel Kenar Çıkarma

Gradient tabanlı kenar çıkarma algoritmasıdır. Görüntünün içerisindeki alanların yoğunluk seviyesi değişimi gradient ile ifade edilir. Bu görüntüye ait sayısal fonksiyonun gradienti Eşitlik 1.4'teki gibi ifade edilir (Gonzalez and Woods, 2002).

$$\nabla f = \begin{matrix} G_x \\ G_y \end{matrix} = \begin{matrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{matrix} \qquad 1.4$$

Gradient (kenar) büyüklüğü Eşitlik 1.5'te verilen denklemlerden biriyle bulunur.

$$\nabla f = mag\ \nabla f \qquad \text{a.}$$

$$\nabla f = G_x^2 + G_y^2 \ ^{\frac{1}{2}} \qquad \text{b.} \qquad 1.5$$

$$\nabla f = \frac{\partial f}{\partial x} \ ^2 + \frac{\partial f}{\partial y} \ ^2 \ ^{\frac{1}{2}} \qquad \text{c.}$$

En sık kullanılan gradient tabanlı kenar bulma operatörleri Sobel, Roberts ve Prewitt operatörleridir. Bu operatörler kullandıkları maske boyutu ve bu maskelerin kernel (çekirdek) değerleri farklıdır. Bu operatörlerden kenar bulma başarısı en iyi olan Sobel operatörüdür. Sobel operatörü; sayısal görüntü, fonksiyon olarak değerlendirildiğinde, bir nokta üzerindeki 3x3 komşulukta mümkün olan dört merkezi yönde elde edilebilir gradyan değerlerinin vektör toplamları şeklinde oluşturulması düşüncesine dayanır (Aybar, 2008). Robert işleci 2x2 lik konvülasyon matrisleri ile kenar belirlemeye çalışır. Kenar matrislerinin boyutlarının 2x2, yani küçük olması Robert operatörleriyle yatay düşey satır belirleme başarısını düşürür. Prewitt kenar operatörü yumuşatma ve yaklaşık türev alma süreçlerinden oluştuğu için Robert operatörlerine göre daha iyi sonuçların edinilmesini sağlar (Hazer, 2007). Şekil 1.10 gradient tabanlı en sık kullanılan Roberts, Prewitt ve Sobel maskeleri ile bu maskelerin yatay ve düşey kenarların tespiti için kernel değerleri verilmiştir.

-1	0
0	1

0	-1
1	0

(a) Robert çapraz eğim operatörleri

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

yatay dikey

(b) Prewitt operatörleri

-1	-2	-1
0	0	0
1	2	1

yatay

-1	0	1
-2	0	2
-1	0	1

dikey

(c) Sobel Operatörleri

Şekil 1.10: Gradient tabanlı kenar belirleme operatörleri

Yatay ve dikey olarak belirtilen konvülasyon maskeleri sırasıyla yatay maske ile görüntünün satırlarındaki kenarlar, dikey maske ile görüntünün sütunlarındaki kenarlar tespit edilir. Bu durumda bir pikseldeki kenarı tespit etmek için her piksel yatay ve dikey maskelerle taranarak hedef pikselin satır kenar bilgisi G_x ve sütun kenar bilgisi G_y olmak üzere iki değer elde edilir. Maskeleme işleminden sonra her pikselin yerine ait iki değer oluşur. Bu değerler kenar büyüklüğü (Eşitlik 1.5 ∇f) ve kenara ait yön bilgilerini elde etmek için kullanılır (Soytürk, 2005). Eşitlik 1.6 piksel kenarlarının oluşum yönlerini tespit eder.

$$\theta\ x,y\ = \tan^{-1}\frac{G_x}{G_y} \qquad 1.6$$

1.4.2.2.Laplacian Kenar Çıkarma

Görüntülerdeki kenarları bulmak için görüntü fonksiyonun ikinci türevinde 0 geçişlerine göre kenar belirleyen operatördür. Gradient operatörlerindeki sayısal fonksiyonların ikinci türevlerini aldıkları için yapısal olarak benzerlik gösterse de matematiksel olarak farklı bir duruma işaret etmektedir. Sayısal görüntünün f(x,y) fonksiyonunun birinci türevinde oluşan sinyalin maksimum noktası görüntü sayısal sinyalinin içerisindeki kenarın merkezidir. İkinci türev durumunda, birinci türev sinyalinin tepe noktası 0 olur. İki boyutlu sayısal görüntünün f(x,y) fonksiyonun ikinci türevi Eşitlik 1.7'deki gibidir (Gonzales and Woods, 2002).

$$\nabla^2 f = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \qquad 1.7$$

Laplacian operatörünün gradient operatörü ile benzerlik gösterdiği bir diğer nokta yatay ve dikey kenarların belirlenmesinde kullanılan 3x3'lük maskeleme matrisidir. Konvülasyon matris büyüklükleri aynı olsa da kernel değerleri farklıdır. Laplacien operatör matrislerinin kernel değerleri görüntünün yapısına göre değiştirilebilir. Şekil 1.11 farklı

kernel değerlerine sahip 3x3'lük laplacien matrisleri gösterilmektedir (Gonzales and Woods, 2002).

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

Şekil 1.11: Farklı sayısal görüntülere uygulanan konvolüsyon matrisleri

Laplacian, görüntülerdeki gürültülere çok duyarlı bir operatördür. Böylece görüntü içerisinde siyah arka plan üzerinde sürekliliği olmayan grimsi noktalar oluşturacaktır. Bu olumsuz duruma bağlı olarak laplacian uygulanacak görüntüye öncelikle gauss konvülasyon kerneli uygulanır. Gauss filtresi alçak geçiren özellikte olduğunda görüntü bulanıklaşır. Böylece işlenmemiş görüntüde kenar özelliği göstermeyen alanlar yumuşatılarak bastırılmış olur.

Gradyan operatörleri ile görüntüdeki kalın kenarlar, laplacian operatörleri, görüntüdeki ince kenarları ve gürültü noktalarını belirtirler (Kazan, 2013).

1.4.2.3.Canny Tabanlı Kenar Çıkarma

Bu yöntem gradient ve laplacian tabanlı her iki algoritmayı amaca uygun kullanarak en başarılı sonuçları verir. Canny ile kenarlar 4 adımda belirlenir. Birinci aşamada orijinal görüntüye gauss maskesi uygulanarak görüntü bulanıklaştırılır. Böylelikle görüntü içerisinde kenar olma özelliği taşımayan gürültü piksellerinin parlaklıkları bastırılmış olur. f(x,y) orijinal görüntü fonksiyonun, I(x,y) gauss yumuşatıcı filtre ile konvülasyonu sonucu oluşan görüntü s(x,y) eşitlik 1.8' de verildiği gibidir.

$$s\ x,y\ = I\ x,y\ * f(x,y) \qquad 1.8$$

İkinci aşamada, yumuşatılmış görüntüye gradient (Sobel, Robert, Prewitt) operatörlerinden biri uygulanarak kenarların şiddeti ve yönleri sırasıyla Eşitlik 1.9 ve Eşitlik 1.10'da verildiği gibi hesaplanır.

$$\nabla f = \sqrt{S_x^2 + S_y^2} \qquad 1.9$$

$$\theta\ \mathrm{x,y}\ = \tan^{-1}\ \frac{S_x}{S_y} \qquad 1.10$$

İkinci aşamada bulunan değerler uygun bir konvülasyon matrisi (2x2, 3x3) kullanılarak genlikleri nedeniyle kenar olabilecek fakat maksimum değerde olmayan bu noktaların gradyan değerleri kontrol edilip bastırılarak zemin ile aynı renk değerlerine dönüştürülür. Konvülasyon matrisi ζ(x,y) olmak üzere maksimum olmayan noktalar belirlendikten sonra oluşan görüntünün sayısal ifadesi Eşitlik 1.11'de verildiği gibidir (Arslan, 2011).

$$N\ x,y\ = nms(S\ X,Y\ ,\zeta(\mathrm{x},\mathrm{y})) \qquad 1.11$$

Canny algoritmasının dördüncü ve son aşamasında maksimum noktaları bastırılmış olan N(x,y) görüntüsü biri diğerinin yaklaşık 3 katı büyüklüğünde iki farklı sınır değer ile eşiklenir. Birinci eşik değeriyle maksimum noktaları bastırılmış N(x,y) görüntüsünde halen kenar gibi davranan gürültü alanlarının hepsinin 0 değerini alması amaçlanır. Böylelikle gürültüleri iyice bastırılmış görüntü yeni bir eşikleme değeriyle taranarak kenarların 1 değerini alması sağlanır.

1.4.3. Görüntü Bölütleme/Ayrıştırma İşlemleri

Bir görüntüde farklı özniteliklere (piksel yoğunluğu, piksel parlaklığı, renk vs) sahip alanların görüntüden bir bütün olarak çıkarılmalarını ifade eder. Görüntü işlemede kararlar, görüntülerden çıkarılan anlamlı bölgeler üzerinden yapıldığı için bölütleme; görüntü işlemenin en önemli ve kritik aşamasını temsil eder. Görüntü bölütleme; görüntüyü farklı alanlara, her bir alanın kendi içinde homojen olduğu fakat yakın iki alanın birleşimlerinin homojen olmadığı bölme işlemidir (Karabatak, 2010). Görüntü bölütleme piksellerin kendi içinde özniteliklerine göre yapıldığından her görüntünün yapısına bağlı olarak farklı bölütleme teknikleri tercih edilir.

1.4.3.1.Bölge Tabanlı Ayrıştırma

Görüntüde ardışık bulunan, aynı özniteliklere sahip görüntü alanlarının gruplandırılmasıdır. Temel mantık her pikselin kendi etrafındaki pikseller ile belli bir özniteliğinin karşılaştırılması sonucu bu piksellerin gruplandırılmasıdır.

1.4.3.2.Eşikleme Tabanlı Ayrıştırma

Bu ayrıştırma yönteminde gri seviye tanımlanmış görüntü içerisinde bir nesne için belirlenen ortalama eşik değeri ile bu görüntünün bütün piksellerinin sırası ile karşılaştırılmasıdır. Karşılaştırma işlemi sonucunda hedef piksel eşik değerine eşit veya üzerinde bir değere sahip ise bu piksel aranılan piksel grubuna dâhil edilir, eşik değerinden küçük bir değer ise hedef piksel görüntünün zemini olarak yorumlanır. Eşikleme tabanlı ayrıştırmada görüntü histogramlarından faydalanılır. Eşik değeri, histogramların maksimum ve minimum noktaları göz önünde bulundurularak seçilir (Saphiro and Stockman, 2001).

1.4.3.3.Kenar Tabanlı Ayrıştırma

Kenar belirleme operatörleriyle çizilmiş görüntü sınırlarının içlerinde kalan alanların bir grup olarak etiketlenmesi işlemidir. Kenar tabanlı ayrıştırma işleminin başarısı, tamamen kenar belirleme operatörlerinin başarısına bağlıdır. Kenar belirleme operatörleri ile görüntü içerisindeki alanların tam sınırları tespit edilmişse, kenar tabanlı ayrıştırma ile çok iyi sonuçlar alınabilir. Fakat görüntü içerisinde aynı iki alan arasında farklı yoğunluk değerlerine sahip olabilen kenarlar, kenar bulma operatörleriyle tam olarak bulunamaz ve bu görüntülerde açık sınırlar olduğunda kenar referanslı bölütleme işlemi ile yanlış alanlar gruba dahil edilmiş olur. İçlerinde çok fazla miktarda nesnenin bulunduğu sayısal görüntülerde yapılan kenar bulma işlemleri sonucunda çok sayıda kırık kenarın oluşmasından dolayı kenar tabanlı ayrıştırma bu türden görüntülerdeki nesneleri gruplandırmak için uygun bir yöntem olmadığı söylenebilir. Bu türden görüntüleri kenarlarına göre gruplandırmak için açık olan kenar uçlarının birleştirilmesi gerekmektedir.

1.5. Dijital Ortamda Görüntü Türleri

Sayısal görüntüler, aynı mantıksal yapıya sahiptirler. Sayısal görüntüde her pikselin Kartezyen koordinatlarını belirleyen x, y değerleri ile bu koordinatlardaki pikselin ışık yoğunluğunu belirleyen f fonksiyonundan oluşur. Daha önceki konularda işlendiği gibi f(x,y) şeklinde ifade edilir. Sayısal görüntülerde pikseller dijital ortamda iki boyutlu matris diziler gibi arka arkaya satır ve sütunlarda oluşturularak görüntünün dikdörtgensel yapıda olması sağlanır. Temelde bütün sayısal görüntüler her pikseli bir f(x,y) fonksiyonu ile ifade

edilmiş iki boyutlu bir sayı matrisi yapısındadır. Sayısal görüntüler f(x,y) fonksiyonlarının alabildiği değer ve gri seviye katman sayılarına göre birbirlerinden ayrılırlar.

1.5.1. İkili Görüntü

Piksel değerleri sadece 0 siyah ve 1 beyaz değerlerini alan tek kanaldan oluşan görüntülerdir. İkili görüntüler (Binary Image) genel olarak sayısal bir görüntüde aynı özniteliğe sahip alanları belirlemek için kullanılan görüntü işleme ara basamağıdır. Renkli veya gri seviye görüntüler eşikleme işlemi ile ikili görüntülere dönüştürülebilir. İkili görüntülerin oluşturulma başarısı seçilen eşik değeri ile doğrudan bağlantılıdır. Şekil 1.12 (a)gri düzey tanımlanmış bir görüntü, (b) 90 eşik değerine göre ikilileştirilmiş görüntü verilmiştir.

(a) Gri seviye görüntü (b) İkili görüntü

Şekil 1.12: İkilileştirilmiş tek katmanlı görüntü

1.5.2. Gri Seviyeli Görüntü

Gri görüntü (Gray Scale Image); piksel değerleri 0-255 arasında yoğunluk değerine sahip tek katmanlı sayısal görüntüdür. Bu görüntülerin her pikseli 1 Byte yani 8 bit ile ifade edilir. 8 bit ile bilgisayarda maksimum 2^8=256 sayı ifade edilebilir. Piksel değerleri 0 (siyah) minimum parlaklık değerinden başladığı için 8 bit bilgi ile bir piksel en fazla 255 (beyaz) maksimum parlaklık değerinde olabilir. 0 ve 255 değerleri sırası ile siyah ve beyaz durumlarında oldukları için aradaki 1-254 değerleri gri seviye renk durumlarını oluşturur-

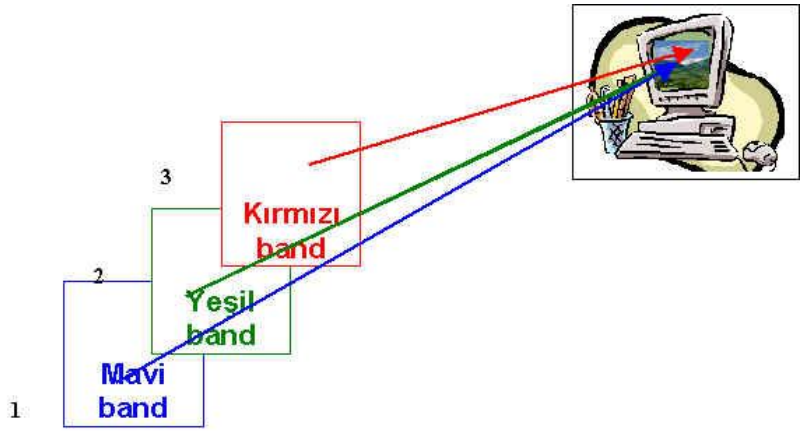
lar. Şekil 1.13 (a) gri seyiye görüntü dağılımı, (b) gri seviye tanımlanmış tek katmanlı görüntü.

(a) (b)

Şekil 1.13: (a) gri seviyeli görüntü, (b) gri seviyeli renklerden tek katmanlı oluşan görüntü

1.5.3. Renkli Görüntüler

Kırmızı, yeşil ve mavi renklerini temsil eden üç gri seviyeli katmanın arka arkaya gösterilmesiyle oluşan görüntülerdir. Renkli görüntü katmanları sadece satır ve sütun bilgilerini saklayan iki boyutlu matrisler değildir, iki boyuta ilaveten bir boyutta renk dizini vardır (Er, 2011). Renkli görüntülerde her piksel üç katmandan oluşur. Her katmanda bir piksel 8 bit ile ifade edildiğinden renkli görüntülerde bir piksel 3x8=24 bit ile ifade edilir. Şekil 1.14: renkli görüntü katmanlarının bilgisayar ekranında gösterilme sırası (Bayram, 2013)



Şekil 1.14: Görüntü katmanları sırasıyla Kırmızı, Yeşil ve Mavi olarak ekranda oluşturulur.

Elektromanyetik spektrumda kırmızı (Red) 0.4-0.5, yeşil (Green) 0.5-0.6 ve mavi (Blue) 0.6-0.7 dalga boylarındadır. Renkli görüntülerin her bir gri seviye katmanlarında piksel değerleri renklerin elektromanyetik spektrum aralıkları için yeniden hesaplanır. Şekil 1.15 renkli görüntünün (a) kırmızı, (b) yeşil ve (c) mavi katmanları gri düzey renk yoğunlukları verilmiştir.

(a Kırmızı) (b Yeşil) (c Mavi)

Şekil 1.15: renkli görüntü katmanlarının farklı düzey renk yoğunlukları

1.6. Görüntü Karakteristikleri

Sayısal bir görüntüde her bir bölgenin ayırt edici özellikleri o görüntünün ya da görüntü içerisindeki nesnenin karakteristiğini ifade eder. Görüntü karakteristikleri, görüntü içerisinde sadece bir nesne veya bir bölge ile ilgili olabileceği gibi görüntünün genel karakteristiğini de ifade edebilir. Renk, doku, şekil, piksel yoğunlukları, histogramlar, parlaklık, çözünürlük ve karşıtlık gibi nitelikler görüntüleri karakterize ederler. Sayısal görüntülerde görüntü bölütleme, görüntü karakteristiklerine göre yapılır. Görüntü işlemeye başlamadan önce görüntü karakteristikleri belirlenir ve bu karakteristiklerden bölütleme başarısını en iyi etkileyecek hedef nitelik seçilir.

1.6.1. Çözünürlük

Çözünürlük (Resoluotion) bir görüntüde bulunan detayların sayısal miktarını ifade eder. Bu bakımdan çözünürlük; görüntüyü oluşturan piksellerin toplam miktarıdır. Çözünürlük; sayısal görüntüdeki detayların ayır edilmesiyle ilgilidir (Toyran, 2008). Çözünürlük, görüntüdeki piksel sayısını ifade ettiğinden, bir görüntünün çözünürlüğünü ifade etmek için görüntüyü oluşturan toplam piksel sayısını hesaplamak gerekir. Daha önceki ko-

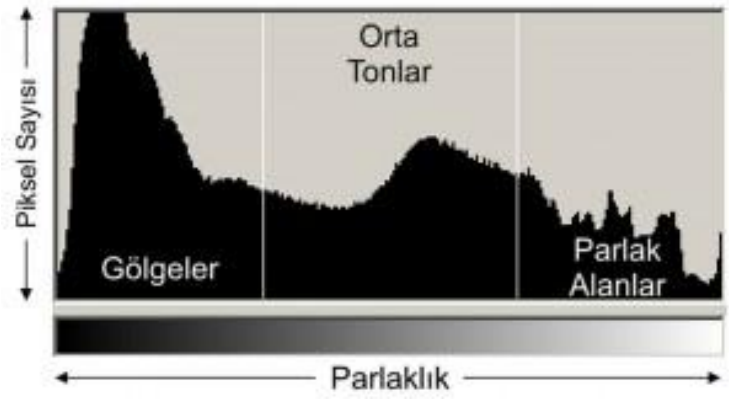
nularda sayısal görüntünün, her piksel değeri f(x,y) fonksiyonu ile tanımlanmış iki boyutlu satır ve sütunlar matrisiyle ifade edildiği belirtilmişti. Bu iki boyutlu matristeki eleman sayısı, yani her satırdaki piksellerin toplam sayısı bize çözünürlük değerini verir. Yani X piksel yüksekliğinde ve Y piksel genişliğindeki bir görüntünün çözünürlüğü X*Y işleminin sonucu kadardır. Bu açıdan 800 satır 600 sütundan oluşan bir görüntünün çözünürlüğü 800*600 = 480000'dır.

Bir sayısal resimde örnekleme yapılan uzamsal frekans (örnekleme frekansı) değeri çözünürlüğü göstermek için kullanılan nesnel bir ölçüttür (Yalman, 2010). Sayısal görüntüleri örnekleme frekansı PPI (Pixel Per Inch: 1 inçlik alanda piksel) ya da DPI (Dot Per Inch: 1 inçlik alanda nokta) ile ifade edilir. Örnekleme frekansı ne kadar yüksekse görüntü çözünürlüğü o kadar iyi olur. Şekil 1.16; fiziksel boyutları aynı görüntülerin örnekleme frekansına bağlı çözünürlükleri verilmiştir (Yalman, 2010).

Şekil 1.16: Çözünürlük ile görüntü boyutları arasındaki ilişki

1.6.2. Histogram

Sayısal görüntülerde, her piksel değerinden görüntüde kaç adet olduğunu gösteren grafiktir. Histogramlar hem ikili hem gri seviye ve hem de renkli görüntüler için oluşturulabilir. Histogram grafiklerinden görüntünün parlaklık durumu, kontrastı ya da tonları hakkında bilgi sahibi olunabilir (Akar, 2009). Görüntü bölütleme de histogram grafiklerinden, genellikle uygun eşik değerinin belirlenmesi ve nesnelerin tespiti için faydalanılmaktadırlar. Şekil 1.17 de sayısal görüntünün histogramı ile bu sayısal görüntü karakteristiklerinin histogramda dağılım alanları gösterilmiştir (URL-5, 2013).



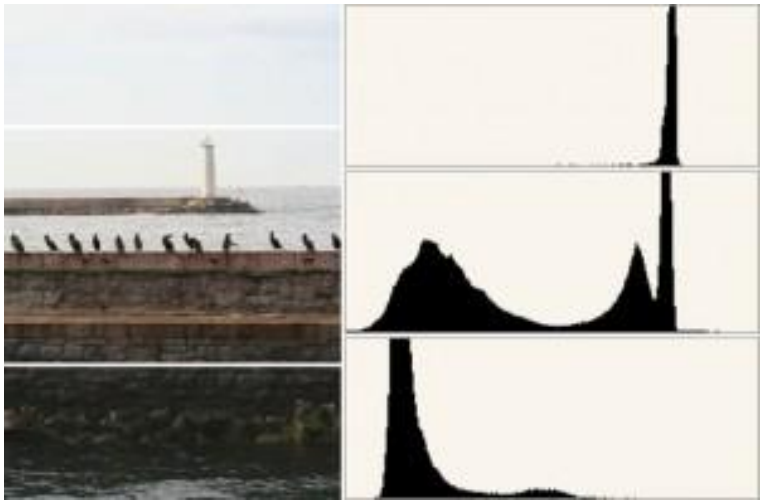
Şekil 1.17: Histogram grafiği

Histogram grafiklerinde 0 değerine yakın bölgeler gölgeleri, 255 değerine yakın değerler parlaklıkları temsil etmektedirler. Şekil 1.18; aynı görüntünün karanlık, orijinal ve parlak örnekleri için histogram grafikleri oluşturulmuştur (URL-5, 2013).

Şekil 1.18: Farklı piksel yoğunluklarına sahip görüntülerin histogram grafikleri

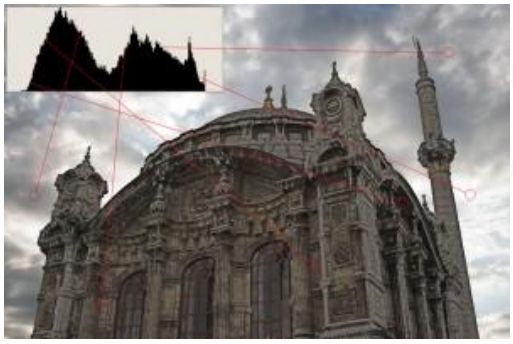
Histogram grafiklerinden görüntülerin kontrastları hakkında da yorum yapılabilir. Düşük kontrastlar histogram grafiğinde dar bir alanda dağılım gösterirken, yüksek kontrastlar ise daha geniş bir tonal alan dağılımı gösterirler. Şekil 1.19'da aynı resim üzerinde

düşük kontrastlı üst ve alt bölgelerin histogram dağılımları ile bu bölgelerden daha yüksek kontrast oranına sahip orta bölgenin histogram grafikleri verilmiştir (URL-5, 2013).



Şekil 1.19: Farklı kontrast oranlarındaki görüntü alanlarının tonal dağılımları

Histogram grafiklerinde görüntülerin dağılım gösterdikleri alan tahmini olarak belirlenebilir. Şekil 1.20; sayısal görüntünün işaretlenen noktalarının histogram grafiğindeki yerleri gösterilmiştir (URL-5, 2013).

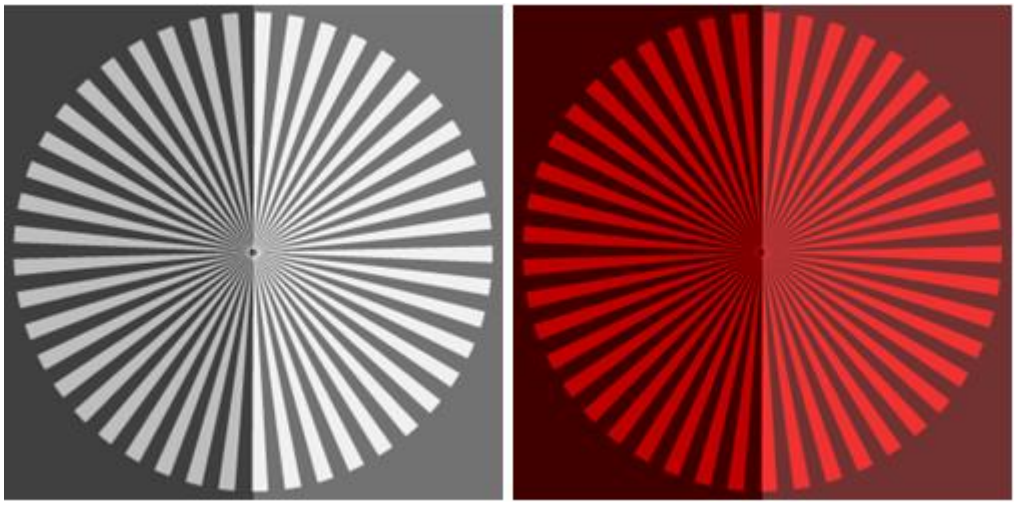


Şekil 1.20: Örnek bölgelerin histogram üzerindeki dağılım alanları

1.6.3. Parlaklık

Parlaklık sayısal görüntülerde siyahların tam siyah ile beyazların tam beyaz olma durumlarını ifade etmektedir. Karakteristiksel eş parlaklık değerine sahip alanlar görüntü histogramından eşikleme ile seçilebilir ve bu alanlar belirlenen amaçlar doğrultusunda iş-

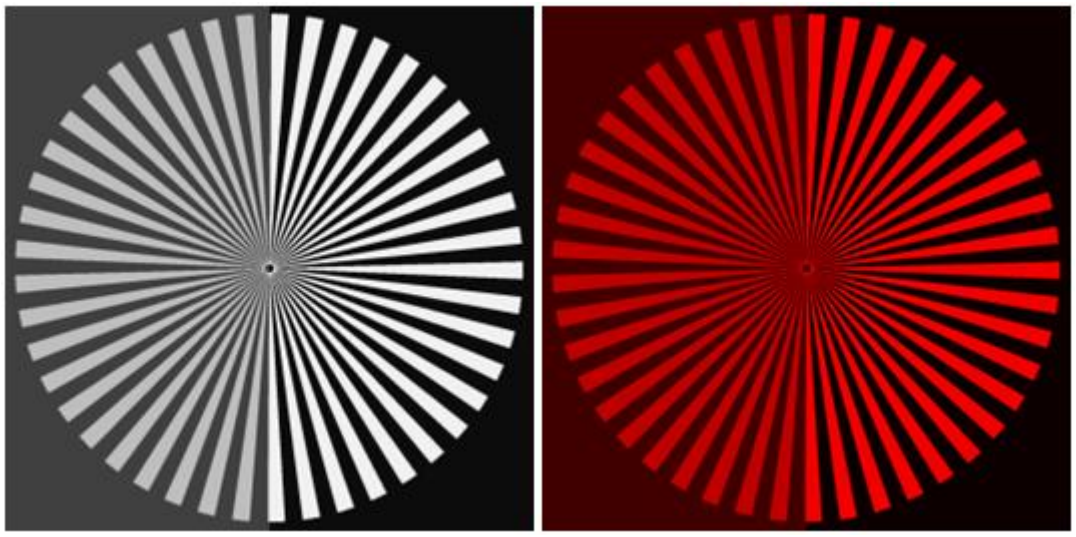
lenebilir. Şekil 1.21 de gri ve renkli görüntülerde piksel parlaklıkları, görüntülerin yarılarının parlaklık değerleri arttırılarak gösterilmiştir (Chaney, 2013).



Şekil 1.21: Parlaklıkları arttırılmış gri ve renkli görüntüler.

1.6.4. Karşıtlık

Karşıtlık, görüntüdeki en parlak ve en koyu alanları arasındaki mesafeyi ifade eder. karşıtlık işlemi ile görüntünün gölgeleri daha koyulaştırılarak parlaklıkları arttırılır. Görüntü işlemede karşıtlık amaca uygun olarak hedef görüntüyü bütün görüntü içerisinde belirginleştirerek görüntünün bölütlenmesini kolaylaştırır. Karşıtlık ayarı ile görüntü alanları birbirlerine yakınlaştırılabilir ya da uzaklaştırılabilir. Şekil 1.22'de görüntülerin sağ bölgelerinin karşıtlık değerleri arttırılmıştır (Chaney, 2013).



Şekil 1.22: Gri ve renkli görüntülerin sağ alanlarının parlaklıkları arttırılmıştır.

1.7. Dijital Renk Uzayları

Renk, mekânsal veya geçici ışık özelliklerini içeren, gözün retinasına ışığın uyarlanmasından kaynaklanan ve görsel algılamalar aracılığıyla bir gözlemcinin farkına vardığı ışıksal enerjidir (Hardeberg, 1999). Görüntü işlemenin renkler ile ilk ilişkisi bu ışıksal enerjiyi kayıpsız ve doğada olduğu şekliyle bilgisayar ortamına aktarmaya çalışmasıyla başlar.

Renkmetri biliminin temelinde; tüm renkleri temsil eden renk uzaylarındaki renklerin elde edilmesinde, birbirinden bağımsız üç değişken kullanılması gerektiği fikri yakmaktadır.

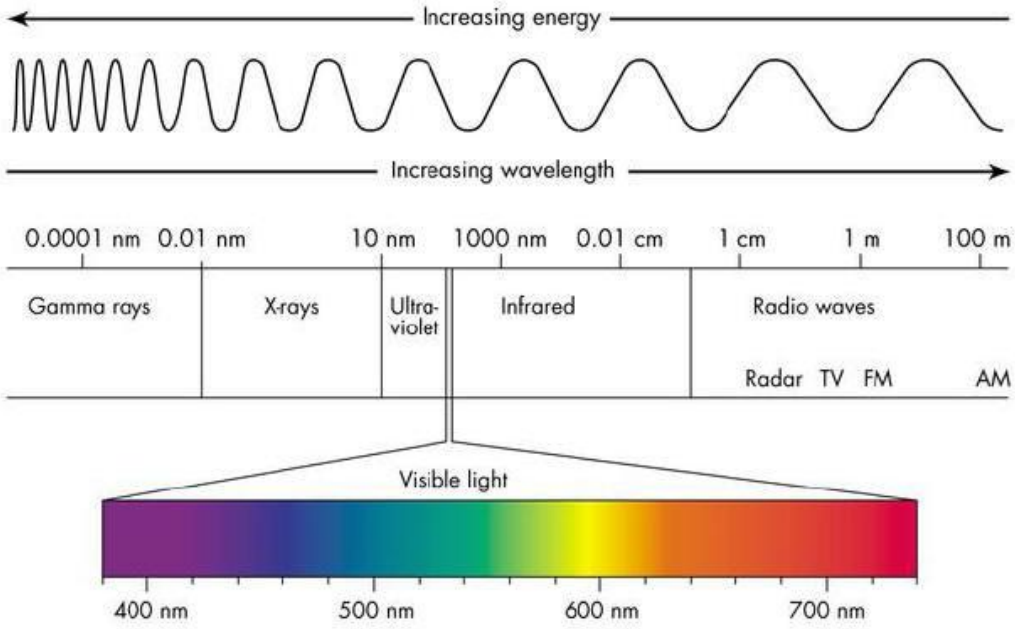
15. yüzyıldan başlayan ve günümüze kadar devam eden süre içinde rengin, ışığın taşıdığı bilgilerden biri yani ışığın bir özelliği olduğu ve renk algılamanın da görsel algılamanın bir parçası olduğu ortaya konulmuştur (Yılmaz, 2002). Elektronik cihazların gelişimi ile birlikte, renklerin sayısal ifade biçimleri üzerine 1900'lü yılların başından itibaren çeşitli çalışmalar yapılmıştır. Şekil 1.23'te şematize edildiği gibi renk uzaylarını cihaz bağımlı ve cihaz bağımsız diye sınıflandırmak mümkündür (Yılmaz, 2002).

Şekil 1.23: Renk uzayları

Renk uzayları şemasında da görüldüğü gibi günümüzde çok sayıda renk uzayı kullanılmaktadır. Bunlar arasında, bilgisayar grafiklerinde en çok tercih edilen RGB, baskı aletlerinde CMYK ve video sistemlerin YCbCr renk uzaylarıdır.

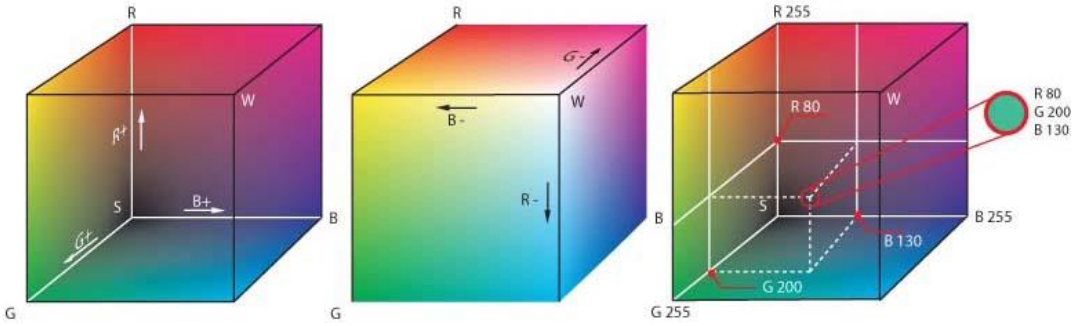
1.7.1. RGB Renkler

Bilgisayar ortamında, elektromanyetik spektrumda insan gözü ile görülebilen maksimum ve minimum dalga boyları arasındaki noktalarda oluşan renkler ile bilgisayarda renkler oluşturulur. İnsan gözü elektromanyetik spektrumda 400nm ile 700nm dalga boylarını algılar. Bir rengin sahip olduğu dalga boyunun grafiksel eğrisine, o rengin spektral tanımı denir (Katırcıoğlu, 2007). Doğada bulunan tüm renkler belirlenen spektrum dalga aralığındaki üç temel rengin belirli oranlarda karışmasıyla oluşur. Elektromanyetik spektrum göz ile görülebilir ışık aralığı Şekil 1.24'te gösterilmiştir (Özmercan, 2013).



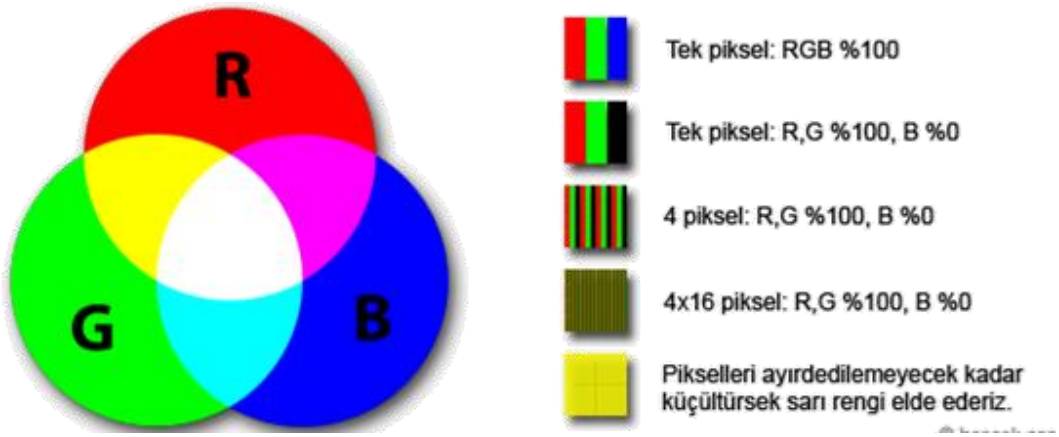
Şekil 1.24: Elektromanyetik spektrumda görülebilir ışık aralığı

RGB uzayı renkleri, bir birim küpün içinde toplam renk karışımı yöntemi ile kırmızı, yeşil ve mavi renklerin belli oranlarda karışımı ile renklerin oluşturulması fikrine dayanmaktadır. RGB renk uzayında renk oluşumlarını gösteren birim küp Şekil 1.25'te verilmiştir (URL-6, 2013).



Şekil 1.25: RGB renk uzayı birim küp

Ana renklerin kendi aralarındaki ikili kombinasyonları ikincil renkleri verir. RGB renkler additive (eklemeli) yani her renkteki ışık birbiri üstüne geldikçe aydınlık artar. Eklemeli RGB renklerinin üçü de, 0 değerinde birbirine eklendiği zaman siyah, 1 ya da 255 değerlerinde birbirlerine eklendikleri zaman en parlak renk olan beyaz elde edilir. RGB renk uzayında ana renkler ikişerli 1 ya da 255 değerinde birbirlerine eklendiklerinde sırasıyla, R-B magenta, B-G cyan ve R-G yellow (sarı) ikincil renkleri vereceklerdir. RGB uzayında renklerin toplanması ve oluşan ikincil renkler Şekil 1.26'te gösterilmiştir (URL-8, 2013).



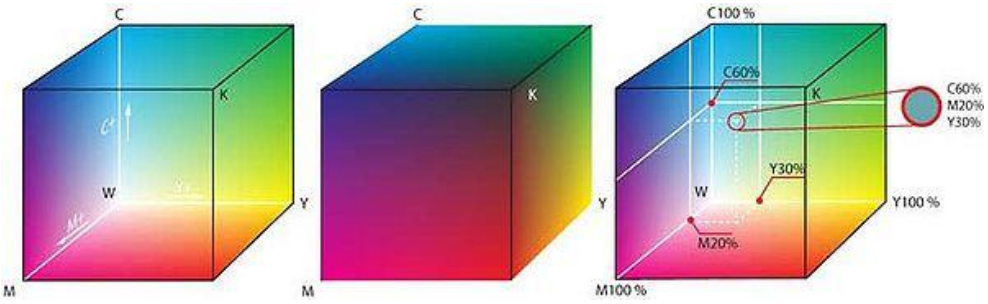
Şekil 1.26: Toplamsal ana renkler ve ikincil renkler.

1.7.2. Alfa RGB Renkler

RGB renk uzayı ile aynı renkleri üretir. Alfa değeri ile oluşturulan renk kombinasyonunda saydamlık etkisi yaratır. Alfa ile renklere 0.0 ile 1 arasında saydamlık değeri verilebilir. 0 değeri tam saydam renkler elde edilirken, 1 değeri ile renklerin saydam olmayan durumları elde edilir.

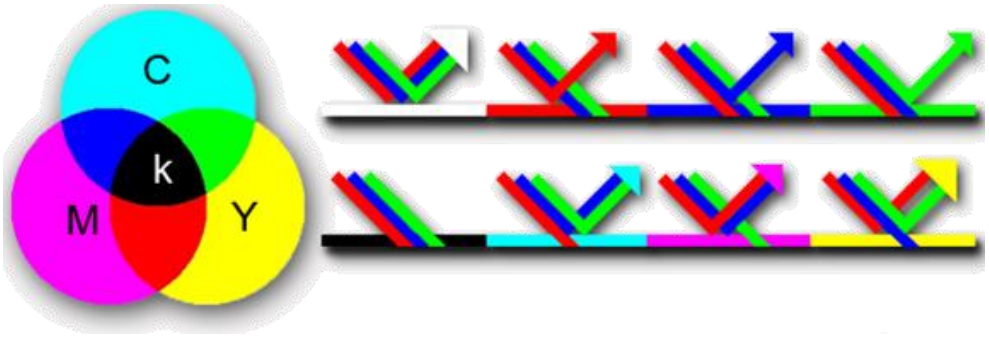
1.7.3. CMY/K Renkler

RGB renk modelinden farklı olarak kullanılan dört ana renkten, diğer renkler, çıkarım mantığı ile elde edilir. RGB birim renk küpünde ikincil renk olarak ifade edilen Cyan (Turkuaz), Magenta (Eflatun), Yellow (Sarı) ve Black (siyah) renkleri üst üste basılarak diğer tüm ara renkler elde edilir. RGB mantığında farklı olarak ara renkler çıkarma işlemi mantığı ile elde edilir. Bu renk uzayı ile ara renk elde edilme başarısı düşük olduğu için genelde baskı araçlarında tercih edilmektedir. İşlenmiş bir görüntü basılmadan önce RGB-CMY/K dönüşümü yapılması tavsiye edilmektedir. Şekil 1.27 birim küp CMYK renk uzayı verilmiştir (URL-7, 2013).



Şekil 1.27: CMYK renk uzayı birim küp

CMYK renk uzayında renkler birbirlerine eklendikçe ışığın bu renklerden yansıma oranı azalır. İkincil renkler ikişerli olarak birbirleri üzerine eklendiklerin RGB renk uzayında ki ana renkler oluşur. M-C mavi, C-Y yeşil ve Y-M kırmızı ana renklerini verecektir. Üç rengin eşit değer ve miktarda birbirlerine eklenmesi ile siyah rengi oluşur. CMYK renk uzayında renkler, RGB renk uzayındaki renklerden farklı olarak birbirlerine eklendikçe daha koyu renkler oluşur. Bu özelliğinden dolayı CMYK, çıkarımsal (subractive) renk uzayı olarak nitelendirilir. Şekil 1.28'de ikincil renklerden çıkarımsal olarak ana ve ara renklerin türetilmesi verilmiştir (URL-8, 2013).

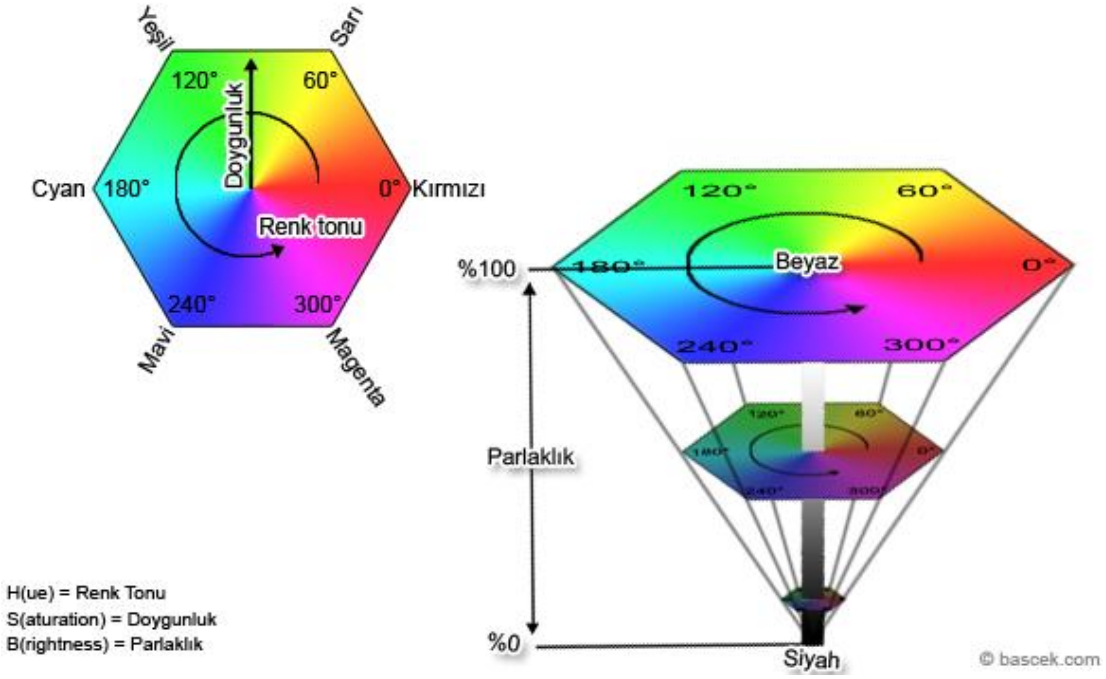


Şekil 1.28: Çıkarımsal ikincil renkler

1.7.4. HSV (Hue Saturation Value) Renkler

Bir rengin görsel özelliklerinin üç bileşen ile tanımlandığı düşüncesine dayanmaktadır. HSV renk uzayı, renkler üzerindeki işlemlerde daha çok sezgisel olması ve yaklaşık olarak insan görme sisteminin algı kabiliyetine yakın olması amacıyla geliştirilmiştir (Yalman, 2010). HSV renk uzayında, renk (Hue), doygunluk (Saturation) değeri ile oluşturulurken son bileşen görüntünün istenilen özellikte olması için ışık (Luminance), parlaklık (Brightness), yoğunluk (Intensty) değişkenlerinde herhangi biri olabilir. HSV renk uzayında renkler, RGB uzayından ana renklerin bir pikselde belli oranlarda yan yana görüntülenmesinden farklı olarak, her renk bir matematiksel denklem ile bir kerede oluşturulur. Renklerin el ile gösterilmeleri gerektiğinde ve kullanıcıların renkleri görerek seçmeleri gerektiği durumlarda idealdirler (Taşkın,2007).

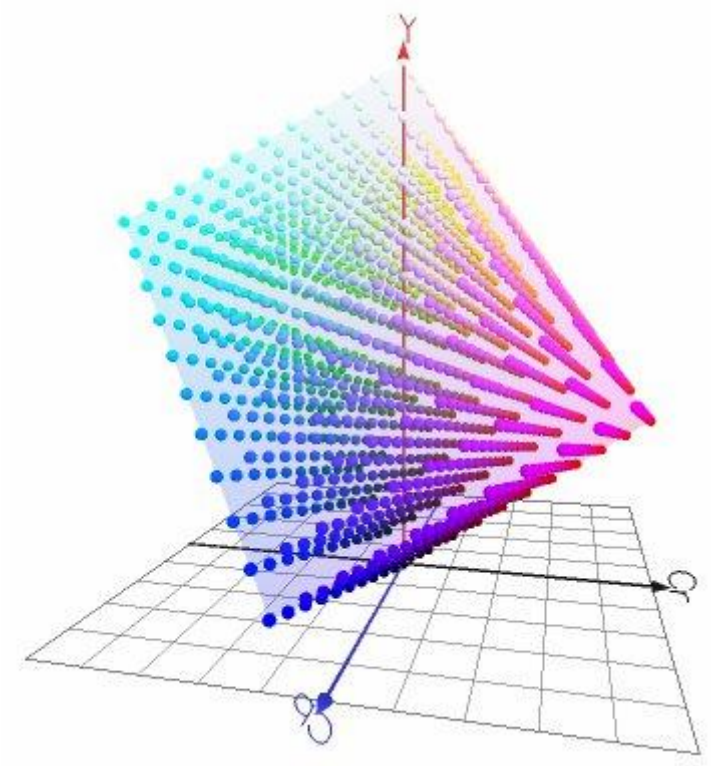
Renk tonunu ifade eden Hue bileşeni, renkleri açısal (0^o-360^o aralığında) olarak oluşturur. Renk geçişleri sırası ile 0^o kırmızı 120^o yeşil ve 240^o derece mavi olacak şekildedir. Her bir derecelik açısal değişimlerde ara renkler oluşur. Saturasyon değişkeni koninin merkezinde yarıçap uzunluğunu ifade eder. Doygunluk koninin yüksekliğini ifade eder ve 0-1 aralığında değer alır. Koninin merkez ekseninde, tüm ana renkler birleştiği için beyazdır. Konin merkez ekseninde kenarlara doğru gidildikçe doygunluk değeri arttığında koninin taban kenarlarında renklerin tam tonları oluşur. Merkez eksende %100 değerine sahip value değeri konin tepesine doğru azalarak gri tonda renklerin oluşmasını sağlar, koninin tam tepesinde (Value=%0) siyah renk oluşur. Şekil 1.29; HSV konik renk uzayını temsil etmektedir (URL-8, 2013).



Şekil 1.29: Konik HSV renk uzayında renk oluşumları

1.7.5. YCbCr (YCC) Renkler

Dijital komponent videolarda kullanılan renk uzayıdır (URL-8, 2013). Y (Luminance) parlaklık, Cb mavi kroma ve Cr ise kırmızı kroma bileşenlerini ifade etmektedir. Resimleri üç katmanda oluşturan YCbCr renk uzayı, görüntüyü daha az bit ile luminanca ve chrominance bileşenlerine ayırarak bellekten az alan kullanır. Dijital ortamda YCbCr renk uzayı ile ifade edilen görüntüler, RGB renk uzayında ifade edilen görüntülerden daha verimlidirler. Şekil 1.30 Y,Cb ve Cr değişkenlerine bağlı renk koordinatı verilmiştir (URL-10, 2013).



Şekil 1.30: YCbCr renk uzayı koordinat sistemi

1.7.6. CIE XYZ Renkler

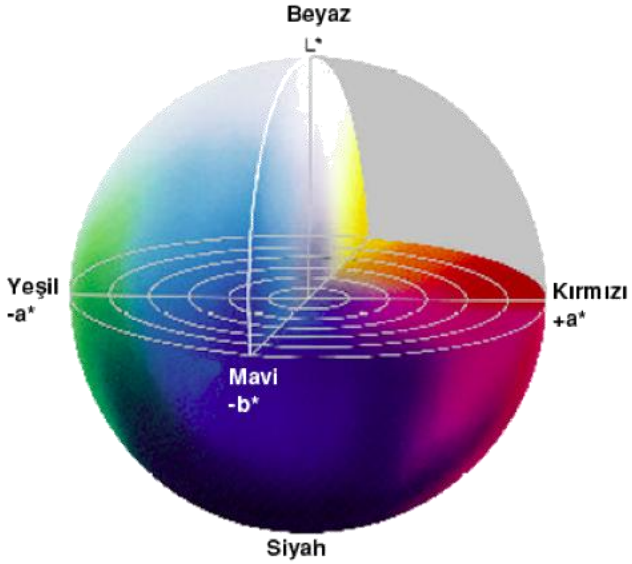
CIE'nin 1931 yılında tanımladığı ve diğer tüm renk uzaylarının temel mantığını oluşturduğu renk uzayıdır (Şahinbaşkan, 2013). XYZ renk uzayı üç renk bileşeni kullanılarak oluşturulmuştur. Sırasıyla X,Y,Z'e karşılık kırmızı, mavi ve sarı renklerdir. XYZ renk uzayı günümüzde renk elde etmekten çok, çalışma mantığı referans alınarak diğer renk uzaylarının oluşturulmasında kullanılır.

1.7.7. CIE L*a*b Renkler

CIE tarafından, renklerin insan tarafından algılanış şeklini referans alması ile oluşturulan renk uzayıdır. L*a*b renk uzayı aygıtlardan bağımsız renklerin nasıl göründüğü ile ilgili standartları tanımlar. Renk yönetim sistemleri L*a*b modelini, bir rengi bir renk uzayından diğerine, sonuçları önceden tahmin edilebilecek şekilde dönüştürmek için kullanır (URL-11, 2013).

L*a*b renk uzayı dikey sarı-mavi, yatayda yeşil kırmızı eksenlerine dayanan küresel bir yapı kullanır. L*a*b renk uzayının iyi dengelenmiş yapısı, bir rengin aynı zamanda

hem yeşil hem kırmızı veya hem mavi hem sarı olamayacağı teorisi üzerine inşa edilmiştir (Şahinbaşkan, 2013). Bu açıdan L, a ve b değişkenleri renk uzayında L; parlaklık (Lightness), a; +a kırmızı, -a yeşil ve b; +b sarı, -b mavi eksenlerini ifade etmektedirler. Şekil 1.31 küresel L*a*b ren uzayı verilmiştir (URL-11, 2013).



Şekil 1.31: Küresel L*a*b renkleri ve eksen dağılımları

1.7.8. CIE L*u*v Renkler

İki boyutlu modelleme ve renk tolerans hesaplamaları dışında L*a*b renk uzayı ile tamamen aynıdır.

2. GÖRÜNTÜ İŞLEME TEKNOLOJİLERİ

Sayısal görüntü işlemede çok fazla işlem ihtiyacı olduğundan çok hızlı sistemlere ihtiyaç duyulmaktadır. Son yılların en önemli çalışma alanlarından biri durumuna gelen görüntü işleme, çok farklı teknolojiler geliştirilerek yapılmaktadır. Günümüz bilgisayarları teknolojik olarak günden güne gelişim gösterseler de seri işlem yapma mantıklarından dolayı görüntü işlemenin artan ihtiyaçları noktasında yetersiz kalmaktadırlar. Birbiriyle ilgili birkaç girdinin eş zamanlı işlenmesi gerektiği durumlarda seri programlama mantığı ile çalışan bilgisayarlar yetersiz kalmaktadır. Görüntü işlemenin artan işlem hacmini karşılayabilecek ve görüntü ile ilgili girdi bilgilerinin paralel olarak işleyebilecek teknolojiler günden güne artmakta ve gelişim göstermektedirler. Paralel hesaplama, bir hesaplama probleminin çözmek için birden fazla bilgisayar kaynaklarının aynı anda kullanılmasıdır (Altıntaş ve Turan, 2011).

Görüntü işleme teknolojileri probleme yaklaşım açılarına bağlı olarak farlı algoritmalar ile çözüm geliştirmeyi amaç edinmişlerdir. Görüntü işleme sistemleri günümüzde; görüntünün sayısal sinyali üzerinde işlem yapan donanımsal teknolojiler ile görüntü sayısal sinyalinin iki veya daha fazla boyutlu sayı matrisi üzerinde işlem yapan yazılımsal teknolojiler olarak sınıflandırmak mümkündür.

2.1.Donanımsal Teknolojiler

Görüntü verilerini sayısallaştırmadan, analog görüntünün sayısal sinyalini işleyen sistemlerdir. Donanımsal teknolojiler görüntü verilerini doğrusal olmayan bir teknolojiyle işler. Görüntülerde işlem hacmi çok fazla olduğundan, görüntü işlenirken paralel programlama teknolojileri tercih edilmektedir.

2.1.1. Bi-i (Biologically Inspired Cellular Vision System) Görü Sistemi

Çok yüksek hızda gerçek zamanlı işlem yapabilen, kompakt, bağımsız ve akıllı bir kameradır (Arslan ve Arık, 2010). Bu sistemler analog görüntünün sayısal sinyalini ACE16K işlemcisinde, hücresel sinir ağı yapısı ile paralel ve çok hızlı bir şekilde işleyebilmektedir. Görüntünün sayısal sinyali üzerinde görüntüyü analiz eden ACE16K işlemci-

sine ek olarak bu sistemde sayısal işaretin kendisini işleyen bir DSP (digital signal processor) mikroişlemci bulunmaktadır.

Bi-i görü sistemlerinde işlemcilerden biri görüntünün sayısal sinyali içerisindeki veri ile ilgilenirken diğer işlemcinin bu görüntünün sayısal sinyalini işlemesi ve bu iki işlemcini etkileşim içerisinde olması Bi-i görü sistemlerinde görüntü işleme başarısını arttırmaktadır.

2.1.2. EYE-RIS

Eş zamanlı görüntü işleme için Anafocus firması tarafından geliştirilmiştir. EYE-RIS görü sistemi, optik olarak görüntüyü elde eder, görüntüyü işler, gerekli bilgileri çıkarır ve sonuca göre karar verir (Karabiber, 2009). İnsanın vizuel sistemini makineye aktarmaya çalışır. EYE-RIS görü sistemi, görüntü verilerini paralel bir şekilde işler. Eye-RIS görü sistemleri, Bi-i görü sistemleri gibi analog görüntünün sayısal sinyalini paralel işleyen yapıda organize edilmiştir.

2.1.3. Nvidia CUDA

CUDA, Nvidia ‘nın GPU (grafik işlem birimi) gücünü kullanarak hesaplama performansında büyük ölçüde artışlara olanak veren paralel hesaplama mimarisidir (URL-12, 2013). CUDA teknolojisi, görüntü işleme için kullandıkları GPU işlemcisinin şuan için yüzlerce çekirdekli olması ile birlikte paralel hesaplamalar yapılabiliyor olması görüntü işleme teknolojileri arasında çok önemli bir konum kazanmasını sağlamıştır. CUDA teknolojisi ile birlikte CUDA teknolojisi ile birlikte GPU işlemciler ile matematiksel işlemlerde yapılabilir olmuştur. CUDA ile birlikte GPU’da matematiksel işlemlerin yapılabiliyor olması GPU’nun bilgisayar içerisinde ikinci bir merkezi işlemci gibi kullanılabilir olmasını sağlamıştır. CPU ve GPU ile birlikte bilgisayarlarda, rastgele bellek erişimleri ve ayrıştırma gibi seri hesaplamalarda CPU, video çözme, görüntü işleme, 3D grafikler ve simülasyon oluşturma işlemlerde ise GPU işlemcisini kullanan heterojen programlama mantığı oluşmuştur. Yazılımcılar C, C++ ve Fortran programlama dillerine yapacakları CUDA eklentisi ile kolaylıkla paralel hesaplama yapan programlar geliştirebilirler. CUDA programlama, görüntü işleme için yazılımsal algoritmalar üretmekten çok görüntü verilerinin çok çekirdekli işlemcilerde paralel işlenebilmesi için bir mimari sağlar.

2.2.Yazılım Tabanlı Teknolojiler

Görüntüyü donanımsal işleyen teknolojilerin yanında, uyarlanabilir donanımlar ile bilgisayar ortamına sayısallaştırılarak alınan görüntüler çeşitli yazılımlar ile işlenebilmektedir. Görüntü işlemenin işlem hacminin büyük olması ve zorunlu eş zamanlı görüntü işleme uygulamalarından dolayı yazılım teknolojileri de paralel çözümleme mantığını esas alır. Bu amaçla farklı yapay zekâ teknikleri kullanılarak çok farklı yapılarda organize edilmiş hücresel algoritmalar geliştirilmiştir.

Görüntü işlemenin çok önem kazandığı günümüzde hemen hemen bütün programlama dilleri görüntü işlemek için araçlar sunmakta ve yine bu amaçla görüntü işleme kütüphaneleri geliştirilmektedir.

2.2.1. OpenCV

OpenCV (Open Source Computer Vision Library) Windows, Linux, Mac OS X, PSP (PlayStation Portable) platformları üzerinde çalışabilen, C diliyle yazılmış, gerçek zamanlı bilgisayarla görme (real time computer vision) ve görüntü işleme (image processing) uygulamaları için kullanılabilen, açık kaynak kodlu bir kütüphanedir. Intel tarafından geliştirilmiştir, Willow Garage tarafından desteklenmektedir (Bradski and Kaehler, 2008).

Optimize edilmiş C dili ile yazılan OpenCV; hesaplama verimliliği ve eş zamanlı uygulamalarda güçlü odaklanma için tasarlanmıştır ve değişen işlemci donanım teknolojileriyle uyumluluk gösterir. CUDA tarafından, verileri paralel hesaplayabilmek geliştirilen çok çekirdekli GPU işlemcilerini kullanabilir.

OpenCV, ücretsiz kütüphanelerin yanında, ayrı olarak işlemci şirketlerinin ürünlerinde daha güçlü optimizasyon sağlamak için geliştirdikleri kütüphane paketleri satın alınıp kullanılabilir. Intel şirketi bu amaçla işlemcileri için **I**ntegrated **P**erformance **P**rimitives (IPP) kütüphanelerini geliştirmiştir.

2.2.2. EmguCV

OpenCV kütüphanelerinin ve fonksiyonlarının framework'ün .Net dillerinde kullanılabilmesi için oluşturulmuş bir sarmalayıcıdır (ara kütüphane). Bu sayede OpenCV'nin bütün kütüphaneleri ve fonksiyonları C#, Visual Basic, IronPython gibi framework plat-

formunda çalışan derleyiciler tarafından kullanılabilir. EmguCV mono olarak derlendiğinden Windows, Linux, Mac OSX, Ipad, Iphone, Android işletim sistemlerinde çalıştırılabilir.

Mono; farklı işletim sistemlerinin program geliştirme bileşenlerini tek bir platformda toplayan ve farklı bir işletim sisteminde geliştirilen bir programın mono kapsamındaki (Windows, Linux, Mac OSX, IOS, Android, BSD, Solaris) bir başka işletim sisteminde çalışmasını olanaklı kılan bir program geliştirme platformudur (URL-17, 2013).

3. OPENCV

Açık kaynak kodlu bilgisayarla görme kütüphanesidir. Donanımsal geliştirilen görüntü işleme sistemleri için optimize edilmiş algoritmik çözümler sunar. OpenCV hareketli ya da hareketsiz görüntüler üzerinde işlemler yapmak için hazırlanmış kütüphanelerdir. Intel görüntü işleme laboratuvarlarında geliştirilmiş ve hız açısından optimize edilmiş fonksiyonlardan oluşmaktadır.

Günümüzde görüntü işleme çok geniş bir kullanım yelpazesine sahip olduğu için OpenCV'de hazırlanmış fonksiyonlar kullanılacak sistemin donanımsal yapısına ve kullanım amaçlarına göre farklı kütüphanelerde tanımlanmışlardır. OpenCV görüntü işleme için geliştirdiği fonksiyonları beş kütüphanede tanımlamıştır.

Hızla ilerleyen bilgisayarla görme teknolojisi (OpenCV kütüphaneleri) sayesinde, hem bilgisayarla görme uygulamaları kolaylaşmış ve hem de bu alanda çalışma yapan programcı sayısı artmıştır (Bradski ve Kaehler, 2008). OpenCV, 500'den fazla fonksiyonla bilgisayarda görüntü işleme alt yapısını basitleştirerek bilgisayarlı görü alt yapısının kullanıldığı birçok sistemin kolaylıkla geliştirilebilmesini sağlamıştır. OpenCV kütüphaneleri ile aşağıda tanımlı uygulamalar kolaylıkla geliştirilebilir (Bradski and Kaehler, 2008).

- İnsan-Bilgisayar Etkileşimi (Human-Computer Interaction – HCI).
- Nesne Kimliklendirme, Bölümleme ve Tanıma (Object Identification, Segmentation and Recognition).
- Yüz Tanıma (Face Recognition).
- İşaret Dili Tanıma (Gesture Recognition).
- Hareket Yakalama, Algılama ve Takibi (Motion Tracking, Ego Motion, Motion Understanding).
- Çiftli ve Çoklu Kamera Kalibrasyonu ve Derinlik Hesaplama (Stereo and Multi-Camera Calibration and Depth Computation).
- Kamera kalibrasyonu.
- Üç boyutlu görme.
- Medikal resim işleme.
- Hareketli Robot Teknolojileri.

OpenCV, kendi içinde sunulan açık kaynak kodlu kütüphanelerin yanında işlemci veya ekran kartı üreticileri ürünlerinden optimum performans elde etmek için ayrıca hazır-

lanan kütüphane dosyaları da eklenerek kullanılır. Genelde bu kütüphaneler kullanıcılara belli bir ücret karşılığı satılır. Intel bu amaçla IPP (Intel Performance Primitives) kütüphanelerini sunar. IPP, Intel şirketinin ürünlerinde çoklu parçacık verileri ve multimedia işlevleri için hazırladığı ücretli kütüphane dosyalarıdır (URL-13, 2013).

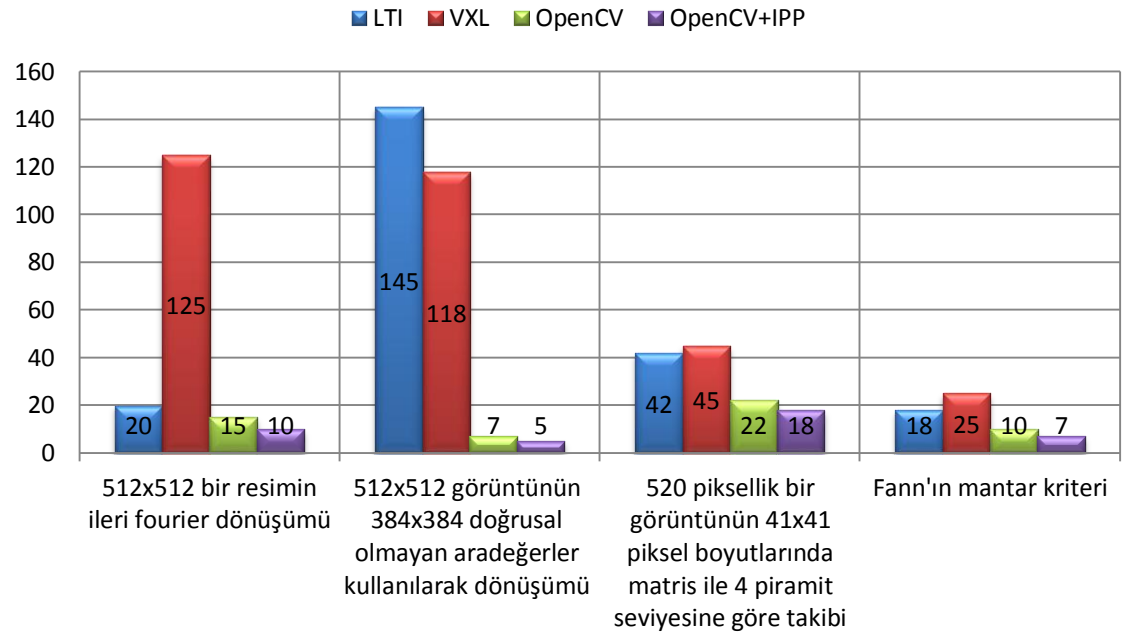
IPP, birçok programcının işbirliği sayesinde probleme yönelik geliştirilen yüksek optimizeli algoritmalar ile OpenCV'nin işlem çözme yeteneğini çok daha hızlandırır (Bradski and Kaehler, 2013). IPP kütüphanelerinin yüksek optimizasyonlu algoritmaları OpenCV'nin çok yüksek hızlarda problemleri çözmesine olanak sağlar.

IPP'nin yanı sıra VXL ve LTI kütüphaneleri de bilgisayarla görüde yüksek hızlarda işlem yapmayı olanaklı kılar. VXL; bilgisayarlı görme ile ilgili işlevsel birkaç algoritma sunar (URL-14,2013). VXL kütüphaneleri işlevlerine göre gruplara ayrılmışlardır.

- VGL (Vision Geometry Library); Geometrik görü kütüphanesi
- VNL (Vision Numerics Library); Rakamsal görü kütüphaneleri
- VIL (Vision Image Processing Library); Resim işleme görü kütüphaneleri
- VSL (Vision Streaming Library); Akış görü kütüphaneleri

Bunlar VXL kütüphanelerinden bazılarıdır. LTI (Luminance Transient Improvement) bir başka nesne yönelimli kütüphanedir. Donanım veya işletim sistemleri değişenleriyle uğraşmak zorunda kalmaksızın birimler arası senkronizasyon, seri port erişim, çoklu işlem sınıfları gibi işlemlerin yapılmasını olanaklı kılan kütüphanedir (URL-14, 2013). Ayrıntıları ortaya çıkaran, küçük geçişleri işleyen ve nesneler arasında ki ayrımı iyileştiren görüntü işleme kütüphanesidir (URL-15, 2013).

Şekil 3.1: Aynı platformda ve sırasıyla aynı örneklere uygulanan görüntü işleme kütüphanelerinin birbirlerine göre başarım grafikleri verilmiştir (grafik üzerinde yaklaşık değerler verilmiştir.) (URL-14, 2013)



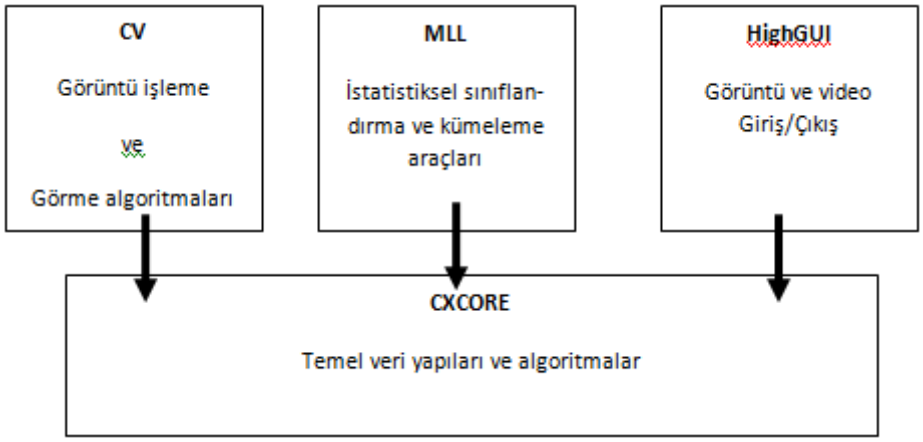
Şekil 3.1: Görüntü işleme kütüphanelerinin aynı örnekler üzerinde başarı grafikleri

İlk deney sonucunda oluşan değerler incelendiğinde; OpenCV'nin IPP ile birlikte en yakın LTI kütüphanelerinin yaklaşık 2 katı hızlı sonuç ürettiği görülmektedir. Görüntü sıkıştırma işleminde OpenCV ve OpenCV+IPP'nin diğer kütüphanelerden çok daha verimli çalıştıkları görülmektedir. Diğer iki deney setinde de OpenCV ve OpenCV+IPP'nin benzerlerinden avantajlı oldukları görülmektedir.

OpenCV, açık kaynak kodlu yapısından dolayı görüntü işlemede büyük programcı toplulukları kurulmuş ve bu sayede sürekli güncellenme imkânı sağlanmıştır. Bu durum OpenCV'nin bilgisayar dünyasında görüntü işlemede büyüyen bir platform olmasını sağlamıştır. Fonksiyonlar Intel mimarisi işlemcilerde daha iyi sonuç almak için optimize edilmiş MMX teknolojisi kullanılmıştır (Bradski and Kaehler, 2008). OpenCV platform bağımsız yapısından dolayı hemen hemen bütün programlama dillerine uyarlanabilir.

3.1.OpenCV Kütüphaneleri

OpenCV görüntü işleme fonksiyonlarını beş ana kütüphanede tanımlamıştır. Bunların dört tanesi Şekil 3.2 de verildiği gibi CV, MLL, HighGUI ve CXCORE birbirlerini tamamlar yapıdadır. Şekilde verilmeyen CXAux ise bu kütüphanelerden elde edilen çıktı verileri üzerinde çeşitli deneysel işlemler yaparak sistemlerin uygun tepkiler vermesi için çözümler sunan kütüphanedir.



Şekil 3.2: OpenCV kütüphane organizasyonları

3.1.1. CV Bileşeni

Dışarıdan okunmuş hareketli ve hareketsiz görüntü verileri üzerinde yüksek seviyeli görüntü işleme algoritmaları sunan temel kütüphanedir. CV kütüphanesi ile görüntü üzerinde temel operasyonlar, yapısal analiz, hareket analizi ve nesne takibi gibi görüntü verilerini işleyen temel fonksiyon gruplarından oluşur. CV kütüphanesinde tanımlanmış fonksiyonlardan bazıları aşağıdaki verilmiştir (URL-16, 2013).

- Resim İşleme
  - Gradientler, Kenarlar ve Köşeler (Sobel, Laplace, Canny).
  - Morfolojik işlemler (CreateStructuringElementEx, Erode, Dilate).
  - Filtre ve Renk Dönüşümleri (Smooth, Filter2D, CvtColor).
  - Görüntü Bölütleme (CvConnectedComp, FloodFill, FindContours).
  - Histogramlar (CvHistogram, CreateHist, ReleaseHist, ClearHist).
- Yapısal Analiz
  - Kontur İşleme (ApproxChains, ReadChainPoint, ApproxPoly).
  - Hesaplamalı Geometri (MaxRect, CvBox2D, BoxPoints, PointPolygonTest).
  - Düzlemsel Alt Bölümler (CvSubdiv2D, CvQuadEdge2D, CvSubdiv2DPoint, Subdiv2DLocate).
- Hareket Analizi ve Nesne Takibi
  - Nesne Takip (MeanShift, CamShift, SnakeImage).
  - Optik Akış (CalcOpticalFlowHS, CalcOpticalFlowLK, CalcOpticalFlowBM).

- Hareket Şablonları (UpdateMotionHistory, CalcMotionGradient, CalcGlobalOrientation, SegmentMotion).

- Örüntü Tanıma
  - Nesne Algılama (CvHaarFeature, CvHaarClassifier, CvHaarStageClassifier, CvHaarClassifierCascade).
- Kamera Kalibrasyonu ve 3D yeniden yapılandırma
  - Kamera Kalibrasyonu (ProjectPoints2, FindHomography, CalibrateCamera2, Rodrigues2, Undistort2).
  - Tahmin Poz (CreatePOSITObjet, ReleasePOTSITObject, CalcImageHomography).

Yukarıda verilen fonksiyonlar, CV kütüphanesi fonksiyon gruplarından sadece bazılarıdır.

3.1.2. MLL Bileşeni

CV kütüphanesindeki fonksiyonlar ile işlenmiş görüntüden bilgisayar ve diğer makinelerin çıkarımlarda bulunmasını sağlayan, temel fonksiyon ve araçların tanımlandığı kütüphanedir. MLL kütüphanesi sahip olduğu fonksiyonlar ve araçları ile verileri sınıflandırma ve istatiksel çıkarımlarda bulunma gibi görüntü işlemenin nihai hedefi olan kararlar için uygun veri analizleri yapar. MLL kütüphanesi aşağıda verilen fonksiyonları tanımlar;

- Ortak sınıflar ve fonksiyonlar; makine öğrenmesi, istatistiksel modeller için temel snıf (CvStatModel)
- Normal Bayes sınıflandırıcılar; normal dağılım verileri için Bayes sınıflandırıcı (CvNormalBayesClassifier)
- K'nın en yakın komşuları modeli (CvKNearest)
- Vektörel makinelerin desteği (CvSVM, CvSVMParams)
- Karar verme ağaçları (CvDTreeSplit, CvDTreeNode, CvDTreeParams, CvDTreeTrainData, CvDTree)
- Artırıcılar (CvBoostParams, CvBoostTree)
- Rastgele ağaçlar (CvRTParams, CvRTrees)
- Beklenti Maksimizasyonu (CvEMParams, CvEM)
- Yapay sinir ağları (CvANN_MLP_TrainParams, CvANN_MLP)

3.1.3. HighGUI Bileşeni

Kullanıcı arayüzleri için form verilerini oluşturmakla birlikte görüntü verilerinin giriş çıkış aygıtlarından yüklenip kaydedilmesini sağlayan giriş çıkış (I/O) fonksiyonlarını içeren kütüphanedir. HighGUI kütüphanesi aşağıdaki fonksiyon sınıflarını tanımlar;

- Basit arayüz fonksiyonları (cvNamedWindow, cvDetroyWindow, cvResizeWindow, cvShowImage gibi)
- Görüntü yükleme ve kaydetme fonksiyonları (cvLoadImage, cvSaveImage)
- Video giriş/çıkış fonksiyonları (cvCapture, cvCreateFileCapture, cvCreateCameraCapture, cvGrabFrame, cvRetrieveFrame, cvCreateVideoWriter gibi)
- Yardımcı sistem fonksiyonları (cvInitSystem, cvConvertImage,)

3.1.4. CXCore Bileşeni

Görüntü işleme için çeşitli veri yapılarının tanımlandığı ve xml desteği sunan kütüphanedir. CXCore görüntü verileri için aşağıdaki yapıları tanımlar;

- Temel yapıları (CvPoint, CvPoint2D32f, CvSize, CvRect, CvScalar, CvMatIpIImage, CvArr),
- Dinamik yapıları (Diziler: CvSeq, CvSlice vs. Grafikler; CvGraph, CreateGraph vs. Ağaçlar; CvTreeNodeIterator, InitTreeNodeIterator, NextTreeNode vs.),
- Diziler üzerinde işlemleri (Kopyalama ve Dolum; Copy, Set, Range. Dönüşümler; Reshape, Repeat, Split Flip dizi elemanlarına erişme, matematiksel işlemler),
- Çizim (Eğriler; CV_RGB, Line, Rectangle vs. ve Metin; InitFont, PutText, GetTextSize) fonksiyonları
- Veri tipi oluşturma (Dosyalama; CvFile Storage, CvFileNode, CvAttrList vs. Yazma; StartWriteStruct, EndWriteStruct, WriteInt vs. Veri Okuma; GetRootFileNode, GetFileNode, ReadInt, ReadReal vs.)
- Hata işleme ve sistem fonksiyonları (GetErrStatus, SetErrStatus, Error, ErrorStr s.)

3.1.5. CXAux Bileşeni

Şekil eşleştirme, yüz tanıma, hareket tanıma, vucüt hareketlerini tanıma gibi pek çok deneysel algoritmanın tanımlandığı standart kütüphanedir. CXAux ile aşağıdaki fonksiyon sınıflarından bazıları tanımlanmıştır.

- Üç boyutlu yazışma fonksiyonları
- Biçimsel fonksiyonlar (cvPreWarpImage, cvMakeScanlines, cvFindRuns, cvDynamicCorrespondMulti, cvMakeAlphaScanlines, cvDeleteMoire vs.)
- 3D Takip fonksiyonlar (cv3dTrackerCalibrateCameras,cv3dTrackerLocateObjects)
- Nesne nitelik fonksiyonları (cvCalcCovarMatrix, cvSVD, cvGEMM cvCalcEigenObjects, cvEigenProjection,)
- Gizli Markov Modelleri (CvImgObsInfo, cvCreate2DHMM, cvRelease2DHMM)

Verilen fonksiyonlar, CXAux kütüphanesinde tanımlı fonksiyonlardan bazılarıdır.

3.2.OpenCV Yapıları Kullanım Kuralları

OpenCV ile programlar geliştirilirken fontların kullanımında, yapıların isimlendirilmesinde ve fonksiyon tanımlamada belli kurallara bağlı kalınır.

3.2.1. Metin Kullanım Kuraları

OpenCV'de metinler kullanım amacına göre farklı şekillerde kullanılır.

Tüm karakterler büyük; OpenCV yapıları ve bayrakları için sabit tanımlama şeklidir. Örneğin; CV_SEQ_KIND_GRAPH.

Büyük küçük karakter; bu karışık yapı CvContourTree; gibi yapı isimlerinde ve fonksiyon isimlerinde kullanılır. Örneğin; cvFindContours();.

Tüm karakterler küçük; değişken ve argümanların tanımlanmasında kullanılır. Örneğin; *value,* int, *src*.

3.2.2. İsimlendirme Kuralları

OpenCV, yapılarını aşağıdaki tanımlama kurallarına göre çağırır. Yapılar karakter kullanım şekillerine, "cv" öneki kullanımına göre birbirlerinden ayırt edilebilirler.

Sabit Tanımlama; sabitlerde bütün karakterler büyük harfle tanımlanır. “ CV” öneki büyük karakterlerle tanımlanır. CV_SEQ_KIND_GRAPH.

Fonksiyon Tanımlama; OpenCV’de kullanılan bütün fonksiyonlar “cv” önekiyle başlar. cvFindContours();.

Yapı Tanımlama; OpenCV’de bütün yapılar “Cv” önekiyle oluşturulmuştur. CvContourTree;.

3.2.3. Fonksiyon Oluşturma Kuralları

OpenCV’deki fonksiyon isimleri standart olarak “cv” öneki ile başlar ve aşağıdaki standart formattadır.

cv<Eylem><Hedef><Mod>();

Eylem; fonksiyonun temel işlevini ifade eden kelimelerdir. Örneğin; -Set-, -Convert-, -Create- gibi.

Hedef; görüntü işlemede işlevin hedefini belirtir. Örneğin –FindCounters-, -ApproxPoly- gibi. Hedef her zaman tek bir ifade olmak zorunda değildir. Bir hedefi ifade etmek için bazen birden fazla kelime kullanımı gerekir. Örneğin; cvUnDistort(); gibi.

Mod; isteğe bağlı bir alandır. Eylemin çekirdek işlevinde bir değişiklik olması gerektiği durumlarda fonksiyon kullanılırken bu çekirdek fonksiyonun nasıl olması gerektiğini tanımlamak gerektiği durumlarda kullanılır. Örneğin. cvFindExtrinsicCameraParams_64d, burada 64d dış kamerada arama işlemini 64d parametresine göre yapılacağını belirtir.

4. EMGUCV ile OPENCV UYGULAMALARI

Studio.Net dilleri ile OpenCV kütüphanelerini kullanabilmek için C# dilinde yazılmış ara kütüphanelerden oluşan çapraz bir platformdur. EmguCV; OpenCV ile C#'ın güçlü yönlerini birleştirerek sistem kaynaklarının daha etkili bir şekilde kullanılmasına olanak tanır. EmguCV, OpenCV'nin optimize edilmiş etkili fonksiyonlarına ek olarak aşağıdaki avantajları da sağlar (URL-18, 2013).

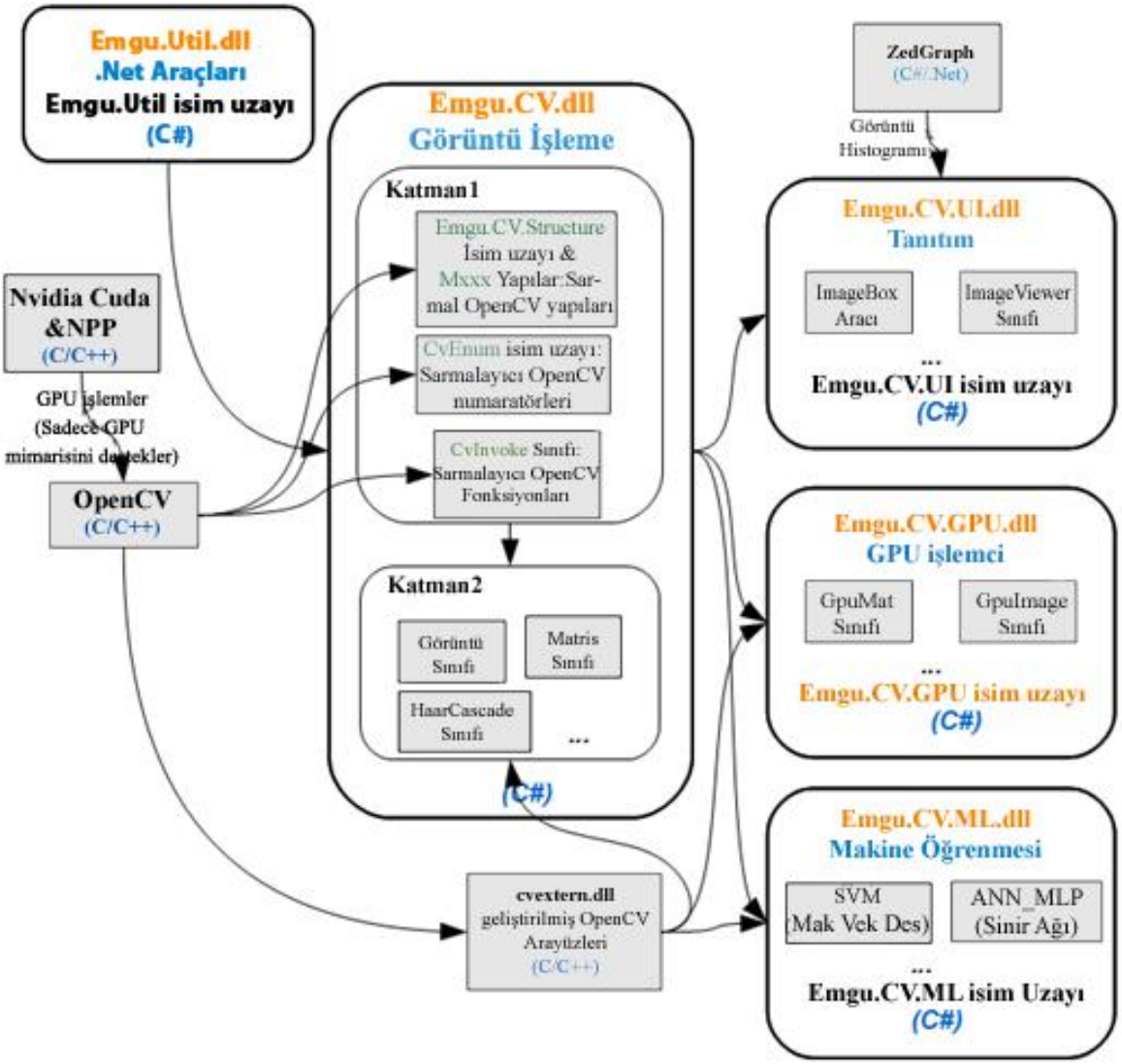
- Genel renk ve derinliği ile görüntü sınıfı
- Otomatik çöp toplama
- XML ile serileştirilebilir görüntü
- Akıllı XML dökümantasyon desteği
- Görüntü ve pikselleri üzerinde genel işlem desteği
- Resim sınıflarına veya direk OpenCV fonksiyonlarına ulaşma seçeneği

4.1. EmguCV Mimarisi

EmguCV iki katmanlı (layer) bir yapı sunar.

- Layer1; OpenCV'nin doğrudan erişilebilen fonksiyon, yapı ve numaralandırıcıları
- Layer2; Studio.Net platformunun sunduğu sınıflar

Şekil 4.1 EmguCV, OpenCV kütüphanelerini sarmalamıştır (URL-18, 2013).



Şekil 4.1: EmguCV-OpenCV ilişkisi

4.2. EmguCV Haritalama

EmguCv ile OpenCV kütüphanelerine ara sarmalayıcı kütüphaneler kullanılarak erişilebilir.

4.2.1. Fonksiyon Haritalama

Studio.Net dilleri içerisinde OpenCV fonksiyonlarına doğrudan ulaşmak için CvInvoke ara kütüphanesi kullanılır. CvInvoke sınıfındaki her fonksiyon OpenCV'de tanımlanmış aynı isimli fonksiyona karşılık gelir. Aşağıda C dilinde yazılmış OpenCV kod bloğu ile .NET tabanlı C# dillinde yazılmış EmguCV kod blokları karşılaştırılmıştır. EmguCV

görüntü işleme fonksiyonları CvInvoke sınıfından OpenCV fonksiyonları olarak kullanılmaktadırlar. CvInvoke ile fonksiyonlara erişim aşağıdaki gibi bir haritalama gerektirir.

Emgu.CV.CvInvoke.*fonksiyon_adı*

.Net Dillerinde (EmguCV) Fonksiyonlar	**C/C++ Dilinde (OpenCV) Fonksiyonları**
Emgu.CV.CvInvoke.cvNamedWindow();	cvNamedWindow();
Emgu.CV.CvInvoke.cvShowImage();	cvShowImage();
Emgu.CV.CvInvoke.cvSmooth();	cvSmooth();
Emgu.CV.CvInvoke.cvSobel();	cvSobel();

4.2.2. Yapı Haritalama

EmguCV yapıları OpenCV yapılarının doğrudan haritalanmış halleridir. EmguCV'de bir yapı ile çalışabilmek için bir yapı yolu haritası gerekliyken OpenCV'de yapı ismi ile direk erişim mümkündür. EmguCV'de yapılara erişim Structure sarmalayıcı kütüphanesi aracılığıyla gerçekleşir. Yapılara erişim aşağıdaki gibi bir haritalama gerektirir.

Emgu.CV.Structure.M*YapıAdı*

.Net Dilleri (EmguCV) Yapıları	**C/C++ Dilleri (OpenCV) Yapıları**
Emgu.CV.Structure.MCvBlob;	CvBlob;
Emgu.CV.Structure.MCvBox2D;	CvBox2D;
Emgu.CV.Structure.MCvMat;	CvMat;
Emgu.CV.Structure.MCvPoint2D64f;	CvPoint2D64f;
Emgu.CV.Structure.MCvKalman;	CvKalman;

4.2.3. Numaratör Haritalama

OpenCV'de görüntü işleme ile ilgili uygulamalarda özel ayarlamaların yapıldığı bayrak işaretlerine EmguCV'de CvEnum.XXX önbildirisi ile erişilir. Aşağıdaki gibi bir haritalama gerektirir.

Emgu.CV.CvEnum.Bayrak

.Net Dilleri (EmguCV) Bayrakları	**C/C++ Dilleri (OpenCV) Bayraklar**
Emgu.CV.CvEnum.ADAPTIVE_THRESHOLD_TYPE	ADAPTIVE_THRESHOLD_TYPE

Emgu.CV.CvEnum.BORDER_TYPE	BORDER_TYPE
Emgu.CV.CvEnum.COLOR_CONVERSION	COLOR_CONVERSION

4.3. Otomatik Çöp Toplama

Studio.Net dillerinin programlarda belleğin daha etkin kullanımı için geliştirilmiş bellek yönetimi yapısıdır. Programlar bu yapının içerisinde kodlanır program bitiminde yapı sonuna gelineceğinden programa ait veriler bellekten programlama dili tarafından otomatik olarak atılır.

Üzerinde çalışılan görüntü işaretçileri bu yapı sayesinde otomatik oluşturulur ve kullanımına izin verilir. Görüntü işleme uygulamalarında otomatik çöp toplayıcısı sayesinde bellek yönetimi programlama dili tarafından otomatik yapılır. Aşağıdaki gibi bir yapıda kullanılır.

```
using (Image<Bgr, Byte> res = new Image<Bgr,Byte>("Open Resim.jpg")){
        // Program buraya yazılır
    }   //Bellek üzerinde görüntü verilerinin temizlenmesi.
```

4.4. Dosyadan Resim Yükleme ve Görüntüleme

Optimize edilmiş OpenCV fonksiyonlarıyla herhangi bir şekilde görüntünün programa yüklenmesi ve görüntülenmesi oldukça basittir.

OpenCV fonksiyonlarını Studio.Net platformunda tanımlı programlama dillerinden herhangi biri ile kullanabilmek için OpenCV kütüphanelerine erişim imkânı sağlayan Emgu.CV ve diğer alt kütüphaneleri derleyici ön bildirisi olarak yüklemek gerekir.

```
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV.UI;
using System.Drawing;
```

C#'ta OpenCV ile görüntü işleyebilmek için yukarıda verilen EmguCV sarmalayıcı ön bildirilerine ek olarak C#'ın görüntü işleme sınıf ve fonksiyonlarına erişebilmek için System. Drawing kütüphanesinin de derleyiciye ön yüklenmesi gerekmektedir.

Geliştirilen Uygulama:

EmguCV ile sarmalanan OpenCV kütüphanelerine erişimi sağlayan isim uzayları derleyiciye yüklendikten sonra basit görüntü yükleme programı aşağıdaki kod parçacığında verilmiştir.

```
private void button1_Click(object sender, EventArgs e){    String resimYukle = "Yüklenen Resim"; //Yeni pencere adı
Emgu.CV.CvInvoke.cvNamedWindow(resimYukle); //Pencere adı atama
using (Emgu.CV.Image<Bgr, Byte> res = new Image<Bgr,Byte>("Open Resim.jpg")) //res değişkenine dosyadan resim yükleme
{ Emgu.CV.CvInvoke.cvShowImage(resimYukle, res.Ptr);// oluşturulan pencereye bellek-teki görüntünün yüklenmesi
Emgu.CV.CvInvoke.cvWaitKey(0); // Pencereyi kapatılana kadar sürekli açık
Emgu.CV.CvInvoke.cvDestroyWindow(resimYukle);// Pencereyi kapatma
}//Bellek üzerinde görüntü verilerinin temizlenmesi.}
```

Yukarıda verilen kodlar derlendiğinde sabit disk üzerinde bulunan "Open Resim.jpg" görüntüsü şekil 4.2'deki gibi pencere yüklenmiştir.

Şekil 4.2: EmguCV ile yüklenen dosya

4.5. Metin Yönetimi

OpenCV'de metin yönetimi CvFont() özel yapısı ile kullanılmaktadır. EmguCV'de, MCvFont() haritalamayla bu yapı kullanılabilmektedir. MCvFont ile görüntü üzerine eklenecek metin, özellikleri belirlendikten sonra bir görüntüymüş gibi işlemler uygulanır.

Geliştirilen Uygulama:

Geliştirilen aşağıdaki uygulama da MCvFont metin yapısı ile oluşturulan özelliklerde metin, görüntü yüklenirken görüntüye eklenmiştir.

```
private void button1_Click(object sender, EventArgs e){
String pencere = "Metin Yaz.";
CvInvoke.cvNamedWindow(pencere);
 using (Image<Bgr, Byte> res = new Image<Bgr, byte>("Resimler\\Lena.jpg")){
MCvFont f = new MCvFont(Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_DUPLEX, 1.0, 1.0);
res.Draw("Resim Yuklendi", ref f, new Point(Size.Height / 2, Size.Width / 2), new Bgr(200, 150, 100));
CvInvoke.cvShowImage(pencere, res.Ptr);
CvInvoke.cvWaitKey(0);
CvInvoke.cvDestroyWindow(pencere); } }
```

Yukarıda verilen kod parçacığı, program derlendiğinde şekil 4.3'te verilen sonucu sağlayacaktır.

Şekil 4.3: Yüklenen bir görüntü üzerine MCvFont yapısı ile görüntü ekleme

4.6. Histogram Grafiği

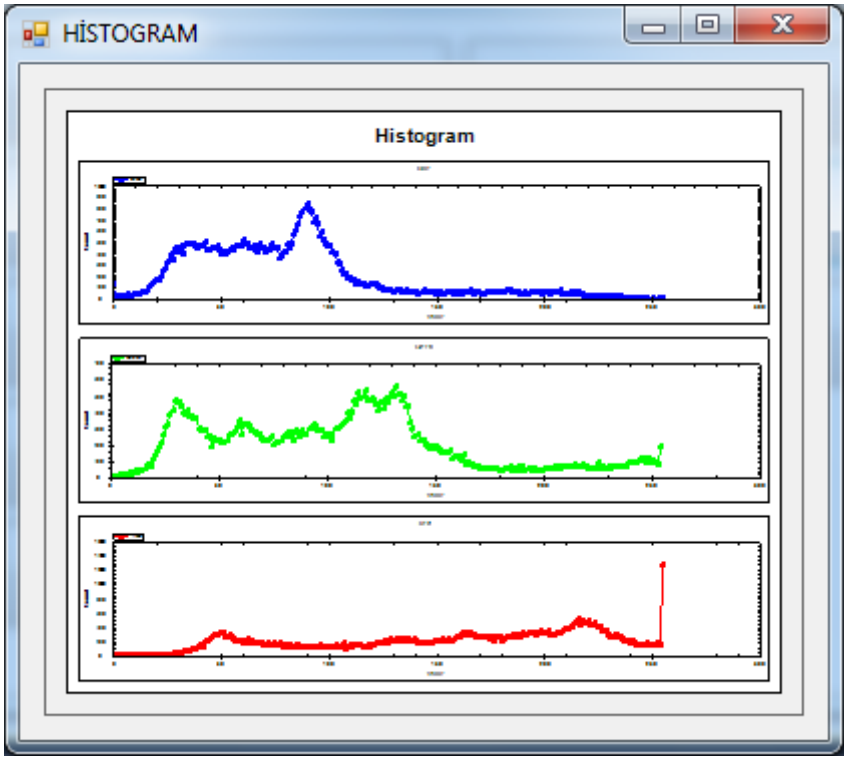
EmguCV ile görüntü histogramları oluşturmak ve bu grafikler ile çalışmak, geliştirilen histogramBox aracı ile son derece basitleştirilmiştir. EmguCV ile histogram grafiği, Emgu.CV.UI.dll kütüphanesi ile programa eklenen histogramBox aracıyla otomatikleştirilmiştir. Görüntü grafiğini oluşturmak için tek yapılması gereken görüntüyü programa yüklemek ve görüntü ile birlikte histogram değer aralığını parametre olarak histogramBox aracına aşağıdaki gibi girmek olacaktır.

Geliştirilen Uygulama:

Aşağıda histogramBox racı ile yüklenen renkli bir görüntüye ait histogram grafiği oluşturulmuştur.

```
private void radioButton2_CheckedChanged(object sender, EventArgs e){
using(Image<Bgr, Byte> resimm = new Image<Bgr, byte>("Resimler\\Lena.jpg")){
histogramBox1.ClearHistogram();
histogramBox1.GenerateHistograms(resimm, 256);
histogramBox1.Refresh();
}        }
```

Şekil 4.4: dosyadan yüklenen RGB tanımlanmış Lena.jpg resim dosyasının 0-255 piksel dağılım grafiği verilmiştir.



Şekil 4.4: RGB görüntü histogramı

4.7. Eşikleme

EmguCV, eşikleme için iki fonksiyon sağlar. cvThreshold(); girilen parametreler aracılığıyla klasik eşikleme yaparken, cvAdaptiveThreshold() ise parametreler ile uyarlanabilir Mean ve Gauss filtreleri yöntemlerine göre eşikleme yapar. EmguCV, CV_THRESH_BINARY ve CV_THRESH_BINARY_INV temel bayraklarının yanında çok farklı seçeneklerde görüntüler üzerinde eşikleme yapar.

4.7.1. Dizi Elemanları Sabit Seviyeli Eşikleme

EmguCV'nin belirlenen bir eşik değerinin altında kalan piksel değerlerini 0, üstünde kalan piksel değerlerini 1 olarak atadığı klasik eşikleme metodudur. cvThreshold(); fonksiyonu ile eşikleme yapar ve aşağıdaki yapıda kullanılır.

```
CvInvoke.cvThreshold(Kaynak,
                     Hedef,
                     Esik Deger,
                     Max Piksel,
                     Eşikleme Tipi);
```

Kaynak: Eşikleme yapılacak giriş görüntüsüdür.

Hedef: Giriş görüntüsü üzerinde eşikleme yapıldıktan sonra oluşacak hedef görüntüdür. Kaynak görüntü ile aynı boyutlarda ve renk uzayında olması gerekir.

Esik Deger: Görüntü piksellerinin karşılaştırılacağı sınır değerdir. 0-255 arasında bir değer olmalıdır.

Max Piksel: Esik değerinden etkilenecek maksimum piksel değerini ifade eder. Bu değerin altında kalan pikseller ile eşikleme yapılır.

Eşikleme Tipi: Gri seviye görüntülerin ikili görüntülere dönüşümü için kuralların tanımlandığı bayrakları ifade eder.

- **CV_THRESH_BINARY:** piksel>eşikdeger ise piksel = 0 değilse piksel = 1
- **CV_THRESH_BINARY_INV:** piksel>eşik ise piksel = 1 değilse piksel = 0
- **CV_THRESH_OTSU:** Görüntünün ortalama piksel değerine göre eşikleme yapar. Piksel>Ortalama ise piksel=0 değilse piksel = 1
- **CV_THRESH_TRUNC:** piksel>esikdeger ise piksel=piksel değil ise piksel = 1/0
- **CV_THRESH_TOZERO:** piksel>esikdeger ise piksel = 0 değil ise piksel = 1
- **CV_THRESH_TOZERO_INV:** piksel>esikdeger ise piksel = piksel değil ise piksel=0

Bu bayraklar tek kanallı gri seviye görüntüler için kullanılır.

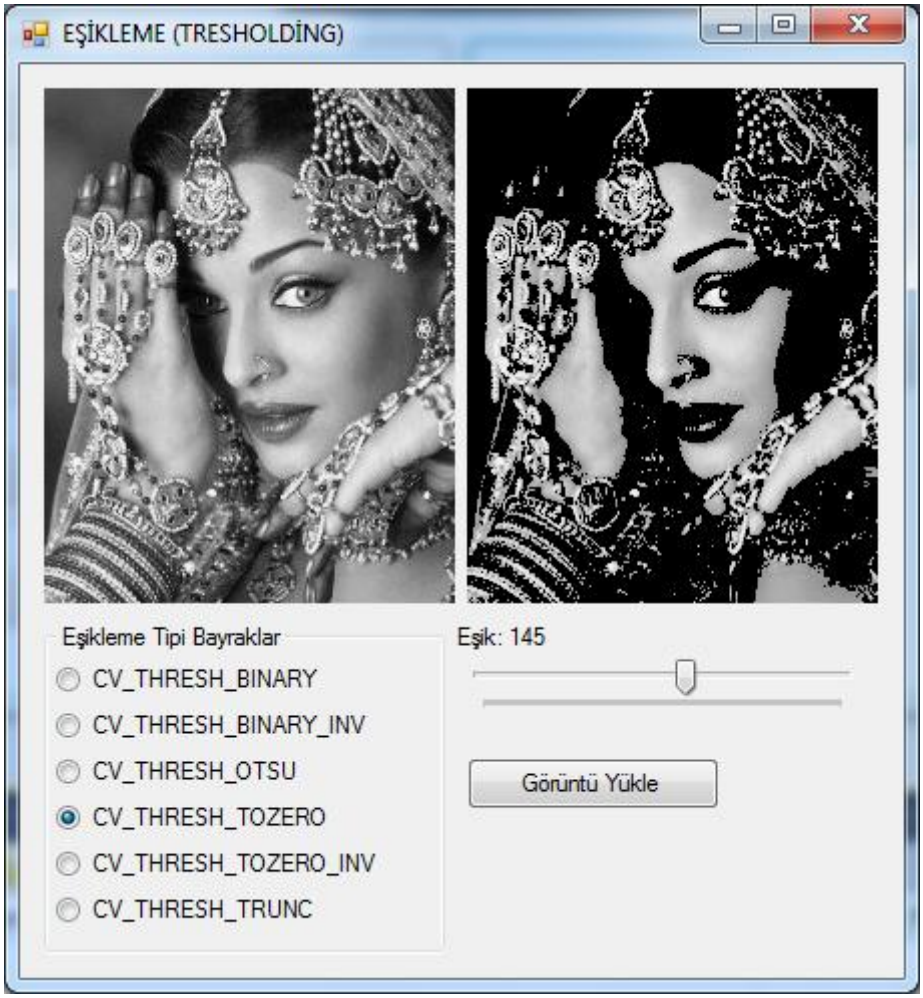
Geliştirilen Uygulama:

Verilen parametreler kullanılarak aşağıda EmguCV uygulaması geliştirilmiştir.

```
 private void trackBar1_Scroll(object sender, EventArgs e){
Image<Gray, Byte> kaynak = new Image<Gray, byte>("Resimler\\hindu.jpg");
ımageBox1.Image = kaynak;
Image<Gray, Byte> hedef = new Image<Gray, byte>(kaynak.Size);
int esik_deger = trackBar1.Value;
label4.Text = "Eşik: " + esik_deger.ToString();
if (radioButton4.Checked)    {
      CvInvoke.cvThreshold( kaynak,
                            hedef,
                            esik_deger,
                            255,
                            THRESH.CV_THRESH_BINARY);    }
  else if (radioButton5.Checked)    {
```

```
    CvInvoke.cvThreshold( kaynak,
                          hedef,
                          esik_deger,
                          255,
                          THRESH.CV_THRESH_BINARY_INV);    }
  else if (radioButton6.Checked)    {
    CvInvoke.cvThreshold( kaynak,
                          hedef,
                          esik_deger,
                          255,
                          THRESH.CV_THRESH_OTSU);    }
  else if (radioButton7.Checked)    {
    CvInvoke.cvThreshold( kaynak,
                          hedef,
                          esik_deger,
                          255,
                          THRESH.CV_THRESH_TOZERO);    }
  else if (radioButton8.Checked)    {
    CvInvoke.cvThreshold( kaynak,
                          hedef,
                          esik_deger,
                          255,
                          THRESH.CV_THRESH_TOZERO_INV);    }
  else if (radioButton9.Checked)    {
    CvInvoke.cvThreshold( kaynak,
                          hedef,
                          esik_deger,
                          255,
                          THRESH.CV_THRESH_TRUNC);    }
ımageBox3.Image = hedef; }
```

Kodları verilen uygulamanın CV_THRESH_TOZERO bayrağına ve 145 eşik değerine göre çıktısı Şekil 4.5'de verilmiştir.



Şekil 4.5: cvThreshold(); fonksiyonu ile sabit seviyeli eşikleme

4.7.2. Dizi Elemanları Uyarlanabilir Eşikleme

EmguCV, görüntü bölütleme ve sınıflandırma gibi durumlarda daha iyi sonuçlar alınabilmesi için tanımlanmış eşikleme yöntemidir. Eşikleme yapılırken görüntü sınırlarının belirlenmesinde daha iyi sonuçlar alabilmek için görüntü üzerindeki gürültülerin minimize edilmesini sağlayan Mean ve Gauss filtreleri kullanılır. Eşiklenecek görüntü belirlenen parametreler ile tercih edilen Mean veya Gauss filtrelerinden biri ile yumuşatılır. EmguCV, uyarlanabilir eşikleme için cvAdaptiveThreshold() fonksiyonunu aşağıdaki gibi tanımlar.

CvInvoke.cvAdaptiveThreshold(kaynak,
hedef,
eşik değer,
uyarlanabilir metot,
eşikleme tipi,

matris boyutu,
metot bağımlısı);

kaynak: Giriş görüntüsünü ifade eder.

hedef: Giriş görüntüsünün eşiklenmiş sonuç görüntüsüdür.

eşik değer: Görüntü piksellerinin karşılaştırılacağı sınır değerdir. 0-255 arasında bir değer olmalıdır.

uyarlanabilir metod: Farklı eşikleme seviyeleri için uyarlanabilir metotları ifade eder. EmguCV, CV_ADAPTIVE_THRESH_GAUSSIAN_C ve CV_ADAPTIVE_THRESH_MEAN_C bayrakları ile iki uyarlanabilir metot sunar.

eşikleme tipi: Gri seviye görüntülerin ikili görüntülere dönüşümü için kuralların tanımlandığı bayrakları ifade eder. Uyarlanabilir eşikleme yöntemi sadece CV_THRESH_BINARY ve CV_THRESH_BINARY_INV eşikleme tiplerine uyarlanabilir.

matris boyutu: Mean ve Gauss yumuşatma filtrelerinin konvülasyon matris boyutlarıdır. 3, 5, 7, 9, 11, … olabilir.

metod bağımlısı: Mean ve Gauss filtreleri ile maksimum olmayan noktaların bastırılması için eşik değeri sağlar. Bu değerin altında değere sahip pikseller zemin olarak kabul edilir.

Geliştirilen Uygulama:

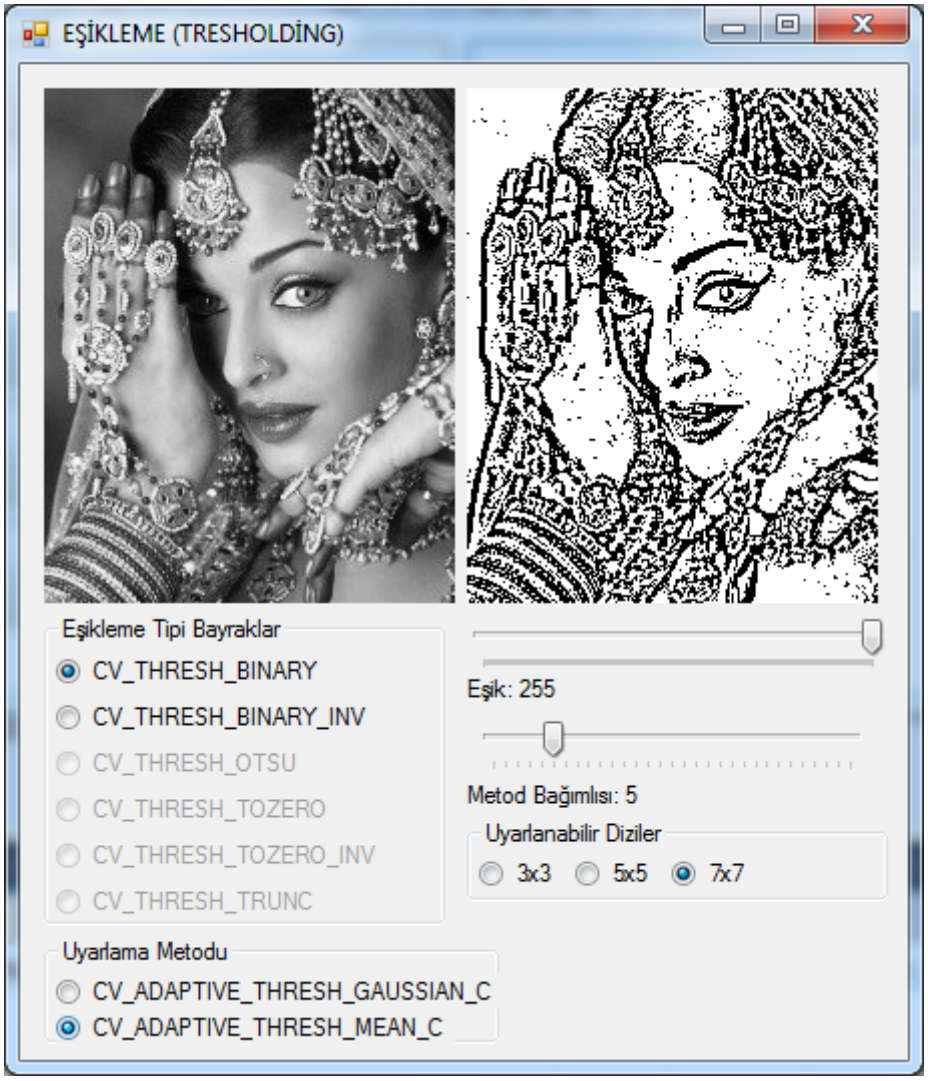
Tanımlanan parametrelere göre aşağıdaki uygulama geliştirilmiştir.

```
private void trackBar1_Scroll(object sender, EventArgs e){
Image<Gray, Byte> kaynak = new Image<Gray, byte>("Resimler\\hindu.jpg");
ımageBox1.Image = kaynak;
Image<Gray, Byte> hedef = new Image<Gray, byte>(kaynak.Size);
int esik_deger = trackBar1.Value;
int metod_bagımlisi=trackBar2.Value;
label4.Text = "Eşik: " + esik_deger.ToString();
label5.Text = "Metod Bağımlısı: " + metod_bagımlisi.ToString();
if (radioButton1.Checked)    {
if (radioButton10.Checked)       {
if (radioButton4.Checked)
CvInvoke.cvAdaptiveThreshold(
                    kaynak,
```

```
                    hedef,
                    esik_deger,
                    ADAPTI-
VE_THRESHOLD_TYPE.CV_ADAPTIVE_THRESH_GAUSSIAN_C,
                    THRESH.CV_THRESH_BINARY,
                    3,
                    metod_bagımlisi);
else if (radioButton5.Checked)
CvInvoke.cvAdaptiveThreshold(
                    kaynak,
                    hedef,
                    esik_deger,
                    ADAPTI-
VE_THRESHOLD_TYPE.CV_ADAPTIVE_THRESH_GAUSSIAN_C,
                    THRESH.CV_THRESH_BINARY_INV,
                    3,
                    metod_bagımlisi);       }
else if (radioButton11.Checked)       {
        if (radioButton4.Checked)
CvInvoke.cvAdaptiveThreshold(
                    kaynak,
                    hedef,
                    esik_deger,
                    ADAPTI-
VE_THRESHOLD_TYPE.CV_ADAPTIVE_THRESH_MEAN_C,
                    THRESH.CV_THRESH_BINARY,
                    3,
                    metod_bagımlisi);
        else if (radioButton5.Checked)
CvInvoke.cvAdaptiveThreshold(
                    kaynak,
                    hedef,
                    esik_deger,
```

```
                    ADAPTI-
VE_THRESHOLD_TYPE.CV_ADAPTIVE_THRESH_MEAN_C,
                    THRESH.CV_THRESH_BINARY_INV,
                    3,
                    metod_bagımlisi);       }    }
   else if (radioButton2.Checked)    {
      if (radioButton10.Checked)       {
         if (radioButton4.Checked)
CvInvoke.cvAdaptiveThreshold(
                    kaynak,
                    hedef,
                    esik_deger,
                    ADAPTI-
VE_THRESHOLD_TYPE.CV_ADAPTIVE_THRESH_GAUSSIAN_C,
                    THRESH.CV_THRESH_BINARY, 5,
                    metod_bagımlisi);
         else if (radioButton5.Checked)
… // Programın tamamı verilen örnek uygulamada bulunmaktadır.
   ımageBox3.Image = hedef;   }
```

Geliştirilen uygulamada yüklenen görüntü eşik 255, CV_THRESH_BINARY eşikleme tipine, CV_ADAPTIVE_THRESH_MEAN_C eşikleme metoduna, 7x7 matris boyutuna ve 5 metod bağımlısı değerine göre oluşan çıktı Şekil 4.6'da verilmiştir.



Şekil 4.6: cvAdaptiveThreshold(); fonksiyonu ile uyarlanabilir eşikleme

4.8. Aritmetik Mantık İşlemler

EmguCV ile görüntü verileri üzerinde aritmetik ya da mantıksal işlemler yapılabilir. Bu işlemler görüntü işleme basamaklarında daha iyi sonuçlar edinmek için genelde yardımcı işlemler olarak kullanılırlar.

Matematiksel niceliklerde olduğu gibi görüntü verileri birbirlerine eklenip çıkarılabilir veya görüntülerin piksel değerleri sayısal bir büyüklük ile aritmetiksel veya mantıksal işlem sonucu yeni bir piksel değeri elde edilebilir.

İki görüntü arasında aritmetiksel veya mantıksal işlem yapılırken, işlem yapılan görüntülerin ve aynı zamanda sonuç verisinin saklanacağı görüntü matrisinin aynı boyutlarda ve renk derinliklerinde olması zorunluluğu vardır.

4.8.1. Toplama

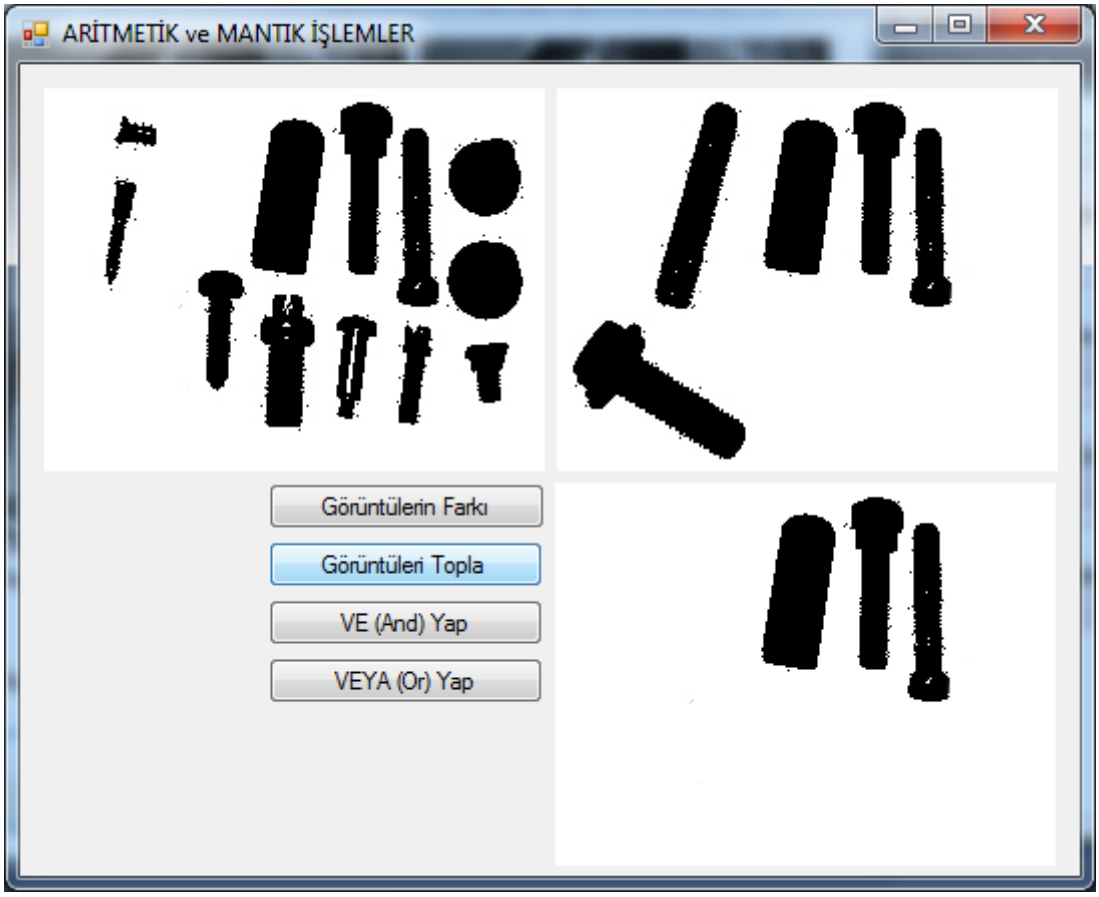
Görüntü verileri üzerinde yapılan aritmetiksel toplama işleminde, görüntü piksel değerlerinin niceliklerinin toplam sonucundan ziyade birbirleriyle toplanan görüntülerdeki ortak nesneler işlem sonucu olarak elde edilir.

Geliştirilen Uygulama:

Aşağıda yüklenen iki görüntünün birbirleriyle toplanması sonucu oluşan sonuç görüntüsü programı verilmiştir.

```
private void Toplama_Yap_Click(object sender, EventArgs e){
Image<Gray, Byte> esik1 = new Image<Gray, byte>("Resimler\\civatae4.jpg");
Image<Gray, Byte> esik2 = new Image<Gray, byte>("Resimler\\civatae3.jpg");
ımageBox1.Image = esik1;
ımageBox2.Image = esik2;
Image<Gray, Byte> toplam = esik1 + esik2;
ımageBox3.Image = toplam;  }
```

Aritmetiksel toplama işlemi ile görüntülerdeki ortak alanların tespiti yapılır. Yukarıda verilen programda binary yüklenmiş iki görüntünün toplamından tespit edilen ortak nesneler Şekil 4.7'de verilmiştir.



Şekil 4.7: İkilileştirilmiş iki görüntünün toplamı

4.8.2. Çıkarma

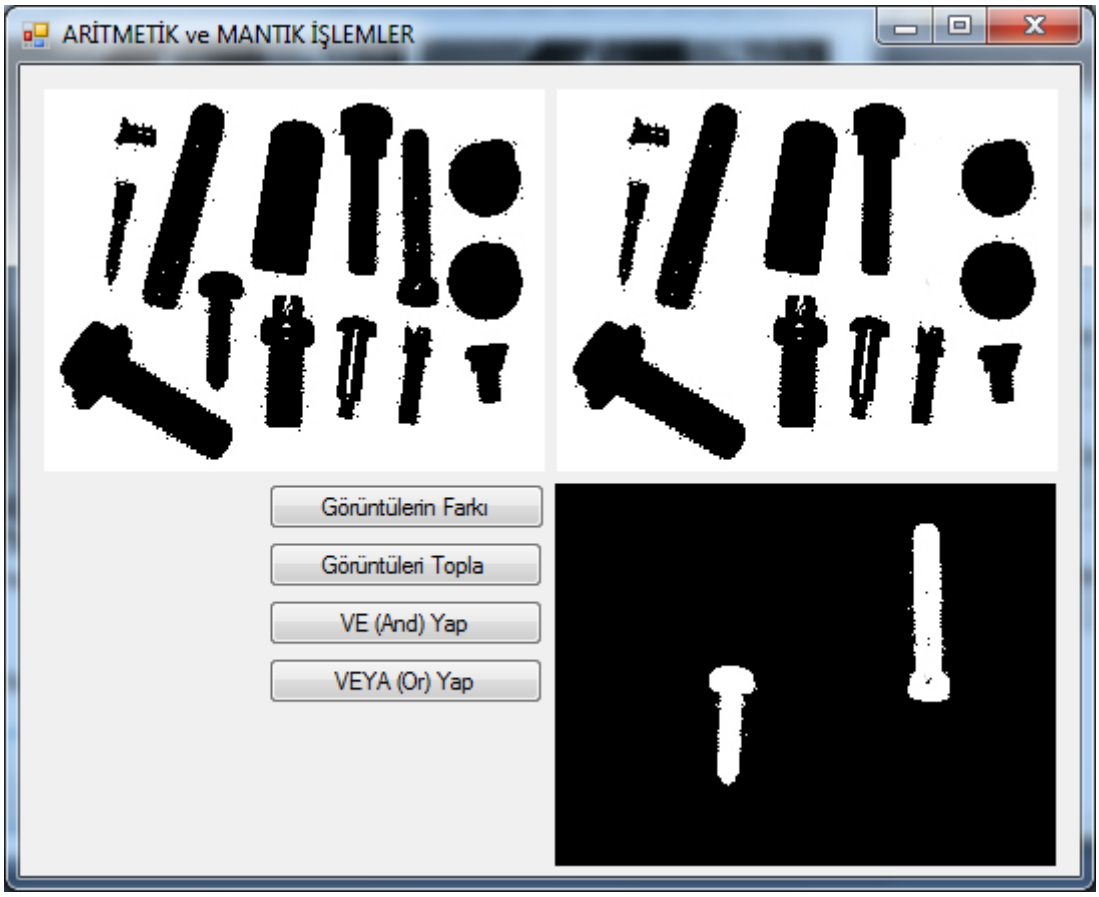
Görüntülerin aritmetiksel fark işlemlerinde, eksiltilen görüntünün eksilen görüntüden farklı nesneleri çıktı verisi olarak elde edilir. Görüntülerin fark alma işlemleri, görüntü işlemede daha çok, karmaşık bir görüntüden eksilen alanların tespiti için kullanılır.

Geliştirilen Uygulama:

Yüklenen birinci görüntüden ikinci görüntünün çıkarılması sonucu oluşacak sonuç görüntüsünü veren uygulama aşağıda verilmiştir.

```
private void Cıkarma_Yap_Click(object sender, EventArgs e){
Image<Gray, Byte> esik1 = new Image<Gray, byte>("Resimler\\civatae2.jpg");
Image<Gray, Byte> esik2 = new Image<Gray, byte>("Resimler\\civatae1.jpg");
ımageBox1.Image = esik1;
ımageBox2.Image = esik2;
Image<Gray, Byte> fark = esik1 - esik2;
ımageBox3.Image = fark;      }
```

Aritmetiksel fark işleminde yeni görüntüde olmayan nesneler ikinci görüntünün orijinal görüntüden farkı işlemi sonucu Şekil 4.8'de tespit edilmiştir.



Şekil 4.8: Aritmetiksel fark alma işlemi

4.8.3. Ve (And) İşlemi

Görüntülerin birbirleri ile mantıksal olarak ve'lenmesi, mantıksal 've' işlemine giren görüntülerin birbirlerine eklenmesi şeklindedir.

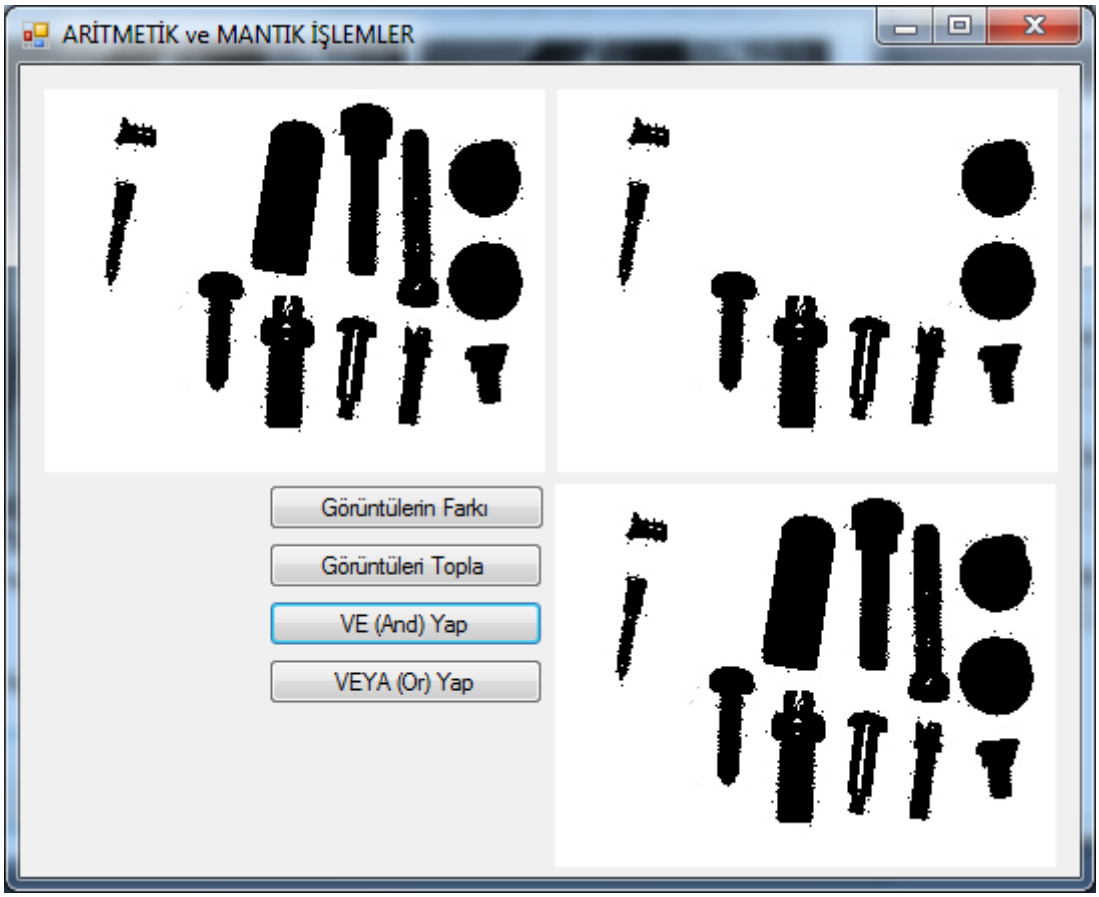
Geliştirilen Uygulama:

Yüklenen iki görüntünün birbirleriyle ve'lenmesi sonucu oluşacak veri aşağıdaki uygulamada verilmiştir.

```
private void Ve_İslemi_Yap_Click(object sender, EventArgs e){
Image<Gray, Byte> esik1 = new Image<Gray, byte>("Resimler\\civatae4.jpg");
Image<Gray, Byte> esik2 = new Image<Gray, byte>("Resimler\\civatae5.jpg");
ımageBox1.Image = esik1;
ımageBox2.Image = esik2;
Image<Gray, Byte> ve = esik1.And(esik2);
```

ımageBox3.Image = ve; }

Mantıksal ve işlemi sonucu görüntüler birbirlerine eklenir sonuç olarak iki görüntünün toplamından oluşan ortak görüntü Şekil 4.9'da verilmiştir.



Şekil 4.9: Mantıksal Ve (And) işlemi

4.8.4. Veya (OR) İşlemi

Mantıksal veya işlemi, işleme giren görüntülerin ortak alanlarını tespit eden aritmetiksel toplama işlemi ile aynı sonucu verir.

Geliştirilen Uygulama:

Veya işleminde ikinci görüntüde olan birinci görüntü alanları aşağıda geliştirilen uygulamada tespit edilmiştir.

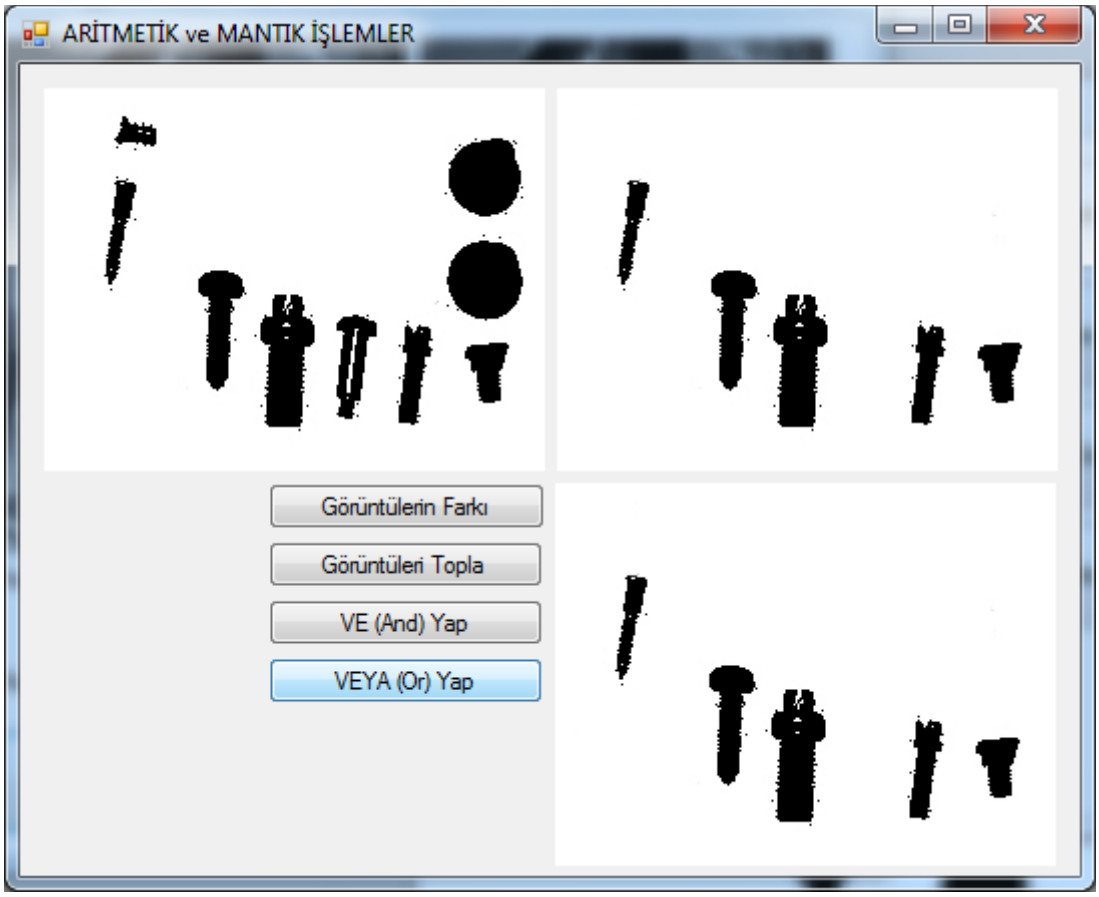
```
private void Veya_İslemi_Yap_Click(object sender, EventArgs e){
Image<Gray, Byte> esik1 = new Image<Gray, byte>("Resimler\\civatae5.jpg");
Image<Gray, Byte> esik2 = new Image<Gray, byte>("Resimler\\civatae6.jpg");
ımageBox1.Image = esik1;
```

ımageBox2.Image = esik2;

Image<Gray, Byte> veya = esik1.Or(esik2);

ımageBox3.Image = veya; }

İkinci görüntünün birinci görüntüyle eşleşen nesneleri Şekil 4.10'de verilmiştir.



Şekil 4.10: Mantıksal Veya (Or) işlemi

4.8.5. Değilleme (Not) İşlemi

İşlemsel olarak verilen görüntülerin piksel değerlerinin 255 değerine göre tümleyenini alan mantıksal işlemlerdendir. Görsel olarak koyu alanlarda gizlenmiş görüntü alanlarını belirlemek için kullanılan görüntü işleme tekniklerindendir.

Geliştirilen Uygulama:

Yüklenen görüntünün her piksel değerinin 255 değerine göre tümleyenini alıp her piksel değerini yeniden hesaplayan veya olumsuzlayan uygulama aşağıda geliştirilmiştir.

```
private void Renkli_radioButton1_CheckedChanged(object sender, EventArgs e){
Image<Bgr, Byte> resimm = new Image<Bgr, byte>("Resimler\\aletler.jpg");
ımageBox1.Image = resimm;
ımageBox3.Image = resimm.Not();}
private void Gri_radioButton2_CheckedChanged(object sender, EventArgs e){
Image<Gray, Byte> resimm = new Image<Gray, byte>("Resimler\\aletler.jpg");
ımageBox1.Image = resimm;
ımageBox3.Image = resimm.Not();}
```

Yüklenen görüntünün piksel değerleri, piksel[x,y] = 255 – piksel[x,y], ile her pikselin değeri yeniden hesaplandığında oluşan örnek görüntü Şekil 4.11'de verilmiştir.

Şekil 4.11: Mantıksal Değilleme (Not) işlemi

4.9. Kenar Çıkarma

Kenar belirleme, bütün görüntü işleme problemlerinin başlıca basamaklarından biridir ve sistemin nihai hedefine ulaşmasında başlıca etkendir. Hemen hemen bütün görüntü işleme uygulamalarında görüntü içerisinde belli bir alan ya da nesne ile ilgilenilir. Görü

sistemleri temel olarak sonuçlar çıkarma ya da karar verme işlemlerini bu alanlar üzerinden gerçekleştirir. Bu nedenle görüntü işleme teknolojilerinin hepsinde görüntü içerisinde alanlar arasındaki kenarları bulmada kendine has yaklaşımlar geliştirmelerinin yanında standartlaşmış bazı algoritmaları da destekledikleri görülmektedir.

EmguCV ile kenar çıkarma işleminde işleme alınan kaynak görüntüsü ile işlem sonucunda edinilen hedef görüntüsünün aynı renk uzayında ve renk derinliklerinde olması zorunluluğu vardır.

EmguCV, görüntüyü anlamlı parçalara ayırabilmek için optimize edilmiş cvSobel(), cvLaplacian() ve cvCanny() kenar bulma operatörünü sunmaktadır.

4.9.1. Sobel Kenar Çıkarma

Sobel, EmguCV'nin temel kenar bulma operatörlerindendir ve sayısallaşmış görüntü sinyalinin birinci türevinde oluşan maksimum noktaların belirlenmesi prensibine dayanır. EmguCV genişletilmiş Sobel operatörü ile görüntünün birinci, ikinci, üçüncü veya daha karmaşık türevlerini hesaplar. EmguCV ile Sobel yöntemine göre kenar bulma cvSobel(); fonksiyonu ile yapılır. EmguCV Sobel operatörü ile renkli ve gri düzey tanımlanmış görüntülerin kenarlarını başarılı bir şekilde belirler. CvInvoke sarmalayıcı kütüphanesi ile OpenCV'nin CV kütüphanesinden erişilen cvSobel() aşağıdaki yapıda kullanılır.

```
CvInvoke.cvSobel(Kaynak,
                 Hedef,
                 X,
                 Y,
                 Diyafram Açıklığı);
```

Kaynak: Kenarları çıkarılacak giriş görüntüsüdür.

Hedef: Kenarları çıkarılmış kaynak görüntü ile aynı renk uzayı ve derinliğine sahip hedef görüntüdür.

X: x türev sırası

Y: y türev sırası

Diyafram Açıklığı: görüntü üzerinde dolaşacak sobel konvülasyon matrisnin kenar geçiş duyarlılığını ifade eden parametredir. Diyafram açıklığı 1, 3, 5 ve 7 olabilir.

EmguCV'de Sobel operatörü Gauss yumuşatma ve farklılaşma sağlayarak görüntü içerisinde daha az gürültünün kenar olarak algılanmasını sağlar. cvSobel() fonksiyonu, x

türev sırası 1 ya da 0, y türev sırası 0 ya da 1 ve konvülasyon diyafram açıklığı 3 seçildiğinde daha başarılı sonuçlar verdiği görülür. x, y türev sıralarına ve diyafram açıklığı 3 seçildiğinde konvülasyon matrisleri aşağıdakilerden biri olarak seçilir (URL-16, 2013).

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \qquad \text{yada} \qquad \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

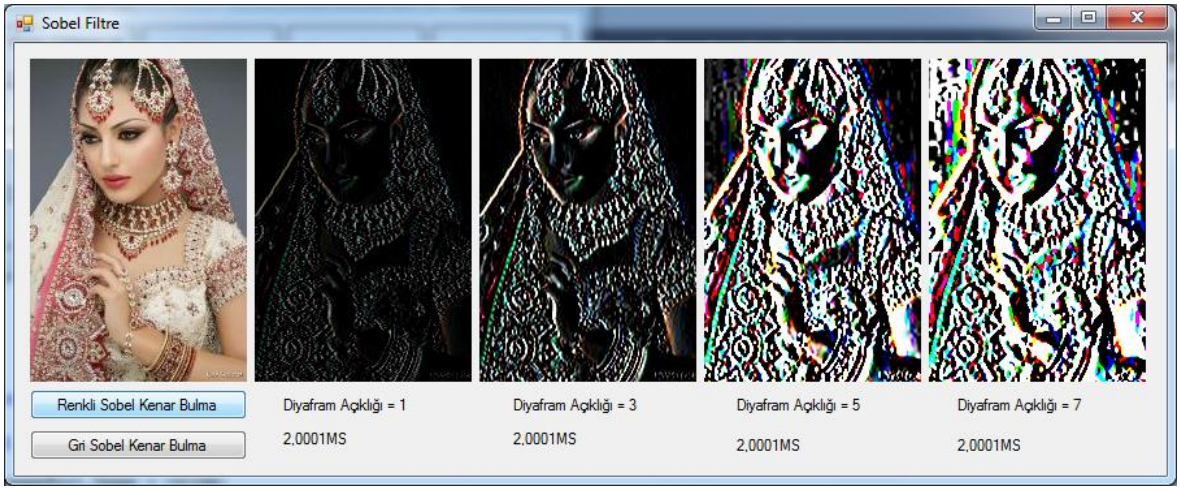
(x = 1, y = 0) (x = 0, y = 1) (x = 0, y = 1)

Geliştirilem Uygulama:

RGB renk uzayında ve byte renk derinliğinde yüklenen bir görüntünün cvSobel() operatörü ile kenarlarını çıkaran uygulama aşağıda verilmiştir.

```
private void Gri_Duzey_Kenar_Button_Click(object sender, EventArgs e){
using (Image<Gray, Byte> resimm = new Image<Gray, byte>("Resimler\\hindu3.jpg")){
ımageBox1.Image = resimm;
Image<Gray, Byte> sobel1 = new Image<Gray, byte>(resimm.Size);
Image<Gray, Byte> sobel3 = new Image<Gray, byte>(resimm.Size);
Image<Gray, Byte> sobel5 = new Image<Gray, byte>(resimm.Size);
Image<Gray, Byte> sobel7 = new Image<Gray, byte>(resimm.Size);
CvInvoke.cvSobel(resimm, sobel1, 1, 0, 1);
ımageBox2.Image = sobel1;
CvInvoke.cvSobel(resimm, sobel3, 1, 0, 3);
ımageBox3.Image = sobel3;
CvInvoke.cvSobel(resimm, sobel5, 1, 0, 5);
ımageBox4.Image = sobel5;
CvInvoke.cvSobel(resimm, sobel7, 1, 0, 7);
ımageBox5.Image = sobel7;  }        }
```

Yüklenen gri düzey bir görüntünün, x = 1, y = 0 türev sıraları ve farklı konvülasyon matrislerinde oluşan kenarlar Şekil 4.12'de verilmiştir.



Şekil 4.12: Farklı türev sıraları ve diyafram açıklıkları verilen Sobel operatöründe çıkarılan kenarlar.

4.9.2. Laplacian Kenar Çıkarma

Görüntünün sayısal sinyalinin ikinci türevinde sıfır noktaları kenar olarak alan operatördür. Birinci türev durumunda oluşan sinyalin tepe noktaları görüntü içerisindeki kenarların merkezini ifade ettiklerinden ikinci türev ile sıfır değerlerini alması sağlanır. Laplace operatörü görüntü kenarlarının merkezleri ile ilgilendiği için sonuçta elde edilen kenarlar Sobel operatörü ile elde edilen kenarlardan daha incedir. İkinci türev durumunda görüntüdeki gürültülerde sinyalde 0 noktasına eşit olacağı için gürültülere oldukça duyarlı bir operatördür. EmguCV Laplace operatörü ile kenar bulma işlemini, CvInvoke sarmal kütüphanesi aracılığıyla OpenCV'nin CV kütüphanesinden cvLaplace() fonksiyonu ile gerçekleştirir. EmguCV'de cvLaplace() fonksiyonu ile kenar çıkarma aşağıdaki gibi bir yapı gerektirir.

```
CvInvoke.cvLaplace(Kaynak,
                   Hedef,
                   Diyafram Açıklığı);
```

Kaynak: Kenar çıkarma işlemi uygulanacak giriş görüntüsünü ifade eder.

Hedef: Giriş görüntüsünün kenarları belirlenmiş çıkış görüntüsüdür.

Diyafram Açıklığı: Giriş görüntüsü üzerinde yatay ve dikey kenarları belirleyecek konvülasyon matrisinin kenar geçiş duyarlılığını ifade eder.

EmguCV, optimize edilmiş 1, 3, 5 ve 7 diyafram açıklığında konvülasyon matrisleri sunar. Laplace operatörü diyafram açıklığı 1 seçildiğinde aşağıdaki konvülasyon matrisi ile en hızlı sonuçları verir (URL-16, 2013).

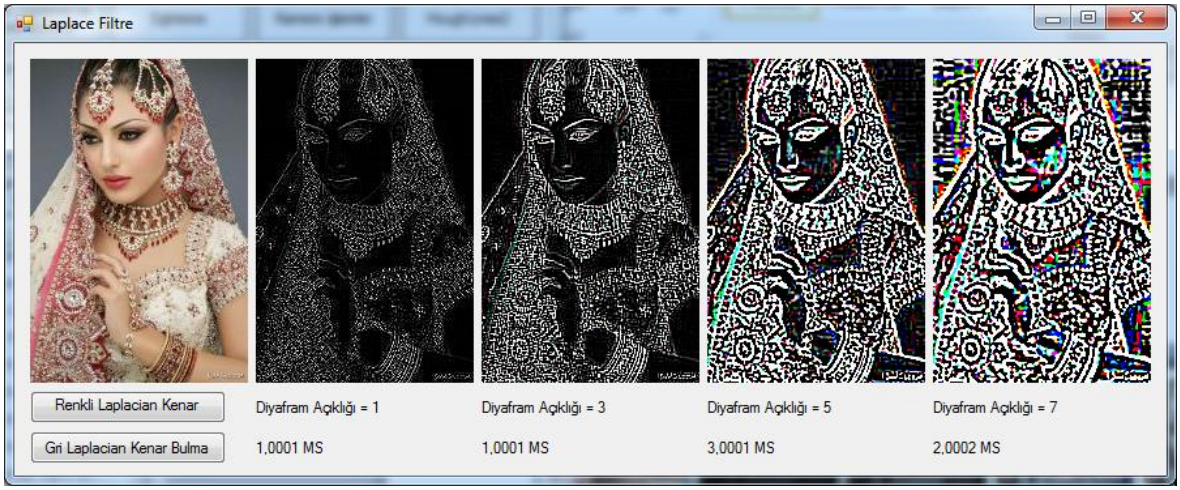
$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Geliştirilen Uygulama:

Yüklenmiş RGB renkli bir görüntünün farklı konvülasyon matrislerinde oluşan kenarları aşağıdaki uygulamada verilmiştir.

```
private void Laplacian_Kenar_Bul_Click(object sender, EventArgs e){
using (Image<Bgr, Byte> resimm = new Image<Bgr, byte>("Resimler\\hindu3.jpg")){
ımageBox1.Image = resimm;
Image<Bgr, Byte> laplacian1 = new Image<Bgr, byte>(resimm.Size);
Image<Bgr, Byte> laplacian3 = new Image<Bgr, byte>(resimm.Size);
Image<Bgr, Byte> laplacian5 = new Image<Bgr, byte>(resimm.Size);
Image<Bgr, Byte> laplacian7 = new Image<Bgr, byte>(resimm.Size);
CvInvoke.cvLaplace(resimm, laplacian1, 1);
ımageBox2.Image = laplacian1;
CvInvoke.cvLaplace(resimm, laplacian3, 3);
ımageBox3.Image = laplacian3;
CvInvoke.cvLaplace(resimm, laplacian5, 5);
ımageBox4.Image = laplacian5;
CvInvoke.cvLaplace(resimm, laplacian7, 7);
ımageBox5.Image = laplacian7;}      }
```

Yüklenen renkli görüntünün farklı diyafram açıklıklarından EmguCV ile yazılmış yukarıdaki programa göre oluşan sonuç görüntüleri Şekil 4.13'te verilmiştir.



Şekil 4.13: Yüklenen görüntünün Laplace operatörü ile farklı diyafram açıklıklarında belirlenen kenarları

4.9.3. Canny Kenar Çıkarma

Canny, Laplace ve Sobel operatörlerinin kenar bulma algılarındaki güçlü noktaları bir araya getirerek optimize edilmiş yeni bir kenar bulma yaklaşımı sunar. Canny, yüklenen görüntüyü öncelikle gauss filtresiyle tarayarak yumuşatır. İkinci aşamada bu görüntüyü Sobel operatörü ile tarayarak görüntü içerisinde kenar özelliği taşıyan alanların yönleri ve şiddetleri belirlenir. İkinci aşamada belirlenen kenar özelliği taşıyan alanlar yeni bir matris ile taranarak kenar olamayacak alanların elemine edilmesi üçüncü aşama olarak gerçekleştirilir. Son aşamada ise görüntü iki farklı eşikleme değeri ile eşiklenir ve hedef görüntü oluşturulur. EmguCV, bütün bu basamakları parametre değişken olarak kontrol eden cvCanny() fonksiyonu ile gerçekleştirir ve aşağıdaki gibi bir yapıda kullanılır.

```
CvInvoke.cvCanny(Kaynak,
                 Hedef,
                 Birinci Eşik,
                 İkinci Eşik,
                 Diyafram Açıklığı);
```

Kaynak: Canny operatörü ile kenarı alınmak istenen giriş görüntüsünü ifade eder.

Hedef: Kaynak görüntüsünün kenarları alınmış hedef görüntüsüdür.

Birinci Eşik: Sobel operatörleri ile yönleri ve şiddetleri belirlenmiş kenar olma özelliği taşıyan gürültülerin elemine edilmesi için kullanılan birinci karşılaştırma değeridir.

İkinci Eşik: Yönleri ve şiddetleri belirlenmiş alanların kenar olarak işaretlenmesi için kullanılan sınır değerdir. İkinci eşik birinci eşikten yaklaşık olarak en az üç kat büyük tercih edilir.

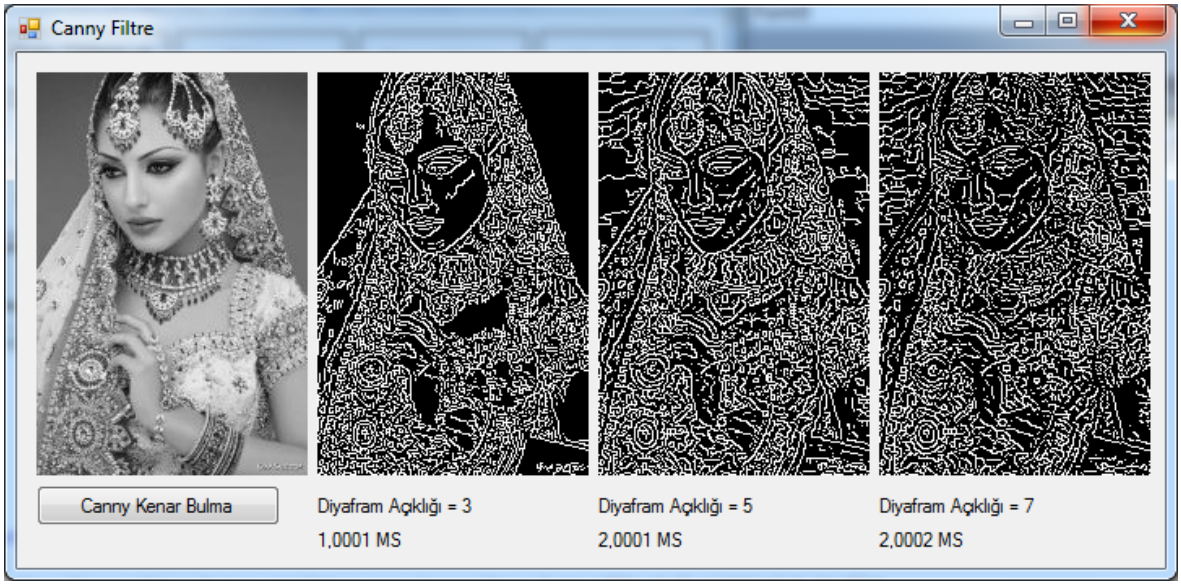
Diyafram Açıklığı: görüntü üzerinde kenar özelliği taşıyan alanların kenar geçiş duyarlılığıdır. Canny, diyafram açıklığı ve konvülasyon matrisi olarak Sobel operatörünü kullanmıştır. Diyagram açıklığı Sobel operatöründen farklı olarak 3, 5, 7 alınabilir.

Geliştirilen Uygulama:

Yüklenmiş görüntünün Canny operatörü ile farklı eşik ve diyafram açıklıklarında oluşacak görüntü aşağıdaki uygulamada verilmiştir.

```
private void Canny_Kenar_Bul_Click(object sender, EventArgs e) {
using (Image<Bgr, Byte> resimm = new Image<Bgr, byte>("Resimler\\hindu3.jpg")){
Image<Gray, Byte> gri = resimm.Convert<Gray, Byte>();
ımageBox1.Image = gri;
Image<Gray, Byte> canny3= new Image<Gray, byte>(resimm.Size);
Image<Gray, Byte> canny5 = new Image<Gray, byte>(resimm.Size);
Image<Gray, Byte> canny7 = new Image<Gray, byte>(resimm.Size);
CvInvoke.cvCanny(gri, canny3, 25, 75, 3);
CvInvoke.cvCanny(gri, canny5, 50, 150, 5);
CvInvoke.cvCanny(gri, canny7, 25, 200, 7);
ımageBox2.Image = canny3;
ımageBox3.Image = canny5;
ımageBox4.Image = canny7;          }       }
```

Verilen uygulamada yüklenen görüntünün farklı eşik ve diyafram açıklıklarından oluşan görüntüsü Şekil 4.14'de verilmiştir.



Şekil 4.14: Eşik değerleri ve diyafram açıklıkları farklı verilmiş Canny kenar görüntüleri

4.10. Renk Uzayı Dönüşümleri

Görüntü işlemenin teknoloji dünyasında ki önemi arttıkça buna paralel olarak görüntü renklerinin oluşturulacakları renk uzayı ayrı bir önem kazanmıştır. Görüntüleri oluşturan renkler ve ifade edildikleri renk kümeleri 1900 yıllardan beri gelişim göstermekte ve bu durumun sonucu olarak renkler günümüzde çok farklı teknikler ile elde edilmektedirler. Her renk uzayı görüntülerin farklı ortamlarda ifadesi için yeni bir yöntem sağlar. Bilgisayar ortamında renklerin ifadesinde RGB, basım araçlarında CMYK, hareketli hareketsiz görüntü renklerinin oluşturulmasında YCrCb renk uzayları çoğunlukta tercih ediliyorken, HSV ve HSL gibi renk uzayları görüntülerin ifade edilmesi için alternatif renk uzayları olarak tercih edilmektedirler. Renklerin öznitelikseL olarak elde edilmesinde CIE XYZ, Lab ve Luv renk uzayları, renkleri insan gözünün algıladığı şekilde bütün olarak oluşturma prensibine dayanır ve renk uzayları arasında sonuçları tahmin edilebilecek şekilde renk dönüşüm yöntemleri sunar. Her renk uzayı üç farklı değişkene bağlı olarak renkleri oluşturur.

EmguCV'de görüntüler, OpenCV tarafından desteklenen dokuz farklı renk uzayında oluşturulabilirler. Görüntüler renk uzayı parametresi tanımlanarak Image<Renk uzayı, Renk derinliği> yapısında oluşturulurlar. İstenilen renk uzayında ifade edilen bir görüntü amaç doğrultusunda optimize edilmiş dönüşüm bayrakları ile birbirleri arasında dönüşüm yapabilirler. Dönüşüm işlemleri temel olarak RGB renk uzayı ile diğer renk uzayları ara-

sında gerçekleştirilir. EmguCV ile RGB renk uzayından farklı başka iki renk uzayı arasında dönüşüm yapılmak istendiği zaman görüntüler önce RGB renk uzayına dönüştürülür sonrasında RGB renk uzayından hedef renk uzayına dönüştürme işlemi yapılarak işlem tamamlanır.

EmguCV'de renk uzayı dönüşümleri cvCvtColor() fonksiyonu ile yapılır ve bu fonksiyon aşağıdaki yapıda kullanılarak bütün renk uzayları arasında dönüşüm işlemini mümkün kılar.

```
CvInvoke.cvCvtColor(Kaynak,
                    Hedef,
                    Dönüşüm Kodu);
```

Kaynak: Başka renk uzayına dönüştürülmek istenen görüntüyü temsil eder.

Hedef: Dönüştürülen görüntüyü ifade edecek yeni görüntü değişkenidir ve kaynak görüntü ile aynı renk derinliğinde olması gerekir.

Dönüşüm Kodu: Renk dönüşüm işleminin yönünü belirler

EmguCV'de optimize edilmiş dönüşüm kodlarıyla renkli görüntülerde dönüşüm işlemi uygulandığında giriş renk uzayı renkleri ile yeni renk uzayının renkleri arasında gözle fark edilemeyecek bir dönüşüm başarısı sağlanır. EmguCV'de dönüşüm kodları/bayrakları aşağıdaki yapıda tanımlanır ve kullanılır.

CV_<Kaynak Renk Uzayı>2<Hedef Renk Uzayı >.

EmguCV'de dönüşüm bayrakları CvEnum ara kütüphanesi aracılığıyla tanımlanırlar. Aşağıdaki gibi bir kullanım söz konusudur.

Emgu.CV.CvEnum.COLOR_CONVERSION.**CV_KAYNAK2HEDEF**

Tanımlı bütün renk uzayları arasında dönüşüm bu temel yapıya bağlı kalınarak aşağıdaki gibi gerçekleştirilir.

4.10.1. RGB, CIE XYZ Dönüşümü

cvCvtColor() fonksiyonu yüklenen bir görüntüyü yüklendiği renk uzayından başka bir renk uzayına dönüşüm koduna bağlı olarak kolaylıkla gerçekleştirir. Her dönüşüm işlemini ifade eden farklı dönüşüm kodları vardır. RGB renk uzayından CIE XYZ renk uzayına tanımlı renkler CV_BGR2XYZ ve CV_RGB2XYZ bayrakları ile dönüşüm yapılırken XYZ renk uzayında RGB renk uzayına CV_XYZ2RGB ve CV_XYZ2BGR bayrakları ile dönüşüm yapılabilir.

RGB renk uzayından CIE XYZ renk uzayına dönüşüm işleminde kaynak renk uzayında ifade edilen görüntünün her pikseli yeni renk uzayı için yeniden hesaplanır. EmguCV ile RGB2XYZ ve XYZ2RGB dönüşümü Eşitlik 4.1 ve 4.2'de ki denklemler ile somutlaşır (URL-16, 2013).

$$\begin{matrix} X \\ Y \\ Z \end{matrix} = \begin{matrix} 0.412453 & 0.357582 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{matrix} * \begin{matrix} R \\ G \\ B \end{matrix} \qquad 4.1$$

$$\begin{matrix} R \\ G \\ B \end{matrix} = \begin{matrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{matrix} * \begin{matrix} X \\ Y \\ Z \end{matrix} \qquad 4.2$$

Geliştirilen Uygulama:

BGR renk uzayında tanımlanmış bir görüntünün XYZ renk uzayına EmguCV ile dönüşümü aşağıdaki uygulamada ifade edilmiştir.

```
private void Rgb2Xyz_Donus_CheckedChanged(object sender, EventArgs e){
using (Image<Bgr, Byte> bgrres = new Image<Bgr, byte>("Resimler\\balik.jpg"))    {
Image<Xyz, Byte> xyzres = new Image<Xyz, byte>(bgrres.Size);
CvInvoke.cvCvtColor( bgrres,
                     xyzres,
                     COLOR_CONVERSION.CV_BGR2XYZ);
ımageBox1.Image = bgrres;
ımageBox2.Image = xyzres;    }}
```

BGR renk uzayında yüklenen bir görüntünün XYZ renk uzayına dönüşümünde oluşan sonuç görüntüsü Şekil 4.15'de görülmektedir.

Şekil 4.15: BGR renk uzayından XYZ renk uzayına çevrilen görüntü.

4.10.2. RGB, YCrCb Dönüşümü

EmguCV'de RGB renk uzayı ile YCrCb renk uzayı arasındaki dönüşümlerde kullanılan bayraklar sırasıyla RGB'den YCrCb'ye; CV_RGB2YCrCb ve CV_BGR2YCrCb, YCrCb'den RGB'ye; CV_YCrCb2RGB ve CV_YCrCb2BGR bayrakları kullanılarak yapılır.

RGB ile YCrCb renk uzayları arasındaki görüntü dönüşümlerinde, görüntülerin her piksel değerleri aşağıdaki denklemlerde yeniden hesaplanır. Kullanılacak eşitlik, tercih edilen dönüşüm bayrağına göre seçilir. Dönüşüm işlemleri, dönüşüm işlemlerinin yönüne göre Eşitlik 4.3 ve 4.4 de verilen denklemler ile sağlanır (URL-16, 2013).

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad \text{(a)}$$
$$Cr = (R - Y) * 0.713 + delta \quad \text{(b)} \qquad 4.3$$
$$Cb = (B - Y) * 0.564 + delta \quad \text{(c)}$$

$$R = Y + 1.403 * (Cr - delta) \quad \text{(a)}$$
$$G = Y - 0.344 * (Cr - delta) - 0.714 * (Cb - delta) \quad \text{(b)} \qquad 4.4$$
$$B = Y + 1.773 * (Cb - delta) \quad \text{(c)}$$

$$delta = \begin{cases} 8\ Bit\ görüntü\ için & 128 \\ 16\ bit\ görüntü\ için & 32768 \\ ondalık\ görüntü\ için & 0.5 \end{cases}$$

Geliştirilen Uygulama:

RGB renk uzayında yüklenen bir görüntünün optimize edilmiş EmguCV fonksiyonlarıyla YCrCb renk uzayına çeviren program aşağıda verilmiştir.

```
private void Rgb2YCrCb_Donus_CheckedChanged(object sender, EventArgs e){
using (Image<Bgr, Byte> bgrres = new Image<Bgr, byte>("Resimler\\balik.jpg"))    {
Image<Ycc, Byte> yccres = new Image<Ycc, byte>(bgrres.Size);
CvInvoke.cvCvtColor( bgrres,
                     yccres,
                     COLOR_CONVERSION.CV_BGR2YCrCb);
ımageBox1.Image = bgrres;
ımageBox2.Image = yccres;    }}
```

4.10.3. RGB, HSV Dönüşümü

Farklı yaklaşımlarla renk oluşturan renk uzayları arasında dönüşüm işlemi de teorik olarak farklı bir yaklaşım gerektirir. İnsanın algı sistemini baz alarak her bir rengi ayrı ayrı formüle eden ve oluşturan HSV renk uzayı ile diğer renk uzayları arasında dönüşüm işlemi optimize edilmiş denklemler ile yapısal olarak diğer dönüşüm işlemlerinden farklılık göstermemektedir.

EmguCV'de RGB renk uzayı ile HSV renk uzayı arasında dönüşümlerde sırasıyla RGB'den HSV'ye CV_RGB2HSV, CV_BGR2HSV ve HSV'den RGB'ye CV_HSV2RGB, CV_HSV2BGR optimize dönüşüm bayrakları kullanılır.

RGB renk uzayı ile HSV renk uzayı arasında görüntü dönüşüm işlemlerinde görüntülerin her piksel değerleri yeni renk uzayı için yeniden tanımlanırlar. 8 bit veya 16 bit renk derinliğindeki görüntülerin renk derinliğinde 0-1 ondalıklı olarak ifade edilir. RGB ve HSV renk uzayları arasında dönüşüm işlemlerinde dönüşüm işlemlerinin yönüne göre aşağıdaki matematiksel ifadeler yeni piksel değerlerinin tanımlanması için kullanılır (URL16-2013);

Value değeri;

$$V = \max\ R, G, B \qquad 4.5$$

Saturation Değeri

$$S = \frac{V - \min\ R,G,B}{V}\ \ V \neq 0, ise\ S = 0 \qquad 4.6$$

Hue (renk) değeri

$$H = \begin{cases} G - B\ * \frac{60}{S}, & V = R \\ 180 + \ B - R\ * \frac{60}{S}, & V = G \\ 240 + \ R - G\ * \frac{60}{S}, & V = B \end{cases} \qquad 4.7$$

H<0 ise H=H+360

Olduğunda

Çıkışlar 0≤V≤1, 0≤S≤1, 0≤H≤360.

Piksel değerleri hedef renk uzayı dönüştürüldüğünde

8 bit görüntü: V=V*255, S=S*255, H=H/2.

16 bit görüntü: V=V*65535, S=S*65535, H=H. (Desteklenmiyor)

32 bit görüntü: V=V, S=S, H=H.

Geliştirilen Uygulama:

Aşağıda verilen program yüklenen bir görüntünün EmguCV'nin optimize edilmiş dönüşüm bayrakları ile BGR renk uzayından HSV renk uzayına dönüşümünü ifade etmektedir.

```
private void Rgb2Hsv_Donus_CheckedChanged(object sender, EventArgs e){
using (Image<Bgr, Byte> bgrres = new Image<Bgr, byte>("Resimler\\balik.jpg"))    {
Image<Hsv, Byte> hsvres = new Image<Hsv, byte>(bgrres.Size);
CvInvoke.cvCvtColor( bgrres,
                     hsvres,
                     COLOR_CONVERSION.CV_BGR2HSV);
ımageBox1.Image = bgrres;
ımageBox2.Image = hsvres;    }}
```

4.10.4. RGB, HLS Dönüşümü

HLS renk uzayı, HSV renk uzayı gibi insanın algı sistemine benzer olarak görüntülerin üç değişkene bağlı olarak tek bir formül ile ifade eden renk uzayıdır. HLS renk uzayı dönüşümleri diğer renk uzayları arasındaki dönüşümlerden yapısal olarak farklı değildir.

EmguCV'de HLS renk uzayı ile diğer renk uzayları arasındaki dönüşüm işlemlerinde CV_BGR2HLS, CV_RGB2HLS, CV_HLS2BGR ve CV_HLS2RGB dönüşüm bayrakları kullanılır.

RGB uzayında yüklenen 8 veya 16 bit görüntülerin HLS uzayına dönüşümünde görüntülerin piksel değerleri 0-1 aralığında ondalıklı sayılarla ifade edildikten sonra aşağıda verilen Eşitlik 4.8, 4.9, 4.10 ve 4.11'de verilen denklemler ile her pikselin değeri yeni renk uzayı için yeniden hesaplanır (URL16-2013).

$$V_{max} = \max\ R, G, B \quad \text{(a)}$$

$$V_{min} = \min\ R, G, B \quad \text{(b)} \qquad 4.8$$

Luminance (aydınlık) değeri

$$L = \frac{V_{max} + V_{min}}{2} \qquad 4.9$$

Saturation (yoğunluk) değeri

$$S = \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}}, & L < 0.5 \\ \frac{V_{max} - V_{\min}}{2 - V_{max} + V_{min}}, & L \geq 0.5 \end{cases} \qquad 4.10$$

Hue (renk) değeri

$$H = \begin{cases} G - B * \frac{60}{s}, & V_{max} = R \\ 180 + B - R * \frac{60}{s}, & V_{max} = G \\ 240 + R - G * \frac{60}{s}, & V_{max} = B \end{cases} \quad 4.11$$

Piksel değerleri hedef renk uzayı dönüştürüldüğünde

8 bit görüntü: V=V*255, S=S*255, H=H/2.

16 bit görüntü: V=V*65535, S=S*65535, H=H. (Desteklenmiyor)

32 bit görüntü: V=V, S=S, H=H.

Geliştirilen Uygulama:

BGR uzayında yüklenen bir görüntünün HLS renk uzayında yeniden ifade eden EmguCV programı aşağıda verilmiştir.

```
private void Rgb2Hls_Donus_CheckedChanged(object sender, EventArgs e){
using (Image<Bgr, Byte> bgrres = new Image<Bgr, byte>("Resimler\\balik.jpg"))    {
Image<Hls, Byte> hlsres = new Image<Hls, byte>(bgrres.Size);
CvInvoke.cvCvtColor( bgrres,
                     hlsres,
                     COLOR_CONVERSION.CV_BGR2HLS);
ımageBox1.Image = bgrres;
ımageBox2.Image = hlsres;    }}
```

4.10.5. RGB, CIE L*a*b Dönüşümü

İnsanın renkleri algılayış şeklini referans alarak renkleri aygıttan bağımsız kendi doğal yapılarında oluşturmaya çalışan L*a*b renk uzayında renklerin elde edilmesi farklı bir mantığa göre oluşturuluyorsa da diğer renk uzayları arasında renk değişimi yine belli matematiksel formüller ile yapılır.

EmguCV ile L*a*b renk uzayından diğer renk uzaylarına görüntü dönüşümlerinde CV_BGR2Lab, CV_RGB2Lab, CV_Lab2BGR ve CV_Lab2RGB bayrakları kullanılır.

8 bit veya 16 bit renk derinliğinde RGB veya BGR renk uzayında yüklenen bir görüntü aşağıda verilen eşitlikler ile L*a*b renk uzayında ifade edilir (URL-16, 2013). Görüntü önce XYZ uzayında ifade edilir.

$$\begin{matrix} X \\ Y \\ Z \end{matrix} = \begin{matrix} 0.412453 & 0.357582 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{matrix} * \begin{matrix} R \\ G \\ B \end{matrix} \quad 4.12$$

$$X = \frac{X}{X_{n,}}, \quad X_n = 0.950456\ ise \quad 4.13$$

$$Z = \frac{Z}{Z_n}, \quad Z_n = 1.088754\ \text{ise} \quad 4.14$$

$$L = \begin{matrix} 116 * Y^{1}{}_{3}, & Y > 0.008856\ için. \\ 903.3 * Y, & Y \leq 0.008856\ için. \end{matrix} \quad 4.15$$

$$a = 500 * \ f\ X\ - f\ Y\ + delta \quad 4.16$$

$$b = 200 * \ f\ Y\ - f\ Z\ + delta \quad 4.17$$

$$f\ t\ = \begin{matrix} {}^{3}\overline{t}, & t > 0.008856\ için \\ 7.787 * t + 16/116, & t \leq 0.008856\ için \end{matrix} \quad 4.18$$

8 bit görüntü için delta = 128.

Ondalıklı görüntü için delta = 0.

Çıkışta 0≤L≤100, -127≤a≤127, -127≤b≤127.

Piksel değerleri son olarak hedef renk derinliği tipine dönüştürülür.

8 bit görüntü için: L=L*255/100, a=a+128, b=b+128

16 bit görüntü: Desteklenmiyor.

32 bit görüntü için: L, a, b değerleri değişmez.

Geliştirilen Uygulama:

BGR renk uzayında tanımlanmış görüntünün L*a*b renk uzayına dönüşümünü veren program aşağıda verilmiştir.

```
private void Rgb2Lab_Donus_CheckedChanged(object sender, EventArgs e){
using (Image<Bgr, Byte> bgrres = new Image<Bgr, byte>("Resimler\\balik.jpg"))    {
Image<Lab, Byte> labres = new Image<Lab, byte>(bgrres.Size);
CvInvoke.cvCvtColor( bgrres,
                     labres,
                     COLOR_CONVERSION.CV_BGR2Lab);
ımageBox1.Image = bgrres;
ımageBox2.Image = labres;    }}
```

4.10.6. RGB, CIE L*u*v Dönüşümü

İki boyutlu modelleme ve renk tolerans hesaplamaları dışında L*a*b renk uzayı ile tamamen aynı mantığa dayanan L*u*v renk uzayı, dönüşüm işlemlerinde birkaç matematiksel hesaplama farklılığı dışında L*a*b renk uzayı ile aynı matematiksel hesaplamalardan ibarettir. EmguCV ile RGB-L*u*v renk uzayları arasında dönüşüm işlemlerinde CV_RGB2Luv, CV_BGR2Luv, CV_Luv2RGB ve CV_Luv2BGR bayrakları kullanılmaktadır.

EmguCV'de yüklenen 8 veya 16 bit renk derinliğinde, 0-1 aralığında ondalıklı piksel değeri alan BGR bir görüntünün dönüşüm bayrakları aşağıdaki matematiksel eşitlikler ile optimize edilmiştir (URL-16, 2013).

Görüntü RGB'den XYZ uzayına dönüştürülür.

$$\begin{matrix} X \\ Y \\ Z \end{matrix} = \begin{matrix} 0.412453 & 0.357582 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{matrix} * \begin{matrix} R \\ G \\ B \end{matrix} \quad 4.19$$

$$L = \begin{matrix} 116 * Y^{\frac{1}{3}} - 16, & Y > 0.008856\, ise \\ 903.3 * Y, & Y \leq 0.008856\, ise \end{matrix} \quad 4.20$$

$$u' = 4 * \frac{X}{X+15*Y+3*Z} \quad 4.21$$

$$v' = 9 * \frac{Y}{X+15*Y+3*Z} \quad 4.22$$

$$u = 13 * L * \ u' - u_n \ , \ u_n = 0.19793943\ oldu\check{g}unda \quad 4.23$$

$$v = 13 * L * \ v' - v_n \ , \ v_n = 0.46831096\ oldu\check{g}unda \quad 4.24$$

Çıkışlar: 0≤L≤100, -134≤u≤220, -140≤v≤122.

Çıkış değerleri hedef veri tipine dönüştürülür.

8 bit görüntü: L=L*255/100, u=(u+134)*255/354, v=(v+140)*255/256

16 bit görüntü: Desteklenmiyor.

32 bit görüntü: L, u, v değerleri aynı kalır.

Geliştirilen Uygulama:

BGR yüklenmiş bir görüntünün Luv renk uzayında ifade eden EmguCV programı aşağıda verilmiştir.

```
private void Rgb2Luv_Donus_CheckedChanged(object sender, EventArgs e){
using (Image<Bgr, Byte> bgrres = new Image<Bgr, byte>("Resimler\\balik.jpg"))    {
Image<Luv, Byte> luvres = new Image<Luv, byte>(bgrres.Size);
CvInvoke.cvCvtColor( bgrres,
```

```
                luvres,
                COLOR_CONVERSION.CV_BGR2Luv);
ımageBox1.Image = bgrres;
ımageBox2.Image = luvres;    }        }
```

4.11. Canny Operatörü ile Eş Zamanlı Görüntü İşleme

Eş zamanlı görüntü işleme, kullanım alanının çokluğu nedeniyle görüntü işlemede çok önemli bir yere sahiptir. EmguCV ile geliştirilen yazılım kütüphaneleriyle sisteme bağlı herhangi bir kameradan eş zamanlı görüntü alınıp işlenebilmekte ve sonuçlar üzerinden kararlar oluşturulabilmektedir. EmguCV ile geliştirilen görüntü işleme platformunda OpenCV'nin açık kaynak kodlu yüzlerce fonksiyonu aracılığıyla kameradan okunan bir görüntü üzerinde eş zamanlı vücut uzuvları takip edilebilir, metin okunabilir, akış halindeki görüntüde belli nesnelerin tespiti, görüntüde ki metni okuma işlemleri gibi çok sayıda işlem eş zamanlı olarak kolaylıkla yapılabilir.

Eş zamanlı olarak bir görüntü verisini işleyebilmek için sistemde tanımlanmış bir kameranın olması yeterlidir. EmguCV ile Studio.Net platformunda kameradan görüntü akışı oluşturmak için olay prosedürleri oluşturulabilir, timer nesnesi kullanılabilir veya bir olay prosedürünün içerisinde bir döngü kurularak giriş biriminden sürekli görüntüler yakalanıp çeşitli araçlar aracılığıyla form ortamında gösterilebilir.

EmguCV ile kameradan sürekli görüntü akışı sağlayabilmek için CV kütüphanesindeki Capture() nesnesinden aşağıdaki gibi bir değişkenin oluşturulup kamera aygıtına erişimin sağlanması gerekir.

```
Emgu.CV.Capture yakala = new Emgu.CV.Capture();
```

Kamera erişimi sağlandıktan sonra yakalama değişkeni aracılığıyla kamera ile ilgili bütün özellikler artık kolayca ayarlanabilir ve aşağıdaki gibi QueryFrame() metodu ile kameradan anlık görüntüler yakalanabilir.

```
using (Image<Bgr,Byte> kaynak = yakala.QueryFrame())
```

Kameradan görüntüler birer resim yapısında yakalanır. Eş zamanlı görüntü işleme mantığı kameradan anlık olarak yakalanan resim verisi üzerinde istenilen işlemlerin uygulanıp işlem sonuçlarının anlık olarak çıkış birimlerine aktarılması mantığı ile gerçekleştirilir.

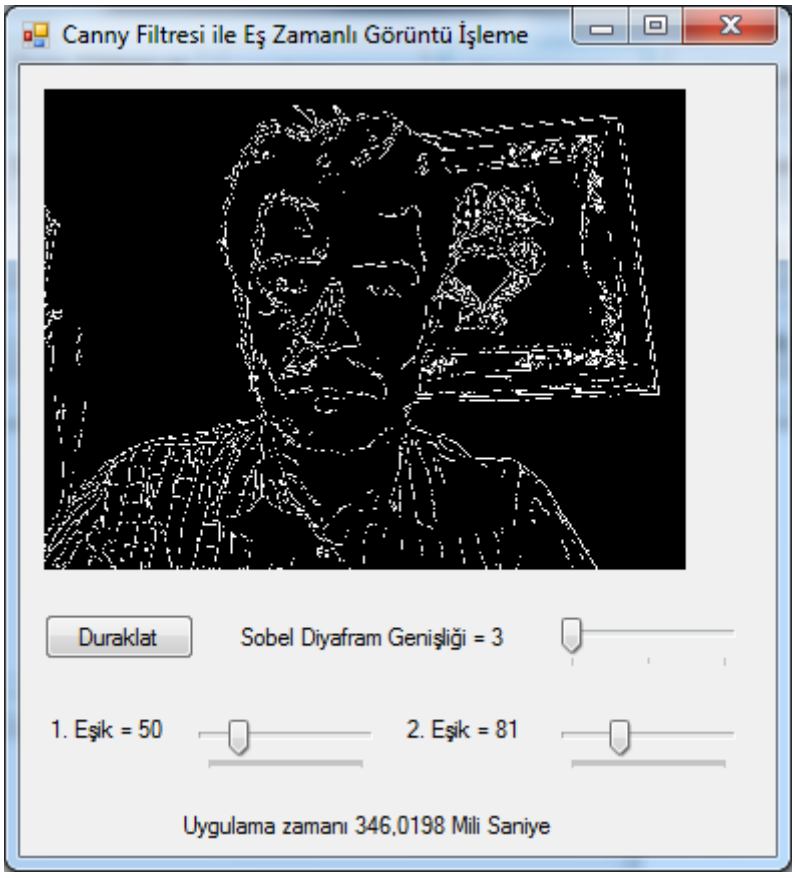
Geliştirilen Uygulama:

Aşağıdaki kod parçacığında, kameradan anlık olarak yakalanan görüntü verisine Canny operatörü ile kenar bulma işlemi uygulanmış ve sonuçlar anlık olarak form üzerindeki ımageBox aracına aktarılmıştır.

```
Emgu.CV.Capture yakala = null;
Image<Gray, Byte> kaynak;
Image<Gray, Byte> hedef;
bool calısma = false;
DateTime Bas, Son;
private void Form13_Load(object sender, EventArgs e){
try{
yakala = new Emgu.CV.Capture();
yakala.FlipHorizontal = !yakala.FlipHorizontal;
}
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}
Application.Idle += Goruntu_Yakala;
calısma = true;
}
void Goruntu_Yakala(object sender, EventArgs arguman){
try{
int diyafram;
if (trackBar1.Value == 1)
diyafram = 3;
else if (trackBar1.Value == 2)
diyafram = 5;
else
diyafram = 7;
label1.Text = "Sobel Diyafram Genişliği = " + diyafram.ToString();
label2.Text = "1. Eşik = " + trackBar2.Value.ToString();
label3.Text = "2. Eşik = " + trackBar3.Value.ToString();
```

```
Bas = DateTime.Now;
using (kaynak = new Image<Gray, byte>(yakala.QueryFrame().Size))
using (hedef = new Image<Gray, byte>(kaynak.Size)){
CvInvoke.cvCvtColor(yakala.QueryFrame(), kaynak, CO-
LOR_CONVERSION.CV_BGR2GRAY);
CvInvoke.cvCanny(kaynak, hedef, trackBar2.Value, trackBar3.Value, diyafram);
ımageBox2.Image = hedef;
Son = DateTime.Now;
label4.Text = "Uygulama zamanı " + (Son - Bas).TotalMilliseconds.ToString() + " Mili
Saniye";
} }
catch (Exception hata){
MessageBox.Show(hata.Message);
} }
```

Yukarıda geliştirilen program çalıştırıldığında oluşan sonuç görüntüsü şekil 4.16'de verilmiştir.



Şekil 4.16: Canny operatörü ile eş zamanlı görüntü işleme

4.12. Görüntü Yumuşatma Filtreleri

Görüntü verilerini amaca uygun zenginleştiren işlemleri ifade eden filtreler; bazen görüntüdeki belli ayrıntıların gizlenmesi veya ön plana çıkarılması işlemleri için tercih edilirler. Kenar keskinleştirme veya kenar yakalama filtrelerine ek olarak yumuşatma filtreleri de görüntü işlemede çok sık kullanılan filtrelerdendirler.

Filtreleme işlemi; MxN matris yapısındaki görüntünün her piksel değerinin çekirdek yapısındaki 3x3, 5x5, 7x7 gibi matrisler ile yeniden hesaplanması işlemidir. Filtre matrislerinin konvülasyon değerleri istenilen amaca yönelik belirlenip işleme sokulurlar.

EmguCV ile yumuşatma işlemleri ulaşılan OpenCV'nin cvSmooth() fonksiyonunda tanımlı filtreleme bayraklarına göre yapılır. Studio.Net ortamında ulaşılan cvSmooth() fonksiyonun kullanımı aşağıdaki gibidir.

```
CvInvoke.cvSmooth(IntPtr kaynak,
                  IntPtr hedef,
                  int SMOOTH_TYPE,
                  int parametre1,
                  int parametre2,
                  double parametre3,
                  double parametre4);
```

Kaynak: yumuşatma işlemine alınan giriş görüntüsüdür.

Hedef: verilen parametrelere göre işlenen görüntü verilerinin aktarıldığı kaynak görüntü ile aynı boyut, renk uzayı ve derinliğinde olan görüntü yapısıdır.

Smooth_Type: kaynak görüntü üzerine uygulanacak yumuşatma filtresinin belirlendiği parametredir. OpenCV'de görüntü yumuşatma işlemli için kullanılan filtreler aşağıdaki bayraklar ile temsil edilir.

- **CV_BLUR_NO_SCALE (Simple Blur No Scaling):** Basit ölçeksiz bulanıklaştırma filtresidir. Görüntülerde amaca uygun olmayan alanların indirgenmesinde tercih edilen filtrelerdendir. Görüntü piksellerini parametre1xparametre2 satır ve sütun genişliğindeki matris ile cvIntegral fonksiyonuna göre yeniden hesaplar.
- **CV_BLUR (Simple Blur):** Görüntülerde amaca uygun olmayan alanların indirgenmesinde tercih edilen filtrelerdendir. 1/(parametre1*parametre2) ölçeğinde parametre1xparametre2 komşulukta piksel değerlerini bir pikselde toplar.

- **CV_GAUSSIAN (Gauss Blur):** parametre1xparametre2 konvulasyon matrisine göre görüntüyü bulanıklaştırır.
- **CV_MEDIAN (Median Blur):** parametre1xparametre1 kare konvülasyon matrisine göre her pikselin değerinin komşu piksellerin ortanca değerini için yeniden hesaplar.
- **CV_BILATERAL (bilateral filter):** Görüntüdeki her pikselin yoğunluk değeri 3x3 komşuluktaki diğer piksellerin yoğunluklarının ağırlıklı ortalamaları ile değiştirilir. Bilateral filtrede parametre1 = renk standart sapmasını ve parametre2 = boşluk standart sapmasını ifade eder.

Parametre1: Filtreler için ilk yumuşatma parametresi

Parametre2: Yumuşatma filtreleri için ikinci yumuşatma parametresi. Şayet parametre2 =0 değeri alınırsa Blur, No Scaled Blur ve Gaussian konvülasyon matrisleri paratmetre1 değerine göre kare olarak oluşturulur. Yani parametre2 değeri parametre1 değerini alır.

Parametre3: Bu parametre Gauss filtresi için standart sapma değerinin ifade eder. paratmetre3 = 0 olması durumunda

- $sigma = \begin{matrix} n \\ 2 \end{matrix} - 1 \ * 0.3 + 0.8, \ n = parametre1 \ yatay \ çekirdek$
  $\begin{matrix} n \\ 2 \end{matrix} - 1 \ * 0.3 + 0.8, \ n = parametre2 \ düşey \ çekirdek$

Standart sigma 3x3 ve 7x7 konvülasyon matrisleri için en iyi sonuçları verir. parametre1 ve parametre2 sıfır değerinde iken parametre3 sıfır değil ise, konvülasyon çekirdeği sigma değerine göre hesaplanır.

Parametre4: Gauss filtresi için kare olmayan konvülasyon çekirdeği durumunda dikey sigma değerini belirtir.

OpenCV kütüphaneleri cvSmooth() fonksiyonu aracılığıyla görüntü yumuşatma işlemleri için çeşitli metotlar sağlamaktadır. Bütün bu metotlar cvSmooth() fonksiyonuna girilecek uygun parametre değerleri ile kolayca kullanılabilir.

Ölçeksiz bulanıklaştırma (blur no scale) filtresi tek kanallı 8 bit görüntüden 16 bit görüntü ve 32 bit ondalıklı görüntüden 32 bit ondalıklı görüntüde işlemler yapabilir. Bulanıklaştırma (blur) ve gauss bulanıklaştırma (gauss blur) filtreleri 1 veya 3 kanallı 8 ve 32 bit görüntülere uygulanabilir. Median filtresi 1 veya 3 kanallı 8 bit renk derinliğindeki görüntülere uygulanabilir

Geliştirilen Uygulama:

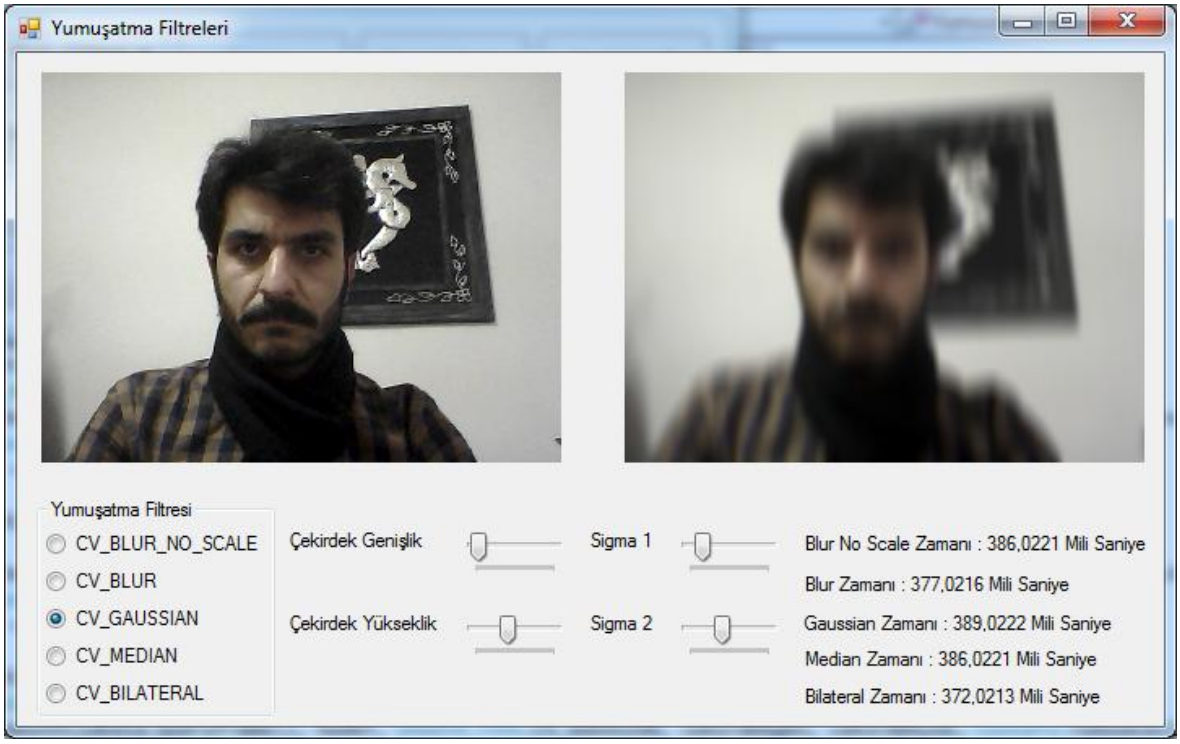
Aşağıda kameradan anlık olarak yakalanan görüntüye belirlenen parametre değerlerine göre seçilen yumuşatma filtresi uygulanmış ve uygulama sonuçları form üzerinde görüntüleme araçlarına anlık olarak aktarılmıştır.

```
DateTime Bas, Son;
Capture yakala;
bool calis = false;
private void Form12_Load(object sender, EventArgs e){
try{
yakala = new Capture();
yakala.FlipHorizontal = !yakala.FlipHorizontal;
}
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}
Application.Idle += YumusatmaFiltresi;
calis = true;
}
void YumusatmaFiltresi(object sender, EventArgs Arguman){
int p1 = trackBar1.Value;
if (p1 % 2 == 0)
p1++;
Bas = DateTime.Now;
using (Image<Bgr, Byte> hedef = new Image<Bgr, byte>(yakala.QueryFrame().Size)){
int cekirdekgen, cekirdekyuk;
if (trackBar1.Value % 2 == 0)
cekirdekgen = trackBar1.Value + 1;
else
cekirdekgen = trackBar1.Value;
if (trackBar2.Value % 2 == 0)
cekirdekyuk = trackBar2.Value + 1;
else
cekirdekyuk = trackBar2.Value;
```

```
ımageBox1.Image = yakala.QueryFrame();
if (radioButton1.Checked == true){ //CV_BLUR_NO_SCALE
CvInvoke.cvSmooth( yakala.QueryFrame(),
                   hedef,
                   SMOOTH_TYPE.CV_BLUR_NO_SCALE,
                   cekirdekgen,
                   cekirdekyuk,
                   Convert.ToDouble(trackBar3.Value),
                   Convert.ToDouble(trackBar4.Value));
Son = DateTime.Now;
label5.Text = "Blur No Scale Filtre Zamanı : " + (Son -
Bas).TotalMilliseconds.ToString()+" Mili Saniye";
}
else if (radioButton2.Checked == true){ //CV_BLUR
CvInvoke.cvSmooth( yakala.QueryFrame(),
                   hedef,
                   SMOOTH_TYPE.CV_BLUR,
                   cekirdekgen,
                   cekirdekyuk,
                   Convert.ToDouble(trackBar3.Value),
                   Convert.ToDouble(trackBar4.Value));
Son = DateTime.Now;
label6.Text = "Blur Filtre Zamanı : " + (Son - Bas).TotalMilliseconds.ToString() + " Mili
Saniye";
}
else if (radioButton3.Checked == true){// CV_GAUSSIAN
CvInvoke.cvSmooth( yakala.QueryFrame(),
                   hedef,
                   SMOOTH_TYPE.CV_GAUSSIAN,
                   cekirdekgen,
                   cekirdekyuk,
                   Convert.ToDouble(trackBar3.Value),
                   Convert.ToDouble(trackBar4.Value));
```

```
Son = DateTime.Now;
label7.Text = "Gaussian Filtre Zamanı : " + (Son - Bas).TotalMilliseconds.ToString() + "
Mili Saniye";
}
else if (radioButton4.Checked == true){ // CV_MEDIAN
CvInvoke.cvSmooth( yakala.QueryFrame(),
                   hedef,
                   SMOOTH_TYPE.CV_MEDIAN,
                   cekirdekgen,
                   cekirdekyuk,
                   Convert.ToDouble(trackBar3.Value),
                   Convert.ToDouble(trackBar4.Value));
Son = DateTime.Now;
label8.Text = "Median Filtre Zamanı : " + (Son - Bas).TotalMilliseconds.ToString() + "
Mili Saniye";
}
else{ // CV_BILATERAL
CvInvoke.cvSmooth( yakala.QueryFrame(),
                   hedef,
                   SMOOTH_TYPE.CV_BILATERAL,
                   cekirdekgen,
                   cekirdekyuk,
                   Convert.ToDouble(trackBar3.Value),
                   Convert.ToDouble(trackBar4.Value));
Son = DateTime.Now;
label9.Text = "Bilateral Filtre Zamanı : " + (Son - Bas).TotalMilliseconds.ToString() + "
Mili Saniye";
}
ımageBox2.Image = hedef; }      }
```

Yukarıdaki kod parçacığı çalıştırıldığında oluşan anlık sonuç görüntüsü şekil 4.17'de verilmiştir.



Şekil 4.17: Kameradan alınan görüntü üzerinde eş zamanlı yumuşatma filtreleri

4.13. Vücut Uzuvları Tanıma ve İşaretleme

EmguCV ile vücut uzuvlarını ve olabilecek diğer nesnelerin görüntü üzerinde tespiti Haar öznitelik tabanlı sınıflandırıcılar sayesinde diğer algoritmik yaklaşımlardan çok daha kolay olabilmektedir. Biyometrik olarak öznitelikleri iyi tespit edilmiş ve sınıflandırılmış bir nesne veya vücut bölgesinin görüntü içerisinde tespiti uygulamaları, öznitelik dosyalarının oluşturulması başarısına bağlıdır.

EmguCV ile öznitelikleri daha önce belirlenmiş ve xml dosyasına kaydedilmiş nesnenin veya vücut uzuvunun xml dosyasını HaarCascade sınıfından türetilmiş bir değişkene aşağıdaki gibi atamak gerekiyor.

```
private HaarCascade yüzler, gözler;
private void Form14_Load(object sender, EventArgs e){
try{
yüzler = new HaarCascade(@"HaarCascadeFile\haarcascade_frontalface_default.xml");
gözler = new HaarCascade(@"HaarCascadeFile\haarcascade_eye.xml");
}
catch (Exception hata){
MessageBox.Show(hata.Message);
}        }
```

Haar öznitelik tabanlı fonksiyonlar ile görüntüde istenilen alanların tespiti için gerekli temel ön koşul, kaynak görüntüsünün gri tanımlanmış olması gerekliliğidir. Görüntü üzerinde istenilen nesnenin EmguCV ile Stduio.Net ortamında C# dilinde tespiti için DetectHaarCascade() fonksiyonun uygun parametreler ile aşağıdaki gibi tanımlanması gerekiyor.

```
grigörüntü.DetectHaarCascade(HaarCascade nitelikdosyası,
                             double ölçekfaktörü,
                             int komşuluk,
                             Haar_Detection_Type bayrak,
                             Size enküçükpencere);
```

nitelikdosyası: Haar öznitelik sınıflandırıcı xml dosyasıdır

ölçekfaktörü: Görüntüdeki nesnenin arandığı çerçeve pencerece ölçekleme derecesidir. İşlem başarısını etkileyen faktörlerdendir. 1.1 ölçek faktörü her defasında %10 oranında ölçeklenen pencereler ile görüntü analizi yapılır.

komşuluk: Bir nesneyi oluşturan minimum komşu dikdörtgen sayısını ifade eder. Bir nesneyi olabilecek minimum sayıda dikdörtgenle ifade etmek için -1 değeri verilir. -1'den daha küçük bir sayı ile hiçbir nesne tespit edilemeye bilir. 0 komşuluk değeri ile herhangi bir dikdörtgen gruplama yapılmaz, kullanıcı için özelleştirilmiş bir gruplama modeli sunar.

bayrak: Aranılan nesne için metod sağlar. Şuan için sadece CV_HAAR_DO_CANNY_PRUNING bayrağı belirtilebilir. Canny kenar bulma operatörünün 1. Eşik ve 2. Eşik değerlerine bağlı olarak yapılan ikilileştirme işleminde aranılan nesne kenarlarının bu eşik değerlerine bağlı olarak zemine indirgenme sakıncası vardır.

enküçükpencere: Görüntü üzerinde dolaştırılacak varsayılan en küçük pencere boyutunu belirtir.

DetectHaarCascade() fonksiyonu ile taranan bir görüntü, konumları belirlenmiş nesne sınırlarında bir dikdörtgenin başlangıç ve bitiş noktaları değerlerini döndürür. Sınır noktaları dizi tipinde tanımlanmış bir değişkende saklanmış olan nesnenin sınırları sonraki adımda oluşturulacak bir döngüde ilgili kaynak görüntüsü üzerinde dikdörtgen şeklinde çizilerek nesne tespiti tamamlanmış olur.

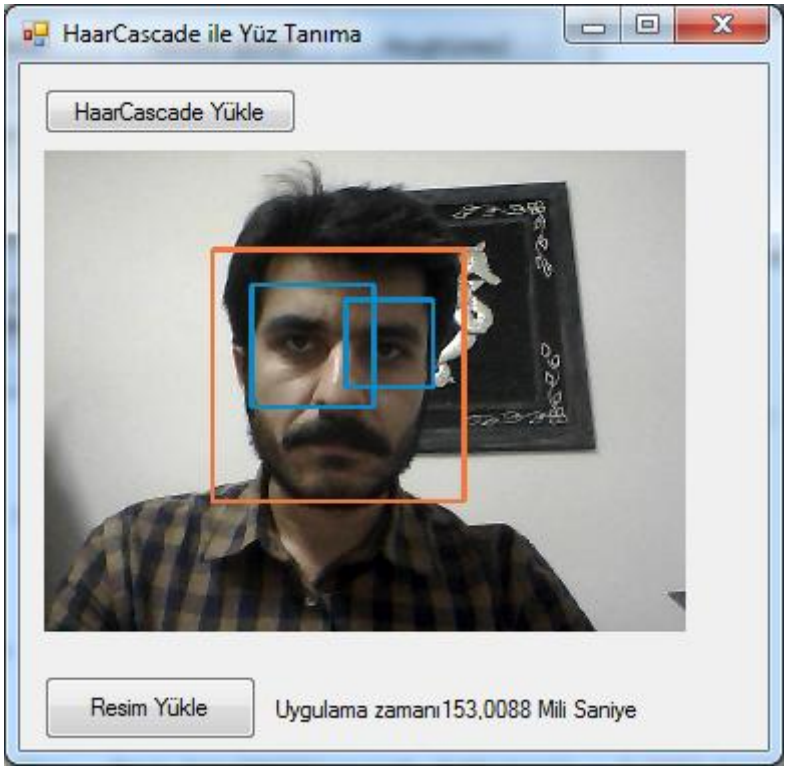
Geliştirilen Uygulama:

Aşağıdaki program prosedürlerinde, öncelikle görüntü üzerinde aranılacak yüz ve göz alanlarının Haar öznitelik kurallarına göre oluşturulmuş xml dosyaları yüklenmiştir. istenilen Haar öznitelik dosyaları yüklendikten sonra DetectHaarCascade() fonksiyonu ile istenilen özelliklerde grileştirilmiş tek kanallı görüntü üzerinde yüz ve göz araması yapılmış ve sonucunda bulunan nesne alanları çerçeve içerisinde anlık olarak gösterilmiştir.

```
private HaarCascade yüzler, gözler;
private Capture yakala = null;
bool calis = true;
DateTime Bas, Son;
private void Form14_Load(object sender, EventArgs e){
try{
yakala = new Capture(0);
yakala.FlipHorizontal = !yakala.FlipHorizontal;
yüzler = new HaarCascade(@"HaarCascadeFile\haarcascade_frontalface_default.xml");
gözler = new HaarCascade(@"HaarCascadeFile\haarcascade_eye.xml");
}
```

```
catch (Exception hata){
MessageBox.Show(hata.Message);
}        }
private void timer1_Tick(object sender, EventArgs e){    // Kameradan görüntü işleme
using (Image<Bgr, Byte> goruntu = yakala.QueryFrame()){
if (goruntu != null){
Image<Gray, Byte> grigoruntu = goruntu.Convert<Gray, Byte>();
Bas = DateTime.Now;
var yuzler = grigoruntu.DetectHaarCascade(yüzler,
                                    1.1,
                                    1,
                                    HAAR_DETECTION_TYPE.SCALE_IMAGE,
                                    new Size(goruntu.Width / 8, goruntu.Height / 8))[0];
var gozler = grigoruntu.DetectHaarCascade(gözler,
                                    1.1,
                                    1,
                                    HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
                                    new Size(goruntu.Width / 8, goruntu.Height / 8))[0];
foreach (var yuz in yuzler){
goruntu.Draw(yuz.rect, new Bgr(60, 120, 240), 3);
}
foreach (var goz in gozler){
goruntu.Draw(goz.rect, new Bgr(195, 135, 15), 3);
}
ımageBox1.Image = goruntu;
Son = DateTime.Now;
label2.Text = "Uygulama zamanı" + (Son - Bas).TotalMilliseconds.ToString() + " Mili
Saniye";
}        }        }
```

Yukarıdaki kod parçacıkları çalıştırıldığında oluşacak anlık görüntüler şekil 4.18'de verilmiştir.



Şekil 4.18: Anlık alınan görüntüde yüz ve göz tanımlama

4.14. Görüntü İçerisindeki Dairelerin Tespiti

Sayısal görüntülerde geometrik şekillerin tespiti görüntü işlemede çok önemli bir yere sahiptir ve uygulama alanı oldukça fazladır. Dışarıdan alınan sayısal görüntülerde çevresel koşullardan dolayı bazen görüntü içerisindeki çemberin sınırları tam olarak belirlenememektedir. Böylesi durumlarda hough dönüşümleri ile sınırları tam olarak belli olmayan olası çemberler bile tespit edilebilmektedir.

Hough dönüşümleri ile çemberlerin tespitinde temel olarak kenarları çıkarılmış görüntüde kenar özelliği taşıyan her bir piksel, olası geometrik şekillerin polar koordinattaki değerlerini kullanan bir akümülatör matrisi üzerinde dolaştırılarak her pikselin olası geometrik şekilleri taraması sağlanır. Akümülatör değerleri yüksek olan şekiller görüntü üzerinde belirgin olan şekillerdir.

Gri ifade edilen görüntü üzerindeki çemberlerin tespitinde gürültü özelliği taşıyan piksellerin çeşitli yumuşatma filtreleri ile bastırılması çember sınırlarının tespitinde daha iyi sonuçları sağlayacaktır.

EmguCV ile Studio.Net ortamında hough dönüşümleri ile görüntü içerisindeki çemberlerin tespiti OpenCV fonksiyonlarından cvHoughCircles() fonksiyonu ile son derece

başarılı sonuçlar alınmaktadır. Hough çember bulma için kullanılan cvHoughCircles() fonksiyonun kullanımı ve parametreleri aşağıdaki gibidir.

```
IntPtr bulunancemberr = CvInvoke.cvHoughCircles(IntPtr kaynak,
                                                IntPtr cember_stor,
                                                HOUGH_TYPE.METOD,
                                                double akü_çöz,
                                                double minMesafe,
                                                double parametre1,
                                                double parametre2,
                                                int minRadius,
                                                int maxRadius);
```

kaynak: 8 bit gri tanımlanmış üzerinde çemberlerin araştırılacağı giriş görüntüsüdür.

çember_stor: Bulunan çemberler için bellekte ayrılan saklama alanıdır. İşlem yapıldığı sırada çemberler için ihtiyaç duyulan koordinat bilgileri, yarıçap uzunluğu gibi bilgiler bellekte 32 bit ondalıklı ve üç boyutlu olarak saklanır. Taranan çemberler için bir tampon bellek görevi görür.

METOD: Kaynak görüntü üzerinde çember arama metodlarını sağlar. En başarılı sonuçlar CV_HOUGH_GRADIENT metodu ile sağlanır.

akü_çöz: Taranan çemberlerin merkezlerinin akümülatör çözünürlük değerleridir. 1 değerinde akümülatör çözünürlük değeri giriş görüntüsünün çözünürlük değeri ile aynı çözünürlükte hesap edilir. 2 değerinde, giriş görüntüsünün genişlik ve yükseklik değerinden iki küçük alınacaktır.

minMesafe: Tespit edilecek çemberlerin merkezleri arasındaki minimum mesafeyi ifade eder. Bu değerin küçük alınması birden fazla dairenin üst üste çakışmasına neden olur. Çok büyük atanması durumunda da görüntü içerisindeki bazı çemberler belirlenemeyebilir.

parametre1: Görüntüdeki kenarların tespiti için kullanılan CV_HOUGH_GRADIENT metodunda kullanılan Canny detektörünün 1. Eşik değeridir.

parametre2: CV_HOUGH_GRADIENT metodunda çemberlerin merkez akümülatör değeridir. Küçük alınması durumunda görüntü yanlış alanlar çember olarak algılanabilir. En büyük akümülatör değerine sahip çember ilk olarak belirlenen daire olur.

minRadius: Aranılacak çemberlerin en küçük yarıçap değerini ifade eder.

maxRadius: Aranılan çemberlerin en büyük yarıçap değerini ifade eder.

cvHoughCircles() fonksiyonu ile tek kanallı 8 bit tanımlanmış gri bir görüntüde belirlenen parametrelere göre tespit edilen çemberlerin her biri için merkez koordinat (X ve Y) bilgileri ve ilgili çemberin yarıçapı bilgilerini üç boyutlu bir dizi için oluşturur. Oluşturulan bu bilgilere göre sonraki basamaklarda her daire görüntü üzerinde belirlenen merkez koordinatlarda ve belirlenen yarıçap genişliğinde çemberler çizilir.

Geliştirilen Uygulama:

Aşağıdaki kod bloğu; yüklenen görüntüye hough dönüşüm fonksiyonun uygulanabilmesi için öncelikle 8 bit renk derinliğinde tek kanallı gri düzey bir görüntüye dönüştürülmektedir. Dönüşüm işleminden sonra görüntüde kenar özelliği taşımayan gürültülerin minimize edilmesi için Gauss filtresi uygulanarak görüntü yumuşatılmıştır. Devamında görüntü hough çember fonksiyonuna göre taranarak, her çember için merkezi X ve Y koordinat bilgilerinin yanı sıra yarıçap bilgisinin de üretilmesi sağlanır. Bu değerler sonraki basamaklarda belirlenen özelliklerde çemberlerin görüntüye anlık olarak eklenmesi ile sonuçlandırılır.

```
DateTime Bas, Son;
void HoughCemberleriResim(){    // Resimde Çember Çıkarma
Bas = DateTime.Now;
using (Image<Bgr, Byte> kaynak = new Image<Bgr, byte>(@"Resimler\para.jpg"))
using (Image<Gray, Byte> hedef = new Image<Gray, byte>(kaynak.Size))
using (Image<Bgr, Byte> houghim = kaynak.Clone()){
CvInvoke.cvCvtColor(kaynak, hedef, COLOR_CONVERSION.CV_BGR2GRAY);
CvInvoke.cvSmooth(hedef, hedef, SMOOTH_TYPE.CV_GAUSSIAN, 9, 0, 0.0, 0.0);
using (MemStorage stor = new MemStorage()){
unsafe{
IntPtr cemberr = CvInvoke.cvHoughCircles(hedef,
                                         stor,
                                         HOUGH_TYPE.CV_HOUGH_GRADIENT,
                                         2,
                                          hedef.Height / 4,
                                          200,
                                         100,
                                         1,
                                         hedef.Width / 4);
```

```
for (int i = 0; ; i++){
try{
float* p = (float*)CvInvoke.cvGetSeqElem(cemberr, i);
CvInvoke.cvCircle(    houghim,
                      new Point((int)Math.Round(p[0]), (int)Math.Round(p[1])),
                      3,
                      new MCvScalar(0, 0, 255),
                      -1,
                      LINE_TYPE.CV_AA,
                      0);
CvInvoke.cvCircle(    houghim,
                      new Point((int)Math.Round(p[0]), (int)Math.Round(p[1])),
                      (int)Math.Round(p[2]),
                      new MCvScalar(0, 255, 0),
                      3,
                      LINE_TYPE.CV_AA,
                      0);
}
catch (Exception hata){
break;
}        }        }
ımageBox1.Image = hedef;
ımageBox2.Image = houghim;
Son = DateTime.Now;
this.Text = (Son - Bas).TotalMilliseconds.ToString();
}        }        }
```

Yukarıdaki kodlar çalıştırıldığında oluşan uygulama sonucu şekil 4.19'da verilmiştir.

Şekil 4.19: Yüklenen görüntüde Hough dönüşümü ile dairelerin belirlenmesi

4.15. Görüntü İçerisinde Eş Zamanlı Çizgi Tespiti

Sayısal görüntü analizinde geometrik şekillerin tespiti genel bir problem durumunu ifade eder. Sayısal görüntülerden geometrik şekillerin analizinde genel olarak kenar çıkarma ve görüntü öz nitelikleri üzerinde çeşitli değişiklikler yapan ön operasyonlar gerçekleştirilir. Bu ön işlemlerden sonra ikilileştirilen görüntü üzerinde kenar özelliği olan piksellerin geometrik şekil olup olmadığı ayrı bir problem konusunu oluşturmaktadır. Hough dönüşümleri her bir geometrik şekil için oluşturulan özellik dizilerinin görüntü içerisinde kenar özelliği taşıyan her piksel ile karşılaştırılıp açık oylama ile ilgili pikselin gösterdiği geometrik karakter ile eşleştirilmesi mantığı ile çalışır. Aynı geometrik karakterde olan pikseller konumlarına göre gruplandırılarak tespit edilen geometrik şekil elde edilmiş olur.

Hough dönüşümleri ile görüntü içerisinde kenar özelliği taşıyan alanların tespiti için öncelikle görüntünün gri düzey tanımlanmış örneğinde Canny operatörü ile görüntüde kenar özelliğindeki alanların ikili ifadesi sağlanmalıdır. Hough dönüşüm algoritması görüntüde ki çizgileri akümülatör olarak adlandırılan iki boyutlu bir dizi ile tespit eder. Görüntüde kenar özelliğindeki piksellerin, bu akümülatör değerlerine göre en yüksek oyu alan pikselleri kendi aralarında gruplandırılır ve bu pikseller doğrusal bir çizgiyi ifade eder.

EmguCV ile geliştirilen yazlım ortamında görüntüde ki çizgilerin tespitinde cvHoughLines2() fonksiyonu kullanılmaktadır. Bu fonksiyon uygun parametre değerlerinde

görüntü içerisindeki çizgilerin tespitinde oldukça etkilidir. cvHoughLines2() fonksiyonu aşağıdaki gibi hazırlanmalıdır.

```
IntPtr cizgiler=CvInvoke.cvHoughLines2(IntPtr görüntü,
                                       IntPtr sakla,
                                       HOUGH_TYPE metod,
                                       double rho,
                                       double theta,
                                        int çizgi_eşik,
                                       double parametre1,
                                       double parametre2);
```

görüntü: 8 bit tek kanallı ikili görüntüyü ifade eder. Probabilistic metot durumunda görüntü fonksiyonu ile çalışılır.

sakla: çizgi özelliği gösteren olası yapıların bellekte saklanması için tek satır ve tek sütun yapısında tampon bellek görevi görür. Akümülatör değerine göre çizgi özelliği taşıyan yapıların başlangıç ve bitiş noktaları aynı yapıdan oluşturulan bir diziye aktarılır.

metod: Görüntüde çizgi çıkarma için metot belirler.

- **CV_HOUGH_STANDARD:** Her çizginin iki ondalıklı sayı (p, θ) tarafından ifade edildiği klasik çizgi çıkarma metodudur. p değeri (0,0) noktası ile çizgi arasındaki mesafe, θ çizginin x ekseni ile olan açısını ifade eder. Bu durumda çizgi için 32 bit float veriler üretilir.
- **CV_HOUGH_PROBABILISTIC:** Görüntüde genellikle düz hatlar mevcutsa kullanılan olasılıksal metottur. Hatları kesik kesik göstermek yerine bir bütün olarak hatları birleştirir. Her hat bir bütün olarak bir başlangıç ve bitiş noktası ile temsil edilir. 32 bit static int (CV_32SC4) veri tipinde değerler oluşturur.
- **CV_HOUGH_MULTI_SCALE:** Çok ölçekli klasik türev dönüşümlü metottur. Görüntülerdeki hatlar CV_HOUGH_STANDARD metoduna göre belirlenir.

rho: Hat ile ilişkili piksellerde çözünürlük mesafesini belirler

theta: Radyan cinsinden açısal çözünürlük değeridir.

çizgi_eşik: Çizgi akümülatörü değerinden büyük değerler alan alanların çizgi olarak belirtilmesinin sağlar.

parametre1: Çizgi bulma metodlarına bağlı birinci parametreyi ifade eder.

- CV_HOUGH_STANDART metodu parametre1=0 değeri için kullanılmaz

- CV_HOUGH_PROBABILISTIC metodu için parametre1 değeri minimum çizgi uzunluğunu belirtir.
- CV_HOUGH_MULTI_SCALE metodunda mesafe çözünürlüğü (rho) parametresi için bölen değeri ifade eder. Burada rho kaba mesafe çözünürlüğünü ifade edecek, rho/parametre1 ise kesin çözünürlük mesafesi için kullanılacaktır.

parametre2: Çizgi bulma metotları için ikinci bağıl prametredir.

- CV_HOUGH_STANDART metodu parametre2 = 0 değeri için kullanılmaz
- CV_HOUGH_PROBABILISTIC metodunda aynı hattı oluşturan çizgi parçaları arasında olabilecek en büyük mesafeyi belirtir.
- CV_HOUGH_MULTI_SCALE metodu için bu parametre yeni theta değerinin belirlenmesinde kullanılır. theta parametresi çok ölçekli metotta kaba theta olarak alınırken yeni theta; theta/parametre2 işlemi ile yeniden hesaplanır.

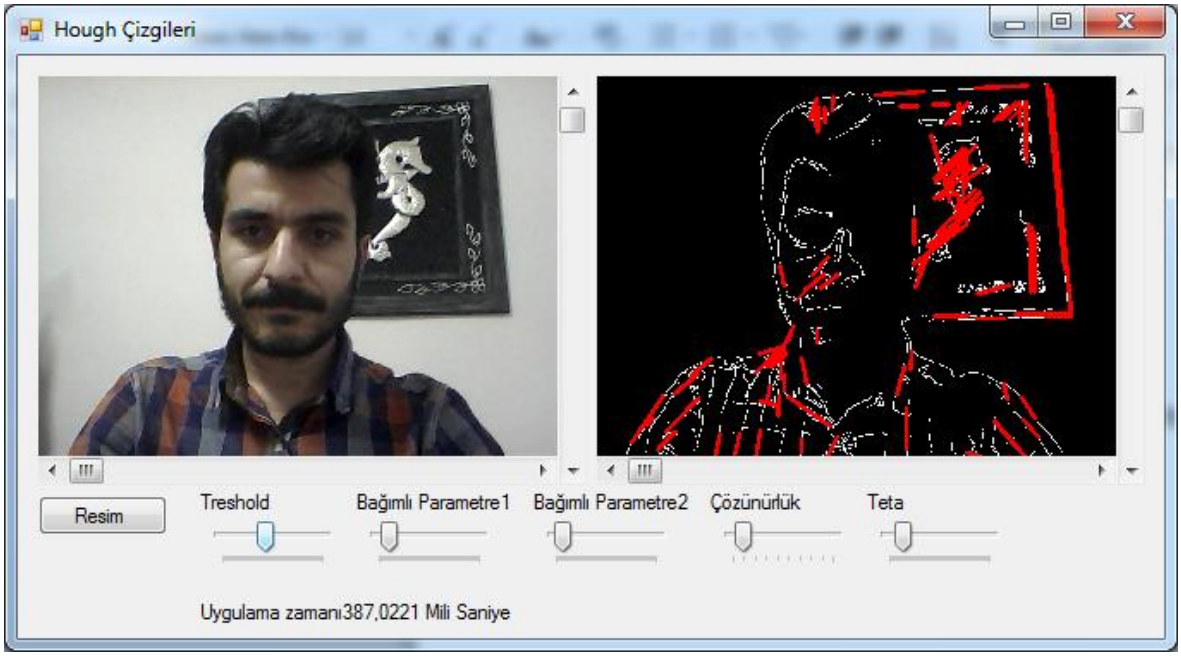
Geliştirilen Uygulama:

Aşağıdaki kod bloklarında geliştirilen uygulama eş zamanlı kamera görüntüsünde tespit edilen doğrusal hatlar işaretlenerek anlık çıkış birimine aktarılmıştır.

```
Capture yakala = null;
bool calis = false;
private void Form16_Load(object sender, EventArgs e){
try{
   yakala = new Capture();
   yakala.FlipHorizontal = !yakala.FlipHorizontal;
}
catch (Exception hata){
   MessageBox.Show(hata.Message);
   return;
}
Application.Idle += HoughCizgiKamera;
calis = true;
ımageBox1.Image = null;
ımageBox2.Image = null;
}
void HoughCizgiKamera(object sender, EventArgs arguman){//Kamerada Çizgi Çıkarma
DateTime bas = DateTime.Now;
```

```
using (Image<Bgr, Byte> kaynak = new Image<Bgr, byte>(yakala.QueryFrame().Size))
using (Image<Gray, Byte> hedef = new Image<Gray, byte>(kaynak.Size))
using (Image<Bgr, Byte> houghim = kaynak.Clone()){
using (MemStorage sakla = new MemStorage()){
CvInvoke.cvCanny(yakala.QueryFrame(), hedef, 50, 200, 3);
CvInvoke.cvCvtColor(hedef, houghim, COLOR_CONVERSION.CV_GRAY2BGR);
IntPtr cizgiler = CvInvoke.cvHoughLines2(hedef,
                                         sakla,
                                         HOUGH_TYPE.CV_HOUGH_PROBABILISTIC,
                                         trackBar4.Value,          //rho
                                         Math.PI / trackBar5.Value,  //theta
                                         trackBar1.Value,          //çizgi eşiği
                                         trackBar2.Value,          //bağlı param1
                                         trackBar3.Value);         //bağlı param2
for (int i = 0; ; i++){
try{
unsafe{
Point* hat = (Point*)CvInvoke.cvGetSeqElem(cizgiler, i);
CvInvoke.cvLine(houghim, hat[0], hat[1], new MCvScalar(0, 0, 255), 3, LI-
NE_TYPE.EIGHT_CONNECTED, 0);
}       }
catch (Exception){
break;
}       }       }
ımageBox1.Image = yakala.QueryFrame();
ımageBox2.Image = houghim;
DateTime son = DateTime.Now;
label6.Text = "Uygulama zamanı" + (son - bas).TotalMilliseconds.ToString() + " Mili Sa-
niye";
}       }
```

Yukarıda verilen kod blokları çalıştırıldığında giriş biriminden alınan görüntüdeki çizgiler şekil 4.20'de anlık olarak form üzerindeki çıkış biriminde gösterilmiştir.



Şekil 4.20: Giriş biriminden alınan görüntüden hough dönüşümü ile çizgi bulma

4.16. Görüntüdeki Köşeleri Bulma

Köşe algılama görüntü işlemde sık başvurulan bir uygulamadır. Görüntünün içeriğinin yorumlanması, özellikleri hakkında yorum yapılabilmesi açısından köşe algılama işlemleri görüntü işleme için çok önemli bir yere sahiptir. Görüntü işlemenin nihai hedeflerinden olan hareket algılama, birbiriyle görüntü iliştirme, 3D modelleme ve nesne tanıma gibi uygulamaların başarısı görüntü içerisindeki nesnelere ait köşelerin başarılı bir şekilde tespit edilmesine bağlıdır.

Günümüzde köşe tespit işlemleri için farklı yaklaşımlar mevcuttur. EmguCV ile geliştirilen yazılım ortamında yönüne göre köşe skoru farkı dikkate alınarak geliştirilen Harris köşe dedektörü kullanılmaktadır. Harris köşe algoritması tek kanalı gri tanımlanmış görüntüler üzerinde uygulanmaktadır.

Görüntüler içerisindeki köşeleri tespit eden cvCornerHarris() fonksiyonu aşağıdaki gibi tanımlanır ve kullanılır.

```
CvInvoke.cvCornerHarris(IntPtr kaynak,
                        IntPtr harrisKöşeler,
                        int blokBoyutu,
                        int diyafram,
```

double k);

kaynak: İçerisinde kenar arama işleminin yapılacağı kaynak görüntüdür.

harrisKöşeler: Kaynak görüntü ile aynı boyutlarda üzerinde tespit edilen köşelerin saklandığı 32 bit float tanımlanmış hedef görüntüsüdür.

blokBoyutu: komşuluk boyutu büyüklüğünü belirtir.

diyafram: Sobel operatörü için diyafram açıklığı değeridir. 1, 3, 5, 7 değerlerini alabilir.

k: harris dedektörü için serbest parametredir. blokBoyutu x blokBoyutu matrisinde ki her piksel için 2x2 gradient kovaryasyonlarını (iki ya da daha çok olasılıksal değişkenin birleşik değişimi) hesaplar ve saklar.

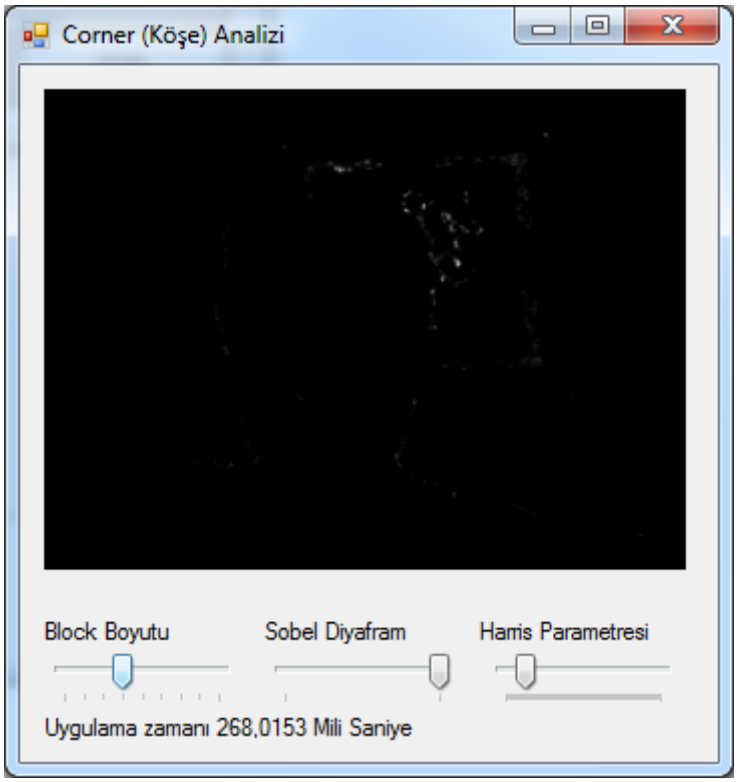
Geliştirilen Uygulama:

Aşağıda, yüklenen bir görüntünün Harris dedektörüne göre köşelerini bulan programın kod blokları verilmiştir. Yüklenen görüntü üzerinde harris dedektörü ile köşelerin tespit edilebilmesi için oncelikle kaynak görüntü tek kanallı gri düzey bir görüntüye dönüştürülüyor, devamında bu görüntü üzerinde tespit edilen köşeler 32 bit float tanımlanmış kaynak görüntü ile aynı boyutlardaki görüntüde işaretleniyor. Görüntü üzerinde belirlenen parametrelere göre tespit edilen köşeler anlık olarak çıkış birimine aktarılıyor.

```
Capture yakala;
bool calis = false;
DateTime Bas, Son;
private void Form17_Load(object sender, EventArgs e){
try{
yakala = new Capture();
yakala.FlipHorizontal = !yakala.FlipHorizontal;
}
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}
Application.Idle += KoseBulKamera;
calis = true;
}
void KoseBulKamera(object sender, EventArgs arguman){
```

```
try{
int diyafram;
if (trackBar2.Value == 1)
diyafram = 3;
else if (trackBar2.Value == 2)
diyafram = 5;
else
diyafram = 7;
Bas = DateTime.Now;
using (Image<Gray, Byte> kaynak = new Image<Gray, Byte>(yakala.QueryFrame().Size))
using(Image<Gray,float> hedef=new Image<Gray,float>(kaynak.Size)){
CvInvoke.cvCvtColor(yakala.QueryFrame(), kaynak, CO-
LOR_CONVERSION.CV_BGR2GRAY);
CvInvoke.cvCornerHarris( kaynak,
                         hedef,
                         trackBar1.Value,    // blok boyutu
                         diyafram,          //Sobel için diyafram
                         trackBar3.Value / 100); // harris parametresi
ımageBox1.Image = hedef;
Son = DateTime.Now;
label4.Text = "Uygulama zamanı " + (Son - Bas).TotalMilliseconds.ToString() + " Mili
Saniye";
}
}
catch (Exception hata){
DialogResult cevap = MessageBox.Show(hata.Message);
if (cevap == DialogResult.OK)
{
Application.Idle -= KoseBulKamera;
yakala.Dispose();
return;
}       }       }
```

Geliştirilen uygulama çalıştırıldığında oluşan sonuç görüntüsü şekil 4.21'de verilmiştir.



Şekil 4.21: Harris dedektörü ile köşe tespit etme

4.17. Blob Analizi

Görüntü işlemede aynı özniteliklere sahip bölgelerin görüntü içerisinde belirlenmesine dayalı temel tekniktir. Blob görüntü içerisinde bazı özellikleri sabit veya belli aralıklarda değişkenlik gösterebilen görüntü alanlarını ifade eder. Bu nedenle nesnelerin zeminden farklı öznitelikseL özelliklere sahip olduğu uygulamalarda genellikle tercih edilen bir nesne tespit uygulamasıdır. Zemin rengi ile görüntü içerisindeki diğer alanların veya özellikle görüntü içerisinden çıkarılmak istenilen alanın piksel değerleri birbirine yakın değilse bu yöntem nesne takibinde yüksek başarı ve esneklik sağlar.

Kabarcık çıkarma yöntemi temel olarak ikilileştirilmiş bir görüntü de aynı niteliksel durumları gösteren bağlı piksel gruplarının görüntüden analizidir. Blob işleminin başarısı kaynak görüntüye uygulanan ikilileştirme işleminin başarısı ile doğru orantılıdır.

EmguCV yazılım ortamında görüntü içerisinde aynı niteliksel karakteristiğe sahip alanları bulmak için geliştirilen metot ve fonksiyonlar Cvb isim alanında oluşturulmuşlar-

dır. Bu metot ve fonksiyonları kullanarak blob analizi yapabilmek için Cvb isim alanını aşağıdaki gibi derleyiciye önceden yüklemek gerekir.

```
using Emgu.CV.Cvb;
```

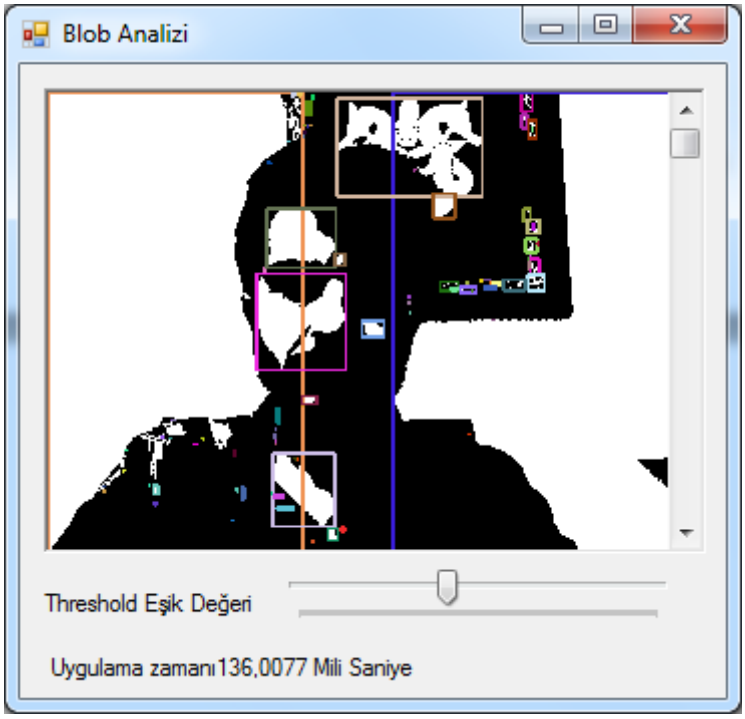
Geliştirilen Uygulama:

Aşağıdaki kod bloğunda giriş birimden alınan görüntüde aynı niteliksel değerler ile eşikleme değerinin üstünde değere sahip kapalı alanlar bir nesne olarak işaretlenmiş ve çıkış birimine anlık olarak iletilmişlerdir. Görüntüde işaretlenen her bir alan eşik değerinin üstünde aynı niteliksel özelliklere sahip farklı alanları göstermektedir.

```
Capture yakala = null;
bool calis = false;
DateTime Bas, Son;
private void Form19_Load(object sender, EventArgs e){
try{
yakala = new Emgu.CV.Capture();
yakala.FlipHorizontal = !yakala.FlipHorizontal;
}
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}
Application.Idle += Blob_Analiz;
calis = true;
}
void Blob_Analiz(object sender, EventArgs arguman){
try{
using (Image<Gray, Byte> kaynak = new Image<Gray, byte>(yakala.QueryFrame().Size))
using (Image<Gray, Byte> kamera = kaynak.Clone()){
Bas = DateTime.Now;
Image<Gray, Byte> threskamera = new Image<Gray, byte>(kamera.Size);
CvInvoke.cvCvtColor( yakala.QueryFrame(),
                     kaynak,
                     COLOR_CONVERSION.CV_BGR2GRAY);
CvInvoke.cvThreshold(kaynak,
```

```
                    threskamera,
                    trackBar1.Value,
                    255,
                    THRESH.CV_THRESH_BINARY);
CvBlobs kamBloblar = new CvBlobs();
CvBlobDetector BlobDedektor = new CvBlobDetector();
uint bulunanBlob = BlobDedektor.Detect(threskamera, kamBloblar);
/*Emgu.CV.Image<Bgr, Byte> blobgoruntu = BlobDedektor.DrawBlobs(threskamera,
                                    kamBloblar,
                                    CvBlobDetector.BlobRenderType.Default,
                                    0.5);*/
Image<Bgr, Byte> blobgoruntu = new Image<Bgr, byte>(threskamera.Size);
CvInvoke.cvCvtColor(threskamera, blobgoruntu, CO-
LOR_CONVERSION.CV_GRAY2BGR);
Random sayi = new Random();
foreach (CvBlob hedef in kamBloblar.Values){
blobgoruntu.Draw( hedef.BoundingBox,
                    new Bgr(    sayi.Next(127, 255),
                                sayi.Next(0, 128),
                                sayi.Next(0, 255)),
                    2);
}
ımageBox1.Image = blobgoruntu;
Son = DateTime.Now;
label2.Text = "Uygulama zamanı" + (Son - Bas).TotalMilliseconds.ToString() + " Mili
Saniye";
}       }
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}       }
```

Geliştirilen uyguluma çalıştırıldığında aynı niteliksel değerdeki alanlar farklı çerçeveler ile şekil 4.22'de işaretlenmişlerdir.



Şekil 4.22: Alınan görüntüde eş zamanlı blob analizi

4.18. Tesseract ile Otomatik Karakter Tanıma

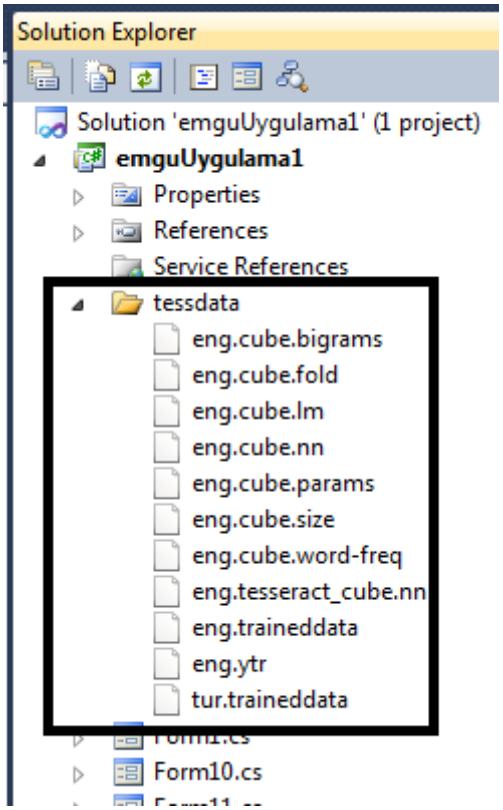
Tesseract HP şirketi tarafından karakter tanımak için geliştirilen farkli işletim sistemlerinin de kullandığı karakter tanıma motorudur. Tesseract OCR mevcut karakter tanıma algoritmaları arasında en iyi sonuçları üreten ve hızlı çalışan algoritmadır. Ayrıca açık kaynak kodlu olarak geliştirilmiştir.

Tesseract ile karakter tanıma, her dilin kullarına göre hazırlanmış olan dil paketleri ile kaynak görüntünün içindeki alanların karşılaştırılması ile eşleşen karakterlerin veri olarak dışarıya aktarılması mantığı ile çalışır. Tesseract sınıfından türetilen bir değişkenle görüntü üzerinde yapılan arama ile görüntüde ki karakterler çok hızlı bir şekilde tanınır. Karakter tanıma başarısı, ilgili dilin karakter tablolarının oluşturulma başarısına bağlıdır. Kuralları iyi tanımlanmış dil paketleri ile çalışmak çok daha iyi sonuçlar verecektir.

Tesseract ile otomatik karakter aramaları yapabilmek için derleyiciye EmguCV isim uzaylarından OCR isim uzayının derleyiciye aşağıdaki gibi bildirilmesi gerekir.

```
using Emgu.CV.OCR;
```

Tesseract ile otomatik karakter taramı yapabilmek için geliştirilecek uygulamaya dil paketinin şekil 4.23'de ki gibi eklenmelidir.



Şekil 4.23: Projeye eklenen Tesseract dil paketleri

Geliştirilen uygulamaya dil paketleri eklendikten son bu dil paketini aşağıdaki gibi Tesseract sınıfından bir değişkene yüklemek gerekmektedir.

```
private Tesseract karakter_tani = new Tesseract("tessdata", "eng",
                                        Tesseract.OcrEngineMode.OEM_DEFAULT);
```

tessdata: projeye eklenen dilpaketlerinin içinde olduğu kalsör. Çalışan uygulama dosyasından farklı bir konumda ise klasör yolunun doğru bir şekilde belirtilmesi gerekmetedir.

eng: hangi dil paketleri içerisinde hangi dil paketlerine erişileceğini gösteren parametredir.

OcrEngineMode: karakter tanıma kurallarının tanımlandığı metotdu belirtir..

- **OEM_CUBE_ONLY:** daha iyi sonuçlar için cube modunda karakter tarama sağlar fakat yavaş çalışır.
- **OEM_DEFAULT:** bu mod belirtildiğinde dil, diğer modlar gözönüne alınarak otomatik olarak tanımlanır ve karakterleri taranır.
- **OEM_TESSERACT_CUBE_COMBINED:** En doğru sonuçları bulma ve birleştirme için yöntem sağlar.

- **OEM_TESSERACT_ONLY:** görüntüyü en hızlı şekilde tarar.

Tesseract OCR, gri düzey tanımlanmış tek kanallı görüntülerde karakter araması yapabilmektedir. Sonuçların daha iyi olabilmesi için bir görüntü üzerinde karakter tanıma işlemini başlatmadan önce ilgili görüntünün dikkatle ikilileştirilmesi veya bir kenar bulma dedektörü ile giriş görüntüsünde kenarların belirlenmesi işlem başarısını olumlu yönde etkileyecektir.

Geliştirilen Uygulama:

Aşağıdaki kod bloğunda ingliz dili "eng" kurallarına göre yüklenen görüntülerde karakter tanıma işlemi farklı modlarda yapılabilmektedir.

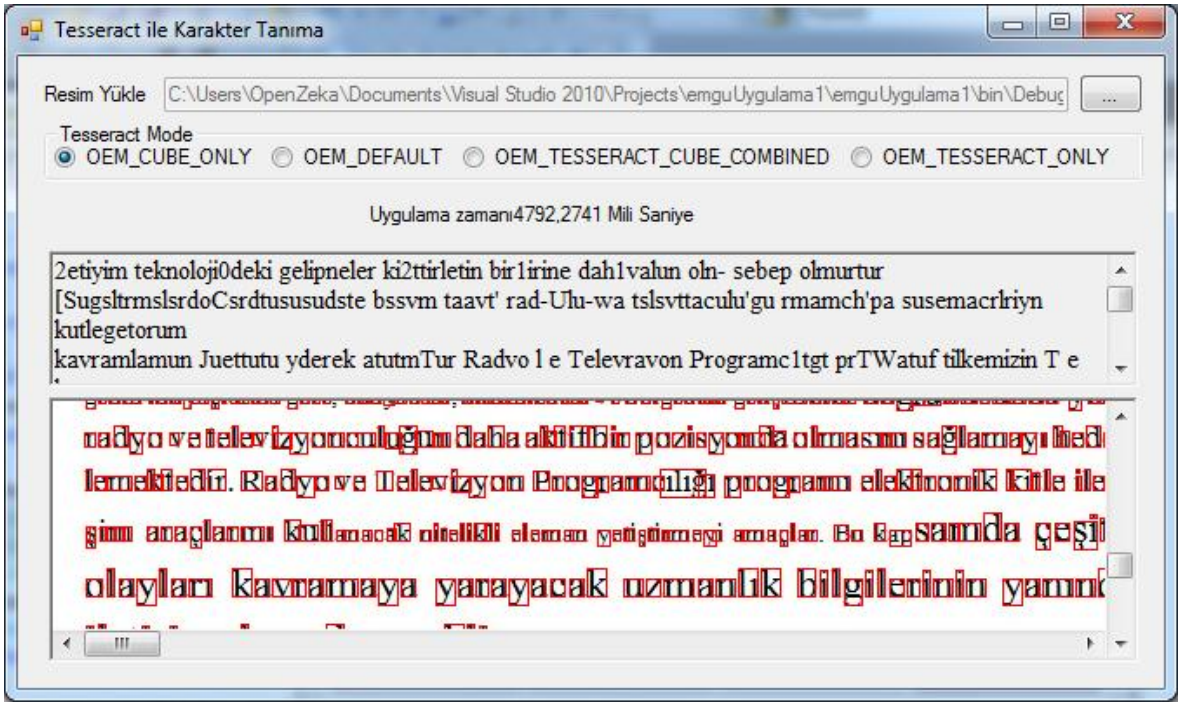
```
Image<Bgr, Byte> resim;
Bgr karakter_rengi;
Boolean KameraVeyaResim = false;
Capture yakala = null;
bool kameracalis = false;
DateTime Bas, Son;
private void Form18_Load(object sender, EventArgs e){
try{
karakter_tani = new Tesseract("tessdata",
                              "eng",
                              Tesseract.OcrEngineMode.OEM_DEFAULT);
openFileDialog1.Title = "Resim Yükle";
openFileDialog1.DefaultExt = "jpg";
openFileDialog1.Filter = "Jpg Dosyaları (*.jpg)|*.jpg|Jpeg Dosyaları (*.jpeg)|*.Jpeg|Tüm Dosyalar (*.*)|*.*";
}
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}       }
void Karakter_Bul(object sender, EventArgs e){
try{
if (kameracalis == true)
resim = yakala.QueryFrame();
```

```
using (Image<Gray, Byte> grii = new Image<Gray, byte>(resim.Size))
{
Bas = DateTime.Now;
CvInvoke.cvCvtColor(resim,
                    grii,
                    Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2GRAY);
CvInvoke.cvCanny(grii, grii, 50, 200, 3);
karakter_tani.Recognize<Gray>(grii);
Tesseract.Charactor[] karakterler = karakter_tani.GetCharactors();
foreach (Tesseract.Charactor k in karakterler){
resim.Draw(k.Region, karakter_rengi, 1);
}
ımageBox1.Image = resim;
richTextBox1.Text = karakter_tani.GetText();
Son = DateTime.Now;
label2.Text = "Uygulama zamanı" + (Son - Bas).TotalMilliseconds.ToString() + " Mili
Saniye";
}       }
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}       }
bool calisti = false;
string Dosya_Yolu;
private void button1_Click(object sender, EventArgs e){
DialogResult cevap = openFileDialog1.ShowDialog();
if (cevap == DialogResult.OK){
Dosya_Yolu = openFileDialog1.FileName;
textBox1.Text = openFileDialog1.FileName;
if (calisti == false){
radioButton1.Enabled = true;
radioButton2.Enabled = true;
radioButton3.Enabled = true;
```

```
radioButton4.Enabled = true;
radioButton2.Checked = true;
calisti = true;
Karakter_Bul(sender, e);
}       }       }

private void radioButton1_CheckedChanged(object sender, EventArgs e){
try{
karakter_tani = new Tesseract("tessdata",
                              "eng",
                              Tesseract.OcrEngineMode.OEM_CUBE_ONLY);
karakter_rengi = new Bgr(Color.Red);
if (kameracalis == false){
resim = new Image<Bgr, byte>(Dosya_Yolu);
Karakter_Bul(sender, e);
}       }
catch (Exception hata){
MessageBox.Show(hata.Message);
return;
}       }
```

Kodları verilen yukarıdaki uygulama çalıştırıldığında oluşan sonuç çıktısı dil kuralla-rının tanımlandığı dosyaların başarısına şekil 4.24'de verilmiştir.



Şekil 4.24: Tesseract ile otomatik karakter tanıma

5. SONUÇ

OpenCV, Intel firmasının görüntü işleme teknolojileri için standart oluşturma ve bu alanda yapılacak çalışma sayısını arttırmak amacıyla geliştirdiği açık kaynak kodlu bilgisayar görü kütüphaneleridir.

Günlük hayatta önemi ve uygulama alanı giderek artan görüntü işleme teknolojilerinden OpenCV kütüphanelerinin EmguCV sarmalayıcı kütüphaneleri aracılığı ile Studio.Net dillerinden C# ile temel görüntü işleme uygulamaları geliştirilmiştir. EmguCV, Studio.Net platformunun sistem kaynaklarını kullanım başarısı ile optimize OpenCV fonksiyonlarının gücünü birleştirmesine ek olarak görüntü işleme uygulamalarında daha hızlı sonuçlar verebilecek ve işlemleri pratikleştirecek araçlar ile görüntü işleme teknolojilerinde yeni bir güç olduğu uygulamalar ile görülmüştür.

Bir görüntüyü bölütlemenin temel basamaklarından biri olan eşikleme, OpenCV ile, dizi elemanları sabit seviyeli ve dizi elemanları Gauss ve Mean filtrelerine göre uyarlanabilen eşikleme ile görüntü işlemede bölütleme başarısını önemli ölçüde etkileyen eşikleme fonksiyonları sunar. Ayrıca optimize edilmiş konvülasyon matrisleri, farklı diyafram açıklıklarında Sobel, Laplace ve Canny operatörleri ile farklı seviyelerde kenar belirleme fonksiyonları, OpenCV'nin görüntü işleme başarısında ki en önemli etkenlerdendir. Temelde renklerden oluşan görüntülerin, kullanım amacına göre farklı renk uzaylarında ifade edilebilmeleri gerektiği noktasında, OpenCV'nin optimize dönüşüm bayrakları ile renk uzayları arasında dönüşüm işleminde işlem başarısı kaynak görüntü ile hedef görüntü arasında gözle görülemeyecek fark başarısı elde edilmiştir.

EmguCV ile geliştirilen kompleks yazılım ortamında geliştirilen uygulamalar kolaylıkla bir giriş birimine bağlanabilmekte ve bu birimden anlık elde edilen görüntüler kolaylıkla işlenip çıkış birimlerine aktarılabilmektedir. Anlık görüntülerin her birinin ayrı bir veri olarak ele alındığı eş zamanlı görüntü işleme sistemi; olay prosedürlerinde, timer nesnesine bağlanarak veya oluşturulacak bir döngü ile oluşturulabilir. Bu yönüyle EmguCV ortamında geliştirilen uygulamalarda tek yönlü yazılım geliştirme mantığından ziyade oluşturulacak sistemin yapısına en uygun çözüm yönteminin tercih edilmesinde esneklik sağlanmış olur.

Geliştirilen yazılım ortamında, Haar öznitelik sınıflandırıcı xml dosyaları ve Tesseract karakter tanıma sınıflandırıcı dosyaları gibi açık kaynak oluşturulan dosyalar ile uygu-

lamalar geliştirilmiştir. Açık kaynak kodlu dosyalardan oluşturulan bu uygulamaların algoritmik olarak daha basit bir programlama sağladığı ve performansın oluşturulan uygulama dışında hazırlanan niteliksel açık kaynak kodlu dosyalara bağlı olduğu görüldü.

EmguCV ile hazırlanmış yazılım ortamında geliştirilen her uygulama için hazırlanan sayaçlardan elde edilen sonuçlar, Matlab ortamında hazırlanmış aynı uygulamaların aynı görüntü verisinden elde edilen sonuçları ile karşılaştırıldığında elde edilen sonuçlar tablo 4.1'de verilmiştir. EmguCV ile geliştirilen birçok uygulama Matlab ortamında ki uygulamalardan daha hızlı olmasının yanında performans olarak daha etkin sonuçlar verdiği gözlemleniştir.

Geliştirilen Uygulama	EmguCV+OpenCV yaklaşık zaman (Mili Saniye)	Matlab yaklaşık zaman (Mili Saniye)
Resim Yükleme ve Görüntüleme	25	16
Sobel Kenar Bulma	20	20
Laplace Kenar Bulma	20	13
Canny Kenar Bulma	20	52
Gaussian Yumuşatma	10	20
Median Yumuşatma	20	20
Blur Yumuşatm	10	27
Blob Analizi	136	1520
Hough Circle Dönüşümü	64	45

Şekil 5.1: EmguCV ve Matlab uygulama zamanları

KAYNAKLAR

Açıkgöz, R., Doğan, S. ve Baneer, G., 1999. Raster Görüntüleri Yapısı, Görüntüleme Tekniklerinin Temelleri ve BITMAP Formatı, Harita ve Kadastro Mühendisliği Dergisi, S 86, s 61-80, Ankara.

Akar, F., 2009. Şablon eşleşme yöntemi ile plaka tanıma ve değerlendirme sistemi, Doktora Tezi, Atatürk Üniversitesi Fen Bilimleri Enstitüsü, Erzurum.

Al-Karawi, M. K., 2012. İkili görüntü kullanarak imza steganografi modellerinin geliştirilmesi, *Yüksek Lisans Tezi*, Gazi Üniversitesi Bilişim Enstitüsü, Ankara.

Altıntaş, V. ve Yegenoğku, 2011. Görüntü işlemede seri ve paralel programlamanın performansı, 6th International Advanced Technologies Symposium (IATS'11), Elazığ, Turkey, 16-18 May.

Arslan, E., 2011. Hücresel Sinir Ağ Sistemleri Kullanarak Hareketli Nesnelerin Görüntü İşleme Uygulamaları, *Doktora Tezi*, İstanbul Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

Arslan, E., Arık, S., 2010. Hareketli görüntüde kenar belirleme algoritmasının analog hücresel sinir ağı ve sayısal işaret işleme işlemcileri üzerinde uygulaması, *Mühendislik Bilimleri Dergisi*, 1, 1-7.

Avcı, A., 2006. Wavelet dönüşümü ile doku öznitelikleri çıkarılan görüntülerin rezonans algoritması kullanılarak bölütlenmesi, *Yüksek Lisans Tezi*, Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü, Trabzon.

Aybar, E., 2008. Sobel İşleci Kullanılarak Renkli Görüntülerde Kenar Bulma, Afyon Kocatepe Üniversitesi Fen Bilimleri Dergisi, Afyon.

Bayram, B., 2013. Sayısal Görüntü İşleme. http://www.yildiz.edu.tr/~bayram/sgi/saygi.htm, 22 Ağustos 2013.

Bradski, G. and Kaehler, A., 2008. Learning OpenCV, O'Reilly Media Inc, Sebastopol.

Chaney, M., 2013. Brightness, Contrast, Saturation and Sharpness, Steve's Digicams, http://www.steves-digicams.com/knowledge-center/brightness-contrast-saturation-and-sharpness.html#b, 26 Ağustos 2013.

Çulha, S., 1996. Sayısallaştırılmış bilgisayarlı tomografi ve manyetik rezonans tomogramları üzerinde görüntü işleme tekniği uygulmaları, *Yüksek Lisans Tezi*, Ankara Üniversitesi Fen Bilimleri Enstitüsü, Ankara.

Er, O., 2011. Görüntü işleme teknikleri kullanarak elma tasnifleme, *Yüksek Lisans Tezi*, Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü, Isparta.

Gonzales, R. C. and Woods, R. E. 2002. Digital Image Processing, Prentice-Hall Inc, New Jersey.

Hardeberg, J. Y.,1999.Acquisition and reproduction of color image: Colorometric and multispectral approaches, PhD thesis, Ecole Nationale Superieure des Telecommunications, Paris.

Hazer, M., 2007. Bulanık topolojiye dayalı kenar bulma algoritması, *Yüksek Lisans Tezi*, Anadolu Üniversitesi Fen Bilimleri Enstitüsü, Eskişehir.

İncearık, M. E., 2011.Grafik-tasarım rehberi, Kodlab Yayım, İstanbul.

Karabatak, E., 2010. Nötrozofi yaklaşımı ile renkli görüntü bölütleme, *Yüksek Lisans Tezi*, Fırat Üniversitesi Fen Bilimleri Enstitüsü, Elazığ.

Karabiber, F., 2009. Analog Hücresel Sinir Ağı İşlemcisi Kullanarak Gerçek Zamanlı Görüntü İşleme Uygulamaları, *Doktora Tezi*, İstanbul Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

Karakoç, M., 2011. Görüntü İşleme Teknikleri ve Yapay Zeka Yöntemleri Kullanarak Görüntü İçinde Görüntü Arama, *Yüksek Lisans Tezi*, Pamukkale Üniversitesi Fen Bilimleri Enstitüsü, Bursa.

Karsan, S., 2008. Sayısal Görüntü ve Sayısal Görüntü İşlemenin Tasarıma Etkisi, *Yüksek Lisans Tezi*, Mimar Sinan Güzel Sanatlar Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

Katırcıoğlu, F., 2007. Renkli görüntülerin bağıntı matrisine dayalı ayrıştırılması ve kenar algılama, Yüksek Lisans Tezi, Düzce Üniversitesi Fen Bilimleri Enstitüsü, Düzce.

Kazan, S., 2013. Bilgisayar Görmesi, Kenar Bulma, http://webcache.googleusercontent.com/search?q=cache:FnlyShhwt3IJ:https://dosya.sakarya.edu.tr/Dokumanlar/2013/422/219777363_hf7_bg_kenar_bulma.pptx+kenar+bulma&cd=4&hl=tr&ct=clnk&gl=tr , 24 Ağustos 2013.

Khurshed, E. S., 2009. Yüz tanımada aydınlanmanın etkisinin uyarlanır histogram eşitleme ile azaltılması, *Yüksek Lisans Tezi*, Gazi Üniversitesi Fen Bilimleri Enstitüsü, Ankara.

Kurtulmuş, F., 2012. Olgunlaşmamış şeftali meyvesini doğal bahçe koşullarında alınmış görüntülerde görüntü işleme teknikleri ve yapay sınıflandırıcılarla saptayarak sayan algoritmaların geliştirilmesi, *Doktora Tezi*, Uludağ Üniversitesi Fen Bilimleri Enstitüsü, Bursa

Morrison, M., 2005. 24 Saatte oyun programlama (çev. Koray Al), Alfa Basım Yayım, İstanbul.

Özmercan, C., 2013. Renk olayı, Cenk Özmercan, http://cenkozmercan.wordpress.com/2010/02/04/renk-olayi/, 26 Ağustos 2013.

Saphiro, L. and Stockman, G., 2001. Computer Vision, Prentice-Hall, New Jersey.

Siyah, B., 2013. Görüntü Filtreleme Uygulamaları ve Amaçları (Matlab). http://www.bulentsiyah.com/goruntu-filtreleme-uygulamalari-ve-amaclari-matlab/ , 24 Ağustos 2013.

Soytürk, M. A., 2005. Sayısal kenar çıkarma ve yapay sinir ağları yardımıyla araç tanıma, *Yüksek Lisans Tezi*, Erciyes Üniversitesi Fen Bilimleri Enstitüsü, Kayseri.

Şahinbaşkan, T., 2013. Renk Evren Modellerinin Matbaacılık Sektöründeki Kullanım Alanları, CMYKlinik, http://www.cmyklinik.com/cms/index.php?option=com_content&view=article&id=56:renk-evren-modellerinin-matbaaclk-sektoeruendeki-kullanm-alanlar&catid=34:bask-oencesi&Itemid=53, 27 Ağustos 2013.

Taşkın, D., 2007.Sıkıştırılmış video akımının düzensiz haritalar ve başlangıç kodlarına dayalı şifrelenmesi, *Doktora Tezi*, Trakya Üniversitesi Fen Bilimleri Enstitüsü, Edirne.

Toyran, M., 2008. Düşük çözünürlüklü görüntülerden süper çözünürlüklü görüntüler oluşturma, *Yüksek Lisans Tezi*, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

URL-1, http://tr.wikipedia.org/wiki/Viz%C3%BCel_sistem Vikipedi Özgür Ansiklopedi, Vizüel Sistem. 22 Ağustos 2013.

URL-2, http://mr.nedir.com/ Nedir?com, Mr Nedir. 22 Ağustos 2013.

URL-3, http://www.nedirnedemek.com/reduced-brightness-nedir-reduced-brightness-nedemek , Nedir Ne Demek, reduced brightness nedir, reduced brightness ne demek?. 23 Ağustos 2013.

URL-4, http://kontrast.bunedir.org/ ,Kontrast nedir, Bu Nedir, 23 Ağustos 2013.

URL-5, http://fotografbilgimerkezi.com/histogramlar/.html , Histogramlar, Fotograf Bilgi Merkezi, 26 Ağustos 2013.

URL-6, http://tr.wikipedia.org/wiki/RGB_renk_uzay%C4%B1, RGB renk uzayı, Vikipedi, 26 Ağustos 2013.

URL-7, http://tr.wikipedia.org/wiki/CMYK_renk_uzay%C4%B1, CMYK renk uzayı, Vikipedi, 26 Ağustos 2013.

URL-8, 2013. http://www.highspeed.pro/forum/konu-disi/isik-ve-renk-nedir-73594.html, Renk Modeli, Renk Uzayı ve Gamut, 27 Ağustos 2013.

URL-9, 2013. http://prezi.com/3nhzpbvpmxbn/ycbcr-color-model/, YCbCr Color Model, Prezi, 27 Ağustos 2013.

URL-10, 2013. http://www.couleur.org/index.php?page=transformations, Color Spaces, Maido Tech, 27 Ağustos 2013.

URL-11,2013 Photoshop CS4'ü Kullanma Klavuzu, http://www.cmyklinik.com/cms/index.php?option=com_content&view=article&id=56:renk-evren-modellerinin-matbaaclk-sektoeruendeki-kullanm-alanlar&catid=34:bask-oencesi&Itemid=53, 27 Ağustos 2013.

URL-12, 2013. http://www.nvidia.com.tr/object/cuda-parallel-computing-tr.html, Cuda Paralel Hesaplama, NVIDIA, 28 Ağustos 2013.

URL-13, 2013. http://www.uludagsozluk.com/k/intel-integrated-performance-primitives/, İntel integrated performance primitives, Uludağ Sözlük, 28 Ağustos 2013.

URL-14, 2013. http://www.aishack.in/2010/07/opencv-vs-vxl-vs-lti-performance-test/, OpenCV vs VXL vs LTI: Performance Test, Al Shack, 28 Ağustos 2013.

URL-15, 2013. http://www.baymerakli.com/2012/02/lti-nedir.html, LTI Nedir?, Güncel Bilgi Blogu, 28 Ağustos 2013.

URL-16, 2013. http://www.cognotics.com/opencv/docs/1.0/ref/opencvref_cv.htm, OpenCV 1.1 Documantation, Cognotics, 29 Ağustos 2013.

URL-17, 2013. http://www.csharpnedir.com/articles/read/?id=329, Linux Platform üzerinde Mono ile .Net, C#nedir?com, 31 Ağustos 2013.

URL-18, 2013. http://www.emgu.com/wiki/index.php/Main_Page, EmguCV, EmguCV, 04 Eylül 2013.

Yalman, Y., 2010. Sayısal görüntüler için histogram temelli ve gizleme yöntemi ve uygulama yazılımı, *Doktora Tezi*, Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, Kocaeli.

Yılmaz, İ., 2002. Renk sistemleri, renk uzayları ve dönüşümler, Jeodezi ve Fotogrametri Mühendisliği Öğretiminde 30. Yıl Sempozyumu, Selçuk Üniversitesi, Konya ,Ekim 16-18.

Young, I. T., Gerbrands, J. J., Van Vilet, L. J., 1998. Fundamenrals of Image Processing, Delf Universty of Technology, Netherlands.

ÖZGEÇMİŞ

02.03.1985 Şırnak Cizre doğumluyum. İlk ve orta öğrenimimi Cizre'de tamamladıktan sonra 2004 – 2008 yılları arasında Fırat Üniversitesi Teknik Eğitim Fakültesi Elektronik Bilgisayar Eğitimi Bölümü Bilgisayar Öğretmenliği Programında lisans eğitimini tamamladım. Lisans eğitimini takiben 2008 - 2010 yılları arasında Diyarbakır Şehmus Sultan Tatlıcı Lisesinde bilgisayar öğretmeni olarak görev aldıktan sonra 2010 Mart ayı itibari ile Şırnak Üniversitesi Cizre Meslek Yüksekokulu Bilgisayar Programcılığı programında öğretim görevlisi olarak görev yapmaktayım.