T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖRÜNTÜ İŞLEME YÖNTEMLERİ KULLANILARAK GÖZDEKİ DAMARLARIN TESPİT EDİLMESİ

YÜKSEK LİSANS TEZİ

Serhat OKUR

Anabilim Dalı: Elektronik ve Bilgisayar Eğitimi
Programı: Bilgisayar Sistemleri Eğitimi

Tez Danışmanı: Doç. Dr. Erkan TANYILDIZI

ARALIK-2015

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖRÜNTÜ İŞLEME YÖNTEMLERİ KULLANILARAK GÖZDEKİ DAMARLARIN TESPİT EDİLMESİ

YÜKSEK LİSANS TEZİ

Serhat OKUR
112131102

Anabilim Dalı: Elektronik ve Bilgisayar Eğitimi
Programı: Bilgisayar Sistemleri Eğitimi

Tez Danışmanı: Doç. Dr. Erkan TANYILDIZI (F.Ü.)

Tezin Enstitüye Verildiği Tarih: 29.12.2015

ARALIK-2015

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖRÜNTÜ İŞLEME YÖNTEMLERİ KULLANILARAK GÖZDEKİ DAMARLARIN TESPİT EDİLMESİ

YÜKSEK LİSANS TEZİ

Serhat OKUR
112131102

Tezin Enstitüye Verildiği Tarih: 27.11.2015
Tezin Savunulduğu Tarih: 18.12.2015

Tez Danışmanı: Doç. Dr. Erkan TANYILDIZI (F.Ü.)
Diğer Jüri Üyeleri: Prof. Dr. Abdulkadir ŞENGÜR
Doç. Dr. Davut HANBAY (İ.Ü.)

ARALIK-2015

ÖNSÖZ

İnsanlarda ortaya çıkan hastalıkların bir bölümü, gözdeki retina tabakasında bulunan kan damarlarının yapısının bozulmasına neden olmaktadır. Bu nedenle retinadaki kan damarlarının yapısal bozukluklarının önceden teşhisi bazı hastalıkların tedavisine olanak sağlamaktadır. Aynı zamanda aynı hastadan değişik zamanlarda alınan retina görüntülerinin karşılaştırılarak aralarındaki farklılıkların izlenmesi ile hastalığın teşhisi ve takibi yapılabilmektedir. Retinadaki kan damarlarının çıkarılması için çeşitli yöntemler bulunmaktadır. Bu tez çalışmasında damar çıkarma için kullanılan yöntemlerden bahsedilmiş ve görüntü işleme kütüphanesi OpenCV kullanılarak gözdeki retina tabakasında bulunan kan damarlarının çıkarılması uygulaması açıklanmıştır.

Bu çalışma esnasında bana her türlü hoşgörü, destek ve bilgisiyle katkıda bulunan değerli danışmanım Doç. Dr. Erkan TANYILDIZI' na, uygulama geliştirme esnasında yardımını esirgemeyen tüm hocalarıma, her zaman bana destek veren ve yanımda olan çok kıymetli aileme, arkadaşlarıma teşekkürü bir borç bilirim.

Serhat OKUR

ELAZIĞ- 2015

İÇİNDEKİLER

ÖZET

Bu tez çalışması çeşitli hastalıklardan dolayı gözdeki retinada bulunan kan damarlarındaki değişimi kolaylıkla izleyip hastalıkların erken teşhisi ve tedavisi için görüntü işleme yöntemleri kullanılarak damarların çıkarılması ile ilgilidir. Tezin amacı hastalıkların erken teşhisi ve tedavisi için retinadaki kan damarlarının eksiksiz bir şekilde çıkarılmasıdır. Tez için gerçekleştirlen uygulamada retina görüntülerindeki kan damarlarının tespiti için şekil hatlarının belirlenmesi yöntemi kullanılmıştır. Şekil hatlarının belirlenmesi yönteminde ilk olarak aynı renk ve şiddete sahip piksel değerleri yani kontürler vektörel bir matris içerisinde toplanmaktadır. Daha sonra bu vektrörel matris içerisinde bulunan kontürlerin alanları hesaplanarak damar olmayacak kadar küçük alanlar elenmektedir. Damar olmayan alanlar elendikten sonra geriye kalan kontürler çizilerek damarlar tespit edilmektedir.

Damar çıkarma için birçok yöntem kullanılmaktadır. Damar çıkarma yöntemleri kullanılarak yazılan birçok program genellikle MATLAB kodları ile yazılmıştır. Uygulamalarda hız önemli olduğu için uygulama geliştirilirken makine diline en yakın üst düzey programlama dilleri tercih edilmiştir [1].

Çalışmada açık kaynak kodlu görüntü işleme kütüphanesi olan OpenCV görüntü işleme kütüphanesi damar çıkarma yöntemlerine uygulanarak program yazılmıştır. OpenCV görüntü işleme kütüphanesinde hazır bulunan filtreler fundus kamera ile çekilmiş göz resimlerine uygulanarak sonuçlar incelenmiştir.

Anahtar Kelimeler: Görüntü İşleme, Görüntü Filtreleme, OpenCV, Kenar Çıkarma ve Sınırlar, Şekil İşleme, Nesne Tanıma, Öznitelik Çıkarma

SUMMARY

Identifying Vessels In The Eye Using Image Processing Methods

This thesis is concerned with the extraction of the vessels using image processing methods for early diagnosis and treatment of diseases and to be able to easily monitor changes in the blood vessels in the eye in the retina due to various diseases. The aim of the thesis is a complete extraction of the blood vessels for the early diagnosis and treatment of diseases of the retina. The method of identifying shape line has been used in the application carried out for the thesis in which blood vessels in retinal images are determined. Firstly, in the method of determining the shape line, pixel intensity values that have the same color and contours are collected in a vector matrix. Then, the area of the contours located within this vector matrix are calculated, and those areas too small to be vessels are eliminated. Vessels are identified by drawing the remaining contours after areas that are not suitable for being vessel are eliminated.

Various methods are used to extract vessels. Many programs used for vessel extraction methods are usually written in Matlab code.

Speed is important in applications. For that reason, the upper level programming languages are preferred while developing an application [1].

In this study, an open source image processing library OpenCV has been written by applying the method of vessel extraction. The filters in OpenCV image processing library that are ready to be used have been applied to eye images taken with fundus camera. After that, the results have been analyzed.

Keywords: Image Processing, Image Fıltering, OpenCV, Extraction of Edges and Borders, Figure Processing, Object Recognition, Extraction of Attributes

ŞEKİLLER LİSTESİ

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

KISALTMALAR LİSTESİ

OCR : Optik Karakter Tanıma

DR : Diyabetik Retinopati

OCT : Optic Coherence Tomogrphy

FFA : Fundus Fluorescien Anjiografi

LPI : Liner per Inch

DPI : Dots per Inch

HSV : Hue-Saturation-Value yani renk tonu – doygunluk – değer,

HSB : Hue – Saturation - Brightness yani renk tonu – doygunluk – parlaklık,

HIS : Hue – Saturation - Intensity yani Renk tonu – Doygunluk – Yoğunluk,

HLS : Hue – Luminance – Saturation yani Renk tonu – Parlaklık - Doygunluk,

RGB : Red (Kırmızı) ,Green (Yeşil), Blue (Mavi)

OpenCV : Open Source Computer Vision Library

IPP : Gömülü Performans Tipleri

MLL : Machine Learning Library

Sk : Histogram Eşikleme

CV : Computer Vision

PCA : Principle Components Analysis

LDA : Linear Discriminant Analysis

YSA : Yapay Sinir Ağları

1. GİRİŞ

Gelişen teknoloji ve karmaşıklaşan dünyada insanlar artık hastalığa sebep olan daha fazla tehditle karşı karşıyadır. Bu tehditlerden ötürü insanlarda birçok hastalık kendini göstermektedir. İnsanlarda ortaya çıkan hastalıklardan bir bölümü, gözdeki retina tabakasında bulunan kan damarlarının yapısının bozulmasına neden olmaktadır. Bu nedenle retinadaki kan damarlarının yapısal bozukluklarının önceden teşhisi bazı hastalıkların tedavisine olanak sağlamaktadır. Retina görüntülerinden kan damarlarının otomatik olarak çıkarılması diyabetik retinopati, glakom, damar sertliği, retinal alter tıkanıklığı vb. hastalıkların bilgisayar destekli tanı ve tedavisinde önemli bir adımdır [2]. Bu konuda insanların ihtiyaçlarını karşılayan, yaşamını kolaylaştıran çalışmalardan bazıları görüntü işleme uygulamasıdır [3].

Görüntü işleme, algılanan görüntüden yeni bir görüntü elde edilmesi için yapılan işlemlerdir [4]. Görüntü işlemenin kullanım amaçlarından bazıları, görüntüyü iyileştirme, onarma, sıkıştırma, analiz etme ve tanımadır [5]. Görüntü işlemenin uygulama alanları oldukça geniş olup, kullanıldığı alanlarda insanlığa sağladığı yararlar oldukça fazladır. Görüntü işleme: tıp (hastalık/kırık belirleme, nodül tespiti, damar belirleme, nesne sayma ve MR, ultrason, gama ışını, tomografi görüntülerinin iyileştirilmesi), uzay çalışmaları (gezegenler, uydular, gökyüzü olayları), uzak yeryüzü kaynakları araştırmaları (uydu görüntüleri), güvenlik (yüz/parmak izi tanıma, hareket tespiti), mühendislik (kalite kontrol), film efektleri, yayıncılık, spor, sanat, belgelerin sayısallaştırılması (Optical Cahracter Recognition/Optik Karakter Tanıma (OCR), kütüphaneler), askeri uygulamalar (hedef tespiti, insansız hava araçları, gece görüşü) gibi birçok alanda kullanılabilmektedir [6]. Görüntü işleme teknikleri ile yapılan çalışmalar bilgisayar bilimlerinin insanoğlunun hayatını ne derece kolaylaştırdığını oraya koymaktadır.

Bir görüntünün kullanışlı bir şekilde işleme tabi tutulabilmesi için öncelikle bu görüntünün bilgisayar tarafından anlaşılabilecek hale dönüştürülmesi gerekmektedir. Bu dönüşüm işlemine görüntünün sayısallaştırılması denir [4]. Görüntü sayısallaştırıldıktan sonra çeşitli algoritmalar kullanılarak görüntünün istenilen hale gelmesi sağlanabilir. Bu algoritmaları uygulamak için çeşitli yazılım dilleri ve kütüphaneler mevcuttur. OpenCV, SimpleCV, GNU Octave, Aforge.NET, VTK, FIJI, OpenGL, GIMP, Inkspace, Blender görüntü işleme alanında kullanılan bazı genel kullanıma açık kütüphane ve araçlardır [5].

Bu tez çalışmasının amacı gözdeki retina görüntülerinden kan damarlarının ayrıştırılarak hastalık teşhisinin daha kolay yapılmasını sağlamaktır.

Literatürde retina görüntülerinden kan damarlarının ayrıştırılması için çeşitli çalışmalar yapılmıştır.

Damarların çıkarılması için kullanılan algoritmalar genel olarak iki grupta incelenmektedir [7]. Birinci grup algoritmalar kural tabanlı olarak isimlendirilmekte ve Damar İzleme [8,9], Uyum Süzgeç Tepkisi [10,11] ve Morfoloji Tabanlı yöntemleri [12-14] kapsamaktadır. İkinci gruptaki algoritmalar ise danışmanlı algoritmalardır ve genellikle piksel sınıflandırması sinir ağı modelleri ve özel tasarlanmış sınıflandırıcılar kullanırlar [15]. Piksel sınıflandırması için KNN (k En Yakın Komşuluk) [15], Bayes Kuralı [16], Destek Vektör Makineleri [17], YSA (Yapay Sinir Ağları) [18] gibi sınıflandırıcılar kullanılmaktadır.

Damar çıkarma konusunda yapılan çalışmalardan bir tanesi Chaudhuri ve arkadaşları tarafından yapılmış ve bu çalışmada damar çıkarma uygulaması için eşleştirme süzgeci (Matched Filter) önermişlerdir [10].

Hoover ve arkadaşları 2000 yılında yaptıkları çalışmalarında retinal kan damarlarının çıkarılması için çalışmasında önerilen iki boyutlu eşleştirme süzgecinin uygulanmasından sonra uygun eşik değerini bulmak için eşik algılama yöntemi önermişlerdir [11].

Martinez-Perez ve arkadaşları 2007 yılında yaptıkları çalışmada retina üzerinde bulunan damarların farklı genişliklerde olması nedeniyle çok ölçekli özellik çıkarmaya dayalı bir yöntem önermişlerdir. Hessian matris ve yön büyüklük değerlerinin yerel maksimum değerleri kullanılarak retinal kan damarları belirginleştirilmiş, daha sonra da özel bir bölge büyütme yöntemi ile damarlar çıkarmışlarıdır [12].

Soares J.V.B. ve arkadaşları 2006 yılındaki çalışmalarında piksel parlaklık değerleri üzerinde farklı ölçeklerde Gabor-Dalgacık dönüşümü uygulamıştır. Daha sonra elde edilen farklı ölçekteki Gabor-Dalgacık dönüşüm çıktıları özellik olarak kullanılarak tüm görüntü Bayes Sınıflandırıcı uygulanarak damar ya da damar olmayan bölgelere ayrılmıştır [16].

Ricci ve arkadaşları damar çıkarma işlemi için daha önce mamografi görüntüleri için kullanılan çizgi operatörünü kullanmışlar ve bu operatörün görüntü çıktısını Destek Vektör Makineleri kullanan sınıflandırıcıya giriş olarak vermişlerdir [17].

Marin ve arkadaşları 2011 yılındaki çalışmalarında retinal görüntülerdeki piksellerin gri seviye özellikleri ile moment değişmezleri tabanlı özellikler kullanarak sinir ağı tasarlamışlardır [18].

Jiang ve Mojon çalışmalarında adaptif yerel eşiklemeye dayalı bir yöntem önermişlerdir. Bu yönteme göre belli eşik aralıklarına göre tüm görüntü her bir eşik aralığına göre siyah beyaz görüntüye çevrilmiş ve elde edilen sonuç doğrulama prosedürü ile damar olabilecek yapılar seçilmiştir. Daha sonra ayrı ayrı eşik değerlerinden elde edilen damar yapıları birleştirilerek damar görüntüsü elde edilmiştir [19].

Jeyasri, Subathra ve Annaram 2013 yılında yaptıkları çalışmalarında retinal kan damarlarının çıkarılması için öncelikle görüntüyü eğricik dönüşüm kullanarak iyileştirmiş ve kenarların tespiti için orjinal görüntüden açma işlemi uygulanmış görüntüyü çıkarma işlemini uygulamışlardır. Damar olmayan kenarların da çıkarılması için morfolojik açma ve lenght filtresi kullanımını önermişlerdir [20].

Raja, Vasuki ve Kumar 2014 yılındaki çalışmalarında öncelikle retina görüntüsü üzerindeki kan damarlarının belirginleşmesi için adaptif histogram eşikleme uygulamış ve renkli görüntüyü renk kanallarına (kırmız, yeşil, mavi) ayırarak yeşil kanalı damar tespiti için kullanmışlardır. Daha sonra morfolojik işlemlerden yayma ve aşındırma işlemini uygulamışlardır. Son olarak SVM sınıflandırıcı uygulanarak damar ya da damar olmayan bölgeler ayrılmıştır [21].

Bu tez çalışmasındaki uygulama gerçekleştirilirken şekil hatlarının belirlenmesi yöntemi kullanılmıştır. Şekil hatları belirlendikten sonra damar olmayacak kadar küçük alanların elenmesi işlemi gerçekleştirlmiş ve son olarak damar olan bölgeler çizilerek gösterilmiştir.

Bu tez çalışmasının ikinci bölümünde göz ve diyabetik retinopati hakkında, üçüncü bölümünde görüntü ve görüntüyü oluşturan bileşenler hakkında, dördüncü bölümde ise görüntü işleme hakkında bilgi verilmektedir. Beşinci bölümde damar çıkarma yöntemlerini uygulamak için OpenCV Kütüphanesinin Visual Studio'ya entegresi ve kütüphanenin bazı fonksiyonlarının açıklaması yapılmıştır. Son olarak altıncı bölümde ise damar çıkarma uygulamaları gösterilmektedir.

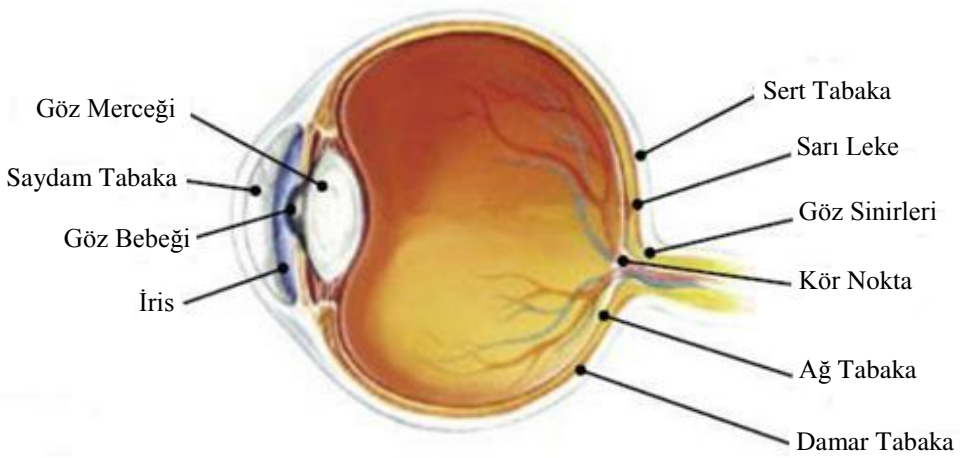
2. GÖZ

Göz, görme duyu organımızdır. İnsan gözü çok uzak mesafelerdeki nesneleri görebilme ve nesneleri sayısız renk ve şekil bilgisine göre ayırt edebilme yeteneğine sahiptir.

Göz, koruyucu yapılar ve görme yapıları olarak iki kısımdan oluşmaktadır. Gözümüzde bulunan ve gözü koruyan yapılar incelendiğinde kirpikler, göz kapakları, göz kasları ve gözyaşı bezlerinden oluşmaktadır [22].

1. Sert tabaka
2. Damar tabaka
3. Ağ tabaka (retina)

olmak üzere üç tabakadan oluşmaktadır.



Şekil 2.1. Göz ve gözün yapısı

2.1. Sert Tabaka

Gözün en dışında bulunan beyaz renkli yapıdır. Gözümüzü dış etkilerden korur. Göz kasları bu tabakaya bağlıdır. Sert tabaka gözün ön kısmında farklılaşarak saydam tabakayı (kornea) oluşturur [23]. Kornea ince kenarlı mercek gibi davranarak göze gelen ışığı kırıp irise ulaşmasını sağlar.

2.2. Damar Tabaka

Gözü besleyen kan damarları burada bulunur. Damar tabakanın farklılaşması ile iris ve göz bebeği oluşmaktadır. İris; gözün renkli olan kısmıdır. Düz kaslardan meydana gelmiştir. İrisin görevi göze girecek olan ışık miktarını ayarlamaktır. İrisin ortasında göze ışık girmesini sağlayan göz bebeği bulunmaktadır. Göz bebeği; göze giren ışığın merceğe ulaşmasını sağlar ve göze girecek olan ışık miktarını ayarlar. Göz bebeği az ışıkta büyüyüp çok ışıkta küçülmektedir. Göz merceği; canlıdır, saydamdır ve ince kenarlıdır. Göze giren ışığı kırarak görüntünün ağ tabakaya (retina) ters olarak düşmesini sağlar [22].

2.3. Ağ Tabaka (Retina)

Gözün en içinde bulunan kısımdır. Yapısında çok sayıda duyu almacı ve sinirler bulunmaktadır. Retinada bulunan duyu almaçları oluşan görüntüyü algılayarak sinirler aracılığıyla beyne iletir. Bu tabakada sarı leke ve kör nokta bulunmaktadır. Sarı leke; duyu almaçlarının en yoğun olduğu, görüntünün en net alındığı kısımdır. Kör nokta; görme sinirlerinin gözümüzden çıktığı kısımdır ve burada duyu almaçları bulunmaz [22].

Hastalıklar retinada dokusunu besleyen kan damarlarının etkileyerek işlevini doğru bir şekilde yerine getirmesine engel olabilir ve körlüğe kadar ulaşan sonuçlar meydana gelebilir. Retina rahatsızlıklarını etkileyen çok sayıda hastalık bulunur. Bu hastalıklar içerisinde en önemli olanları şeker, tansiyon ve kan hastalıklarıdır [24].

Retinadaki kan damarlarından görünen rahatsızlıklardan biri de Diyabetik Retinopati (DR) denilen hastalıktır. Diyabetik Retinopati hastalığı retinadaki kan damarlarının yapısında değişimler meydana getirmektedir. Retina kan damarlarında meydana gelen bu yapısal değişimlere bakılarak kişinin Diyabetik Retinopati hastalığının teşhisi yapılabilir.

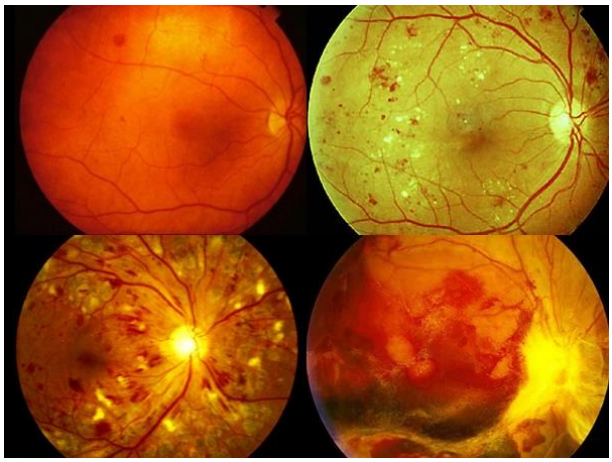
2.4. Diyabetik Retinopati

Yüksek kan şekeri nedeniyle gözün retina tabakasındaki damarların hasar görmesi sonucunda ortaya çıkan bir hastalıktır [25]. Görmek için ışığın hiçbir engel ile karşılaşmadan retinaya ulaşabilmesi gerekir. Diyabetik Retinopati hastalığı ile retinadaki kan damarları tıkanır, sızdırır veya rastgele büyür. Bu sonuçlar ise ışığın retinaya ulaşmasının önünde bir

engel olmaktadır ve tedavi edilmediği takdirde görme yetisinin kaybolmasına kadar kötü sonuçlara varılabilir.

Diyabetik Retinopati proliferatif ve nonproliferatif olarak iki şekilde gerçekleşir. Hastalığın başlangıç esnasında nonproliferatif evre görünür. Bu evre bozuk damarlardan sıvı sızması ve retinada kanamalara sebep olur. Hastaların görme yetisi bu evrede genel olarak etkilenmez. Bu nedenle diyabeti olan her hastanın, görme yetisini kaybetmeden yılda bir kez göz muayenesi yaptırması gerekmektedir.

Tehlikeli olan evre ise proliteratif evredir. Bu evrede retina tabakasındaki damarların ileri derecede bozulmasına bağlı beslenemeyen belgeler oluşur ve böylece yeni damarlar gelişir. Gelişen bu damarlar çok ince ve kırılgandır. Kendiliğinden göz içerisinde kanama yapabilir.



Şekil 2.2. Diyabetik retinopati evreleri: Hafif non-proliferatif diyabetik retinopati (sol üst), orta derece non-proliferatif diyabetik retinopati (sağ üst), şiddeti non-proliferatif diyabetik retinopati (sol alt), proliferatif diyabetik retinopati (sağ alt) [26]

2.4.1. Diyabetik Retinopati Tespiti

Erken safhalarda diyabetik retinopatinin belirgin semptomları yoktur. Bu nedenle kişi hastalık ilerleyene kadar diyabetik retinopati olup olamadığını bilemeyebilir. Hastalık ilerleyince bazı insanlarda;

- Karanlık yüzler noktalar veya örümcek benzeri şekiller görmek
- Kara lekeler ve benekler görmek
- Bulanık görmek

gibi belirti olduğu görülmektedir [27].

Diyabetik Retinopatinin erken teşhisi çok önemlidir. Erken tespit edilen diyabetik retinopatinin tedavisinin başarı ihtimali artmaktadır. Bunun için yılda bir kez göz muayenesine gidilip taratılmalıdır.

Diyabetik retinopati, hastanın göz bebeği damla ile genişletildikten sonra göz dibi muayenesi sonrasında teşhis edilir. Detaylı bilgi OCT (Optic Coherence Tomography) ve göz anjiusu (FFA: Fundus Fluorescien Anjiografi) tetkikleri ile sağlanır [23].

OCT: OCT 840 nm dalga boyunda kızıl ötesi bir ışık kullanarak retinanın katmanlarının detayları ve optik sinir başı ve sinir lifleri tabakası hakkında çok yararlı görüntüler elde etmeye yarayan bir cihazdır [28]. Hastanın gözüne dokunmadan, iğne ile ilaç verilmeden uygulanan bir yöntemdir.

Fundus Kamera: Retinal bulguların görüntülenmesinde kullanılan bir cihazdır. Sadece büyük detaylar inceleyebiliyor olsa da bu cihaz retinal bulguların görüntülenmesinde büyük önem taşımaktadır [29].

2.4.2. Diyabetik Retinopati Tedavisi

Diyabetik retinopati tedavisi için kullanılan yöntemlerden ilki ilaçla tedavi yöntemidir. Göz içine verilen bazı ilaçlarla makula ödemi ve yeni damar oluşumları ortadan kaldırılmaya çalışılır [24].

Tedavi için uygulanan bir diğer yöntem ise lazer fotokagulasyon yöntemidir. Fotokaugulasyon, ışık enerjisinin hedef hücrelerce absorbe edilip ısıya çevrildiği ve irreversible termal denatürasyonun oluştuğu harap edici bir tedavi çeşididir. Hasarlı alanlar yok edilerek sağlam alanlar korunmuş olur. Seanslar halinde yapılır ve retina tabakası ne kadar fazla hasar görmüş ise seans sayısı da o kadar fazla olmaktadır [30].

Bu tedaviler ancak hastanın kan şekerinin düzenlenmesiyle faydalı olur. Sabır gerektiren bir tedavi süreci vardır. Bu nedenle hızlı bir düzelme beklememek gerekir. Hastanın diyetisyen, endokrinolog ve göz doktoru ile birlikte bu tedavi sürecine girmesi sağlanır [25].

Eğer yukarıdaki tedavi süreçlerinden hiçbiri sonuç vermez ve göz içinde yoğun kanama olursa bu kanamaları yok etmek için vitro – retinal adı verilen bir cerrahi operasyon yapılmaktadır [31].

3. GÖRÜNTÜ VE GÖRÜNTÜYÜ OLUŞTURAN BİLEŞENLER

Işığın bir cisme çarparak yansıması sonucu görme olayı meydana gelir. Görüntü ise bir cismin optik bir aygıt tarafından elde edilen resmidir. Görüntünün işlenmesi çeşitli algoritmalarla beraber bazı programlama dilleri ve kütüphanelerini kullanarak gerçekleştirilebilir.

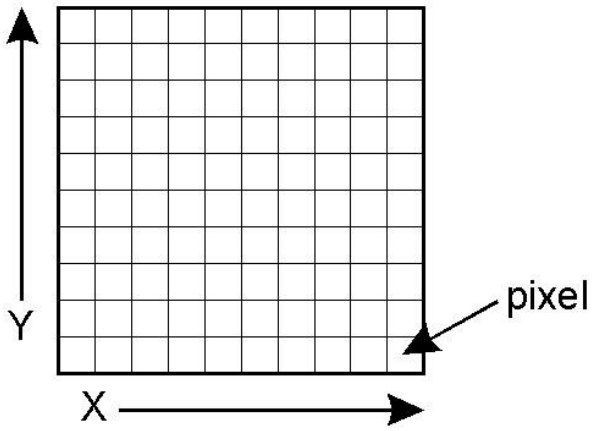
3.1. Görüntünün Temel Bileşenleri

Görüntünün temel bileşenleri piksel, dot (nokta) ve dot pitch (nokta aralığı), çözünürlük, rezolasyon, LPI (Liner per Inch), DPI (Dots per Inch), çerçeve, renk kanalları ve renk uzayları olarak gösterilebilir.

3.1.1. Piksel

Görüntüler kare şeklindeki noktalardan oluşmaktadır. Görüntü yakınlaştırıldığında bu noktalar fark edilebilmektedir. Görüntüyü oluşturan bu kare şeklindeki noktalara piksel denilmektedir. Piksel İngilizcedeki **Pic**ture **El**ement yani resim parçası anlamına gelen kelimelerden çıkarılmıştır [32].

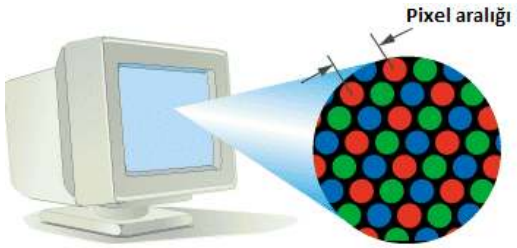
Renkli görüntülerde rengi oluşturmak için bir piksel üç ya da dört renk kullanmaktadır. Üç renk kullanan pikseller kırmızı, yeşil ve mavi renklerini kullanırken dört renk kullanan pikseller ise ciyan (camgöbeği), eflatun, sarı ve siyah renklerini kullanmaktadır.



Şekil 3.1. Görüntünün temel taşı pikseller

3.1.2. Dot ve Dot Pitch

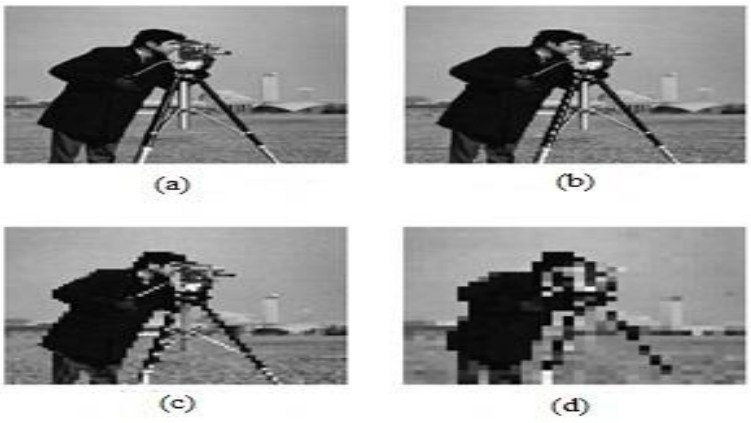
Pikseller üç veya dört renk kullanan kare şeklindeki noktalardır. Pikselleri meydana getiren bu renklerin her birine dot (nokta) denilmektedir. Dot Pitch (nokta aralığı) ise renklerin yani noktaların (dot) birbirine olan uzaklığı anlamına gelmektedir. Nokta aralığı ne kadar az olursa görüntü kaliteside bir o kadar artmaktadır.



Şekil 3.2. Dot (nokta) ve dot pitch (nokta aralığı)

3.1.3. Çözünürlük

Çözünürlük, ekranda görüntülenebilen piksel sayısını ifade etmektedir. Ekrandaki yatayda ve dikeyde bulunan piksel sayılarının çarpılmasıyla görüntünün çözünürlük değeri bulunmaktadır. *64 x 64, 256 x 256, 1024 x 768* gibi bazı çözünürlük değerleri örnek gösterilebilir. Aynı fiziksel büyüklüğe sahip görüntüler aynı çözünürlükte olmayabilirler [33]. Görüntüdeki piksel sayısı arttıkça yani başka bir deyişle çözünürlük değeri arttıkça görüntü kaliteside artmaktadır.



Şekil 3.3. Çözünürlük piksel sayısı ilişkisi (a) *256x256* piksel, (b) *128x128* piksel, (c) *64x64* piksel, (d) *32x32* piksel görüntü

3.1.4. Rezolasyon

Görüntüde bir inç karede bulunan piksel sayısını ifade etmektedir. Yani piksel yoğunluğu anlamına gelmektedir.

49 piksel (1 inç karede)

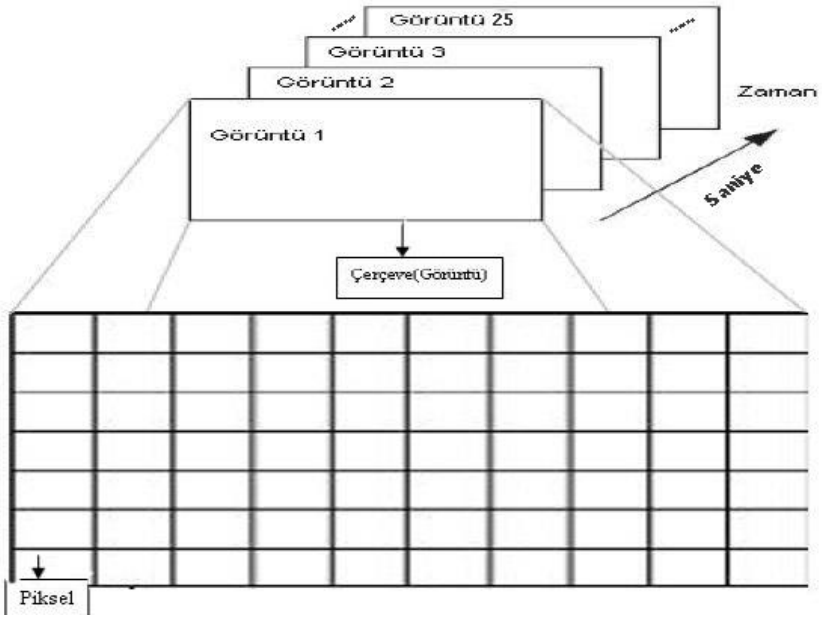
34 piksel (1 inç karede)

24 piksel (1 inç karede)

Şekil 3.4. Rezolasyon değerleri farklı görüntüler

3.1.5. Çerçeve (Frame)

Saniyede arka arkaya gösterilen 25 adet sayısal olarak derlenmiş resim dizisine gerçek zamanlı görüntü veya video denilir. Resim dizisini oluşturan her bir resme ise çerçeve adı verilir [3].



Şekil 3.5. Çerçeve

3.1.6. LPI (Line Per Inch)

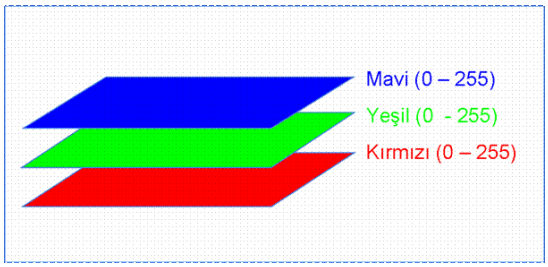
Görüntüde bir inç (*2,54 cm*) yüksekliğindeki alana düşen satır sayısını ifade etmektedir.

3.1.7. DPI (Dot Per Inch)

Çıktı cihazlarında bir inç karede basılan piksel sayısını ifade etmektedir [34]. Bu değer arttıkça basılan görüntünün kaliteside artmaktadır.

3.1.8. Renk Kanalları

Farklı tipte bilgi saklayan gri tonlamalı görüntülerdir [35]. Bir başka deyişle renk kanalları sayısal bir görüntüdeki pikselin ana renk seviyelerinin ayrı ayrı tutulduğu kanallardır [33]. Bu renk kanalları bir araya gelerek renkli görüntüler elde edilir [36]. Örneğin bir RGB (Red – Green – Blue yani Kırmızı – Yeşil – Mavi) bir görüntüde her renk bir kanalı oluşturmaktadır.



Şekil 3.6. Renk kanalları

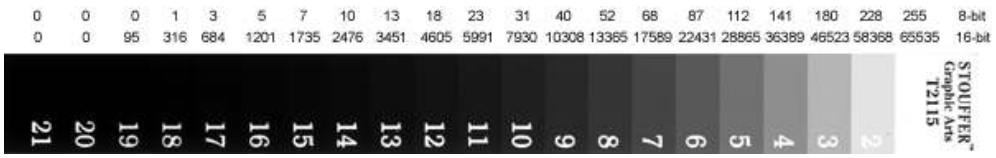
3.1.9. Renk Modelleri

Renk modeli dijital görüntülerde görünen renkleri tanımlamaktadır. RGB (Red – Green – Blue yani Kırmızı – Yeşil – Mavi), CMYK (Cyan – Magenta – Yellow – Key yani Mavi – Kırmızı, Mor – Sarı – Siyah), HSB (Hue – Saturation – Brightness yani renk tonu – doygunluk – parlaklık) gibi renk modelleri bulunmakta ve bu renk modelleri rengin tanımlanmasında her biri farklı olmak üzere bir yöntemi temsil etmektedirler[37].

Renk uzayı, renk modelinin bir varyantıdır ve kendine özgü bir renk aralığına sahiptir. Örneğin RGB renk modeli için birçok renk alanı vardır. Monitör, yazıcı gibi aygıtların kendine özgü renk uzayı vardır ve yalnızca o aralıktaki renkleri üretebilirler. Her aygıtın kendine göre bir renk uzayı olduğundan görüntü de aygıttan aygıta göre değişiklik gösterebilmektedir.

3.1.9.1 Gri Tonlamalı Renk Uzayı

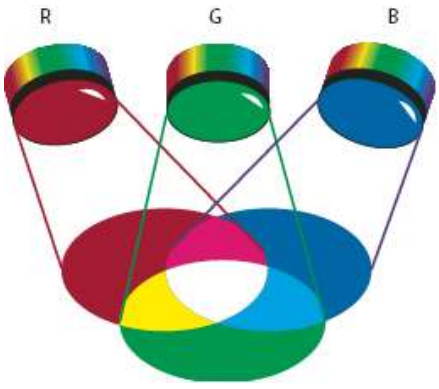
Bir görüntüde *0 – 255* arasında toplamda grinin *256* tonunun kullanıldığı renk uzayıdır. Gri ölçekli görüntüler 8 bitlik derinliğe sahiptir. Gri tonlamalı görüntünün her bir pikselinin parlaklık değeri *0 -255* arasında değişir. *0* siyah *255* ise beyaz renk anlamına gelmektedir. Aradaki değerler ise grinin değişik tonlarını ifade etmektedir. Tek bir kanalı ifade eden bu renk uzayı siyahtan beyaza tüm gri tonlarını ışık yoğunluğuna göre ayarlamaktadır.



Şekil 3.7. Gri tonları (0 – 255)

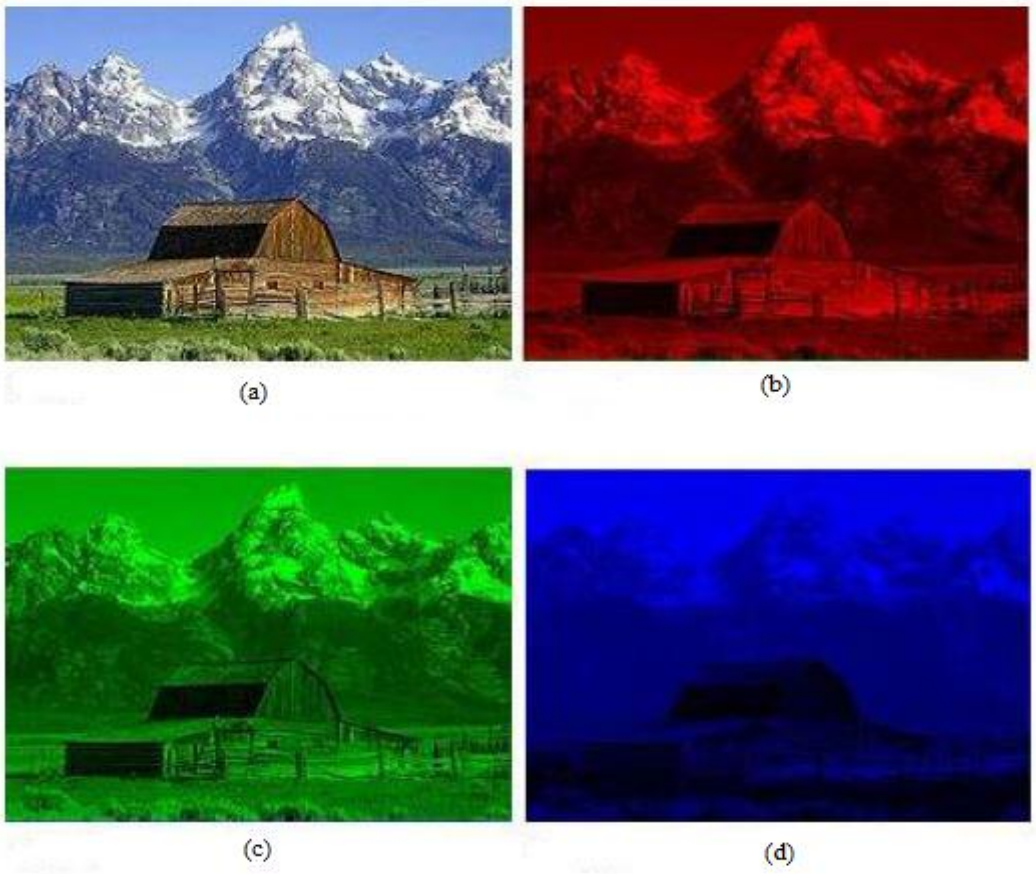
3.1.9.2 RGB (Red – Green – Blue yani Kırmızı – Yeşil – Mavi) Renk Uzayı

RGB: Kırmız, Yeşil ve Mavi için kullanılan bir kısaltmadır. Kırmızı (Red) “R”, Yeşil (Green) “G” ve Mavi (Blue) “B” ile gösterilir [38]. Bu renk uzayında tüm renkler kırmızı, yeşil ve mavinin farklı oranlarda karışımından elde edilir.



Şekil 3.8. RGB renk düzeni

RGB renk uzayında kırmızı, yeşil ve mavi renklerin *%100* oranında karışımından beyaz renk, *%0* oranında karışımından ise siyah renk elde edilmektedir. Bu üç ana rengin diğer oranlarda karışımından ise farklı renkler elde edilmektedir. Fakat doğada bulunan bazı renklerin karşılığı bulunmamaktadır.



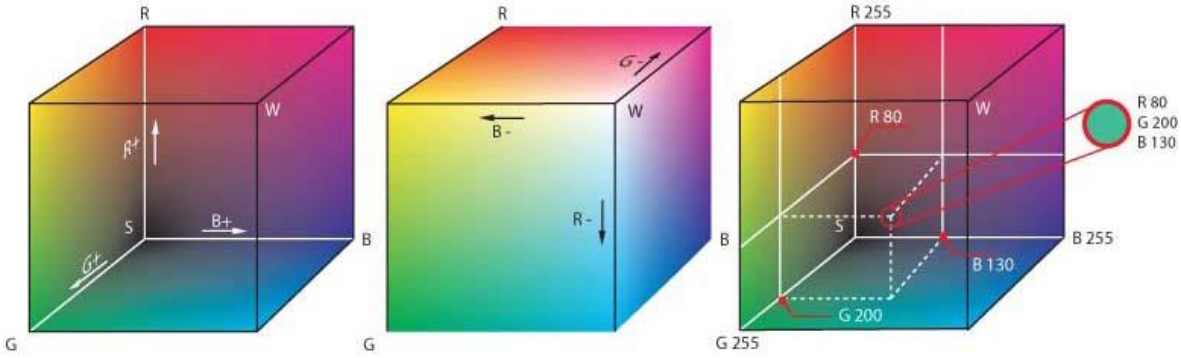
Şekil 3.9. RGB renk uzayı bileşenleri. (a) kırmızı, yeşil, mavi tonların bileşimi ile oluşmuş orijinal görüntü. (b) kırmızı rengin tonları ile oluşmuş görüntü. (c) yeşil rengin tonları ile oluşmuş görüntü. (d) mavi rengin tonları ile oluşmuş görüntü

Tablo 3.1'de çizelgede karışım oranlarına göre ortaya çıkan renkler gösterilmiştir.

Tablo 3.1. RGB renk uzayı karışım oranlarına göre elde edilen renk tonları

	Aralık	Beyaz	Sarı	Camgöbeği	Yeşil	Eflatun	Kırmızı	Mavi	Siyah
R	0- 255	255	255	0	0	255	255	0	0
G	0- 255	255	255	255	255	0	0	0	0
B	0- 255	255	0	255	0	255	0	255	0

RGB renk uzayı koordinat eksenleri kırmızı, yeşil ve mavi olan 3D bir uzay olarak düşünülebilir. Oluşturulmak istenilen renkler bu üç ana rengin koordinatları cinsinden ifade edilebilir [36].

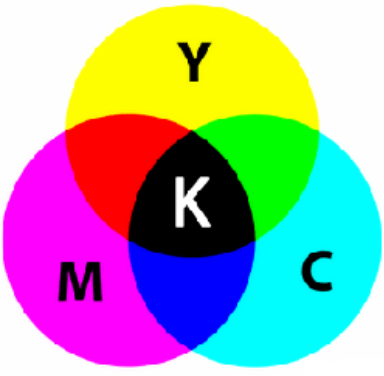


Şekil 3.10. RGB renk uzayı koordinat ekseni

RGB renk uzayındaki bir görüntüyü işlemek kolay ve hızlı bir işlem değildir. Örneğin görüntünün renk yoğunluğu değiştirilmek istendiğinde görüntüden kırmızı (red), yeşil (green) ve mavi (blue) renk yoğunluklarının üçüde okunmalı ve değişiklik yapıldıktan sonra tekrar görüntüye uygulanmalıdır. Farklı bir renk uzayı kullanarak görüntünün yoğunluk ve renk biçimlerine daha kolay ve hızlı erişilebilinir böylelikle bazı işlem basamakları daha hızlı yapılabilir. Bu sorunun üstesinden gelmek için parlaklık ve iki farklı renk sinyali kullanan renk uzayları geliştirilmiştir. Bu renk standartları YUV, YIQ, YCbCr ve HSV renk uzaylarıdır.

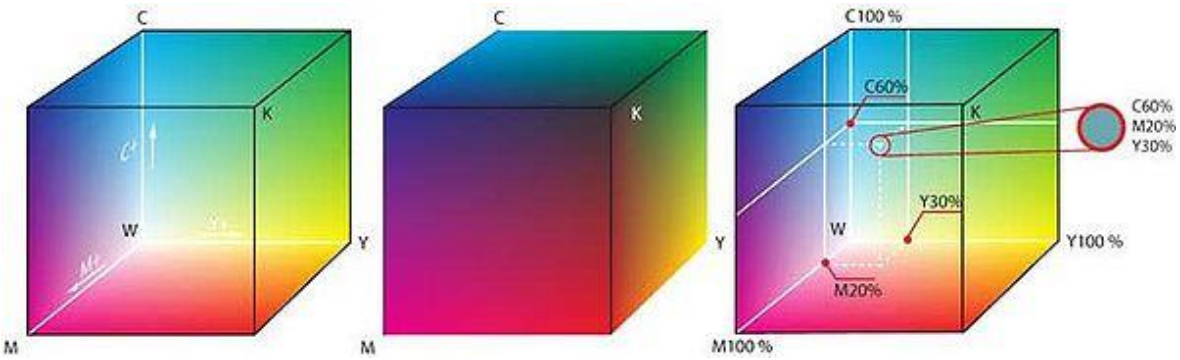
3.1.9.3 CMYK (Cyan – Magenta – Yellow – Key yani Mavi – Kırmızı, Mor – Sarı – Siyah) Renk Uzayı

Cyan, Magenta, Yellow, Black renklerinin oluşturmuş olduğu renk uzayıdır. Tüm renkler Cyan (Mavi), Magenta (Kırmızı – Mor), Yellow (Sarı), Key (Siyah) renklerin karışımından elde edilir. CMYK renk modeli RGB modelinin bir alt kümesi gibidir ve renkli baskı üretiminde kullanılır[38]. RGB ışığın, CMYK ise yansımanın rengidir. RGB' de boş zemin olarak siyah kullanılırken CMYK' da beyaz kullanılmaktadır.



Şekil 3.11. CMYK renk düzeni

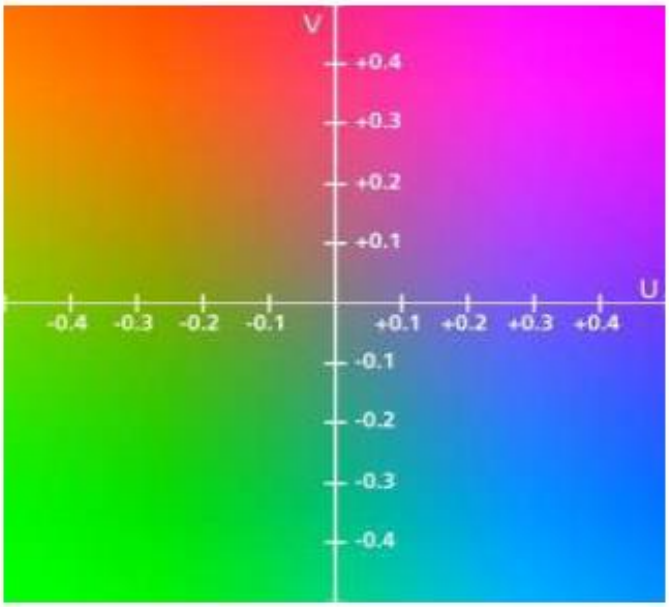
Temelde renk sayısı *C,M,Y* olmak üzere 3 tür. Siyah bunlara sonradan katılmıştır. Bu üç renk teorik olarak *%100* oranlarında karışarak siyahı verirken, pratikte tam rengi vermeyişinden ve bu üç rengin karışımından meydana gelecek (duygusal) maliyetlerden dolayı sonradan eklenmiştir [39].



Şekil 3.12. CMYK renk uzayı koordinat ekseni

3.1.9.4 YUV (Luminance, Chrominancel1, Chrominancel1) Renk Uzayı

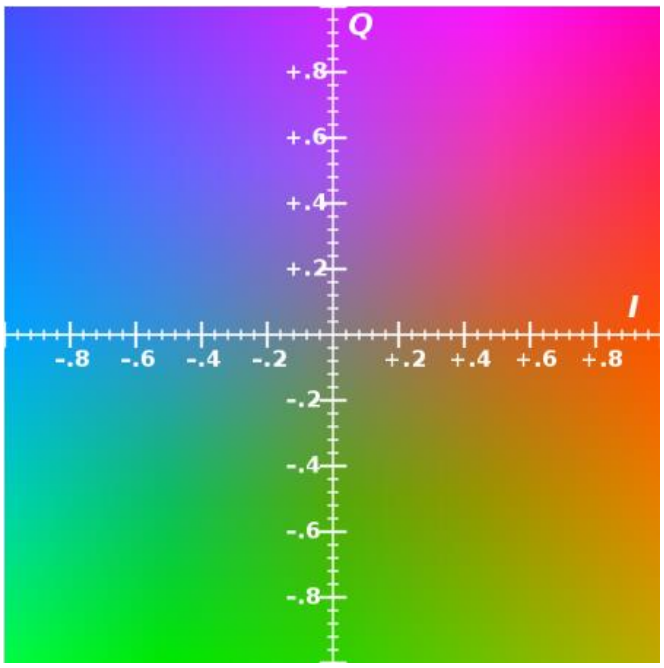
Siyah beyaz resimdeki parlaklık bilgisini *Y* bileşeninde, renk bilgisini *U* ve *V* bileşenlerinde tutan bir renk uzayıdır. YUV renk uzayı PAL (Phase Alternate Lines), NTSC (National Television Stanndarts Committee) ve SECAM (Sequential Couleur A'Memorie) bileşik ve renkli video standartlarında kullanılmaktadır. Bu renk uzayının geliştirilmesindeki temel amaç siyah – beyaz alıcıların, renkli video sinyalini siyah – beyaz şeklinde gösterebilmeleridir [40].



Şekil 3.13. YUV renk uzayı bileşenleri. U ve V koordinat düzleminin bileşenleri renk bilgisini oluşturur.

3.1.9.5 YIQ (Luminance – Orange, Blue – Purple, Green) Renk Uzayı

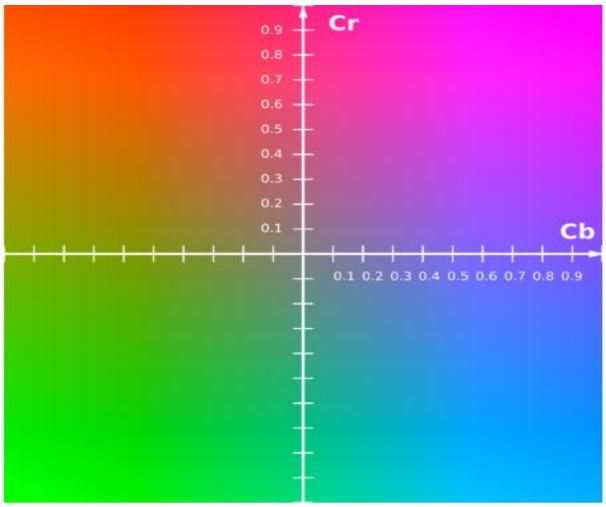
YIQ renk uzayı, YUV renk uzayından türetilmiş ve NTSC bileşik video standardında seçimlilik olarak kullanılan bir renk uzayıdır. Burada *Y* parlaklık değerini göstermekte *I* ve *Q* değerleri ise renk bilgisini transfer etmek için kullanılmaktadır [33].



Şekil 3.14. YIQ renk uzayı bileşenleri. I ve Q koordinat düzleminin bileşenleri renk bilgisini oluşturur.

3.1.9.6 YCbCrUV (Luminance – Chrominance blue – Chrominance red) Renk Uzayı

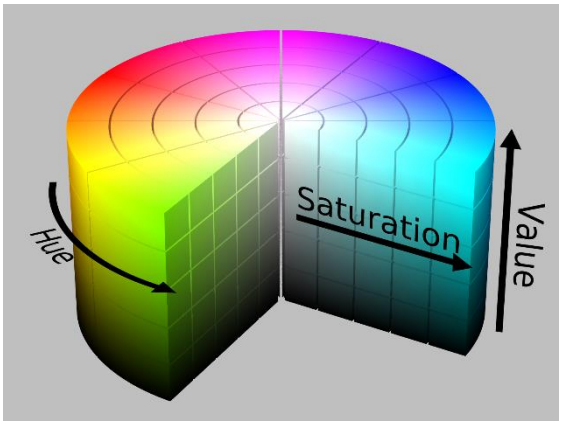
YCbCr, *Y* ile parlaklık sinyalini, *Cb* ve *Cr* ile renk bilgilerini saklayan bir renk uzayıdır. YCbCr renk uzayı, dünya çapında sayısal video standardı oluşturma çabaları sırasında ortaya çıkmıştır. *Y*, *8* bitlik *16 – 235* arasında tanımlanmaktadır. *Cb* ve *Cr* bileşenleri ise *16 – 240* arasında tanımlanmaktadır [41].



Şekil 3.15. YCbCr renk uzayı, Cr ve Cb koordinat düzleminin bileşenleri renk bilgisini oluşturur.

3.1.9.7 HSV (Hue – Saturation – Value) Renk Uzayı

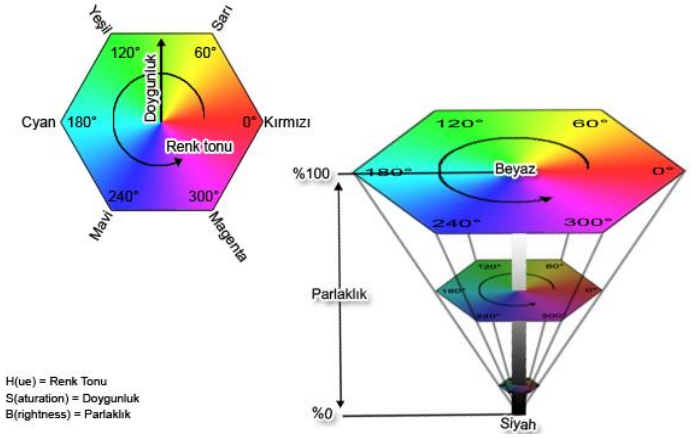
HSV renk uzayı renkleri içerdiği RGB değerlerine göre değil de Hue (renk özü), Saturation (doygunluk) ve Value (parlaklık) değerlerine göre belirten bir renk uzayı türüdür [42].



Şekil 3.16. HSV renk uzayının gösterimi

3.1.9.8 HSB (Hue – Saturation – Brightness) Renk Uzayı

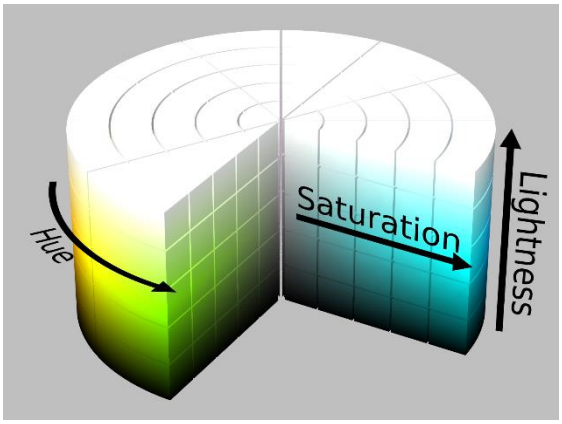
Bu model renkleri üç kanalda ifade eder. *H*: renk (hue), *S*: doygunluk (saturation), *B*: aydınlık (brightness) anlamlarına gelmektedir. Bu modelle birlikte renkler artık rakamlarla ifade edilebilecek hale getirilmiştir. Her kanal *0* ile *255* arasında değerler alır, tüm değerlerin *255* olması durumunda beyaz, tamamının *0* olması durumunda ise siyah renk oluşur [43].



Şekil 3.17. HSB renk uzayının gösterimi

3.1.9.9 HLS (Hue – Luminance – Saturation) Renk Uzayı

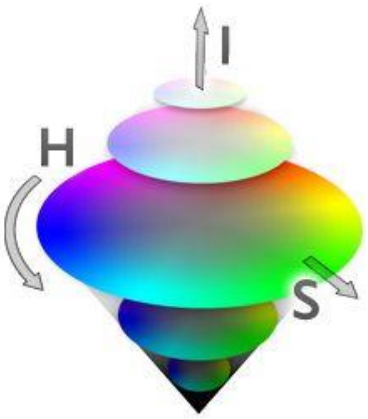
HSL (Hue Saturation Lightness) renk modeli HSV ile benzerlik gösterir; bu modelde tek fark parlaklık yerine aydınlık değeri kullanılmasıdır. *%50* aydınlık değeri nötr renkler verirken, aydınlık değerini düşürmek renklerin daha koyu olmasına neden olur. Benzer şekilde aydınlık değeri yükseltilirse, beyaz renge giderek yaklaşılmış olur.



Şekil 3.18. HSL renk uzayının gösterimi

3.1.9.10 HSI (Hue – Saturation – Intensity) Renk Uzayı

HSL ve HIS birbirine çok benzerdirler. Parlaklık bileşeni *L* yerine yoğunluk bileşeni *I* kullanılmıştır. HSI renk uzayı *R, G, B* değerlerine eşit oranda bağlı olan parlaklık değerleri ile doğrudan alakalı katsayı, eşitleme, histogram gibi geleneksel resim işleme metotları için en iyi renk uzayıdır.



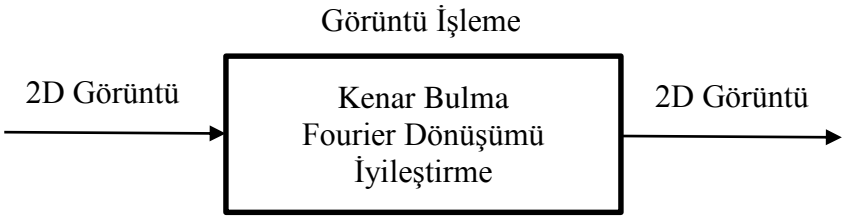
Şekil 3.19. HSI renk uzayının gösterimi

4. GÖRÜNTÜ İŞLEME

Görme duyusu, insanın sahip olduğu en önemli algılamalardan biridir [4]. Gözümüz gördüğümüz nesneleri ayırt etmemizi ve tanımamızı sağlamaktadır. Gelişen teknolojiyle birlikte bir görüntüden nesnelerin tanınması, ayırt edilmesi, şekillendirilmesi gibi birçok işlem gerçekleştirilebilmektedir. Bu işlemler tıp, askeri, güvenlik, robotik vb. birçok alanda insanlara fayda sağlamaktadır.

4.1. Görüntü İşleme

Dijital bir resim haline getirilmiş olan gerçek yaşamdaki görüntülerin, bir girdi resim olarak işlenerek, o resmin özelliklerinin ve görüntüsünün değiştirilmesi sonucunda yeni bir resmin oluşturulmasıdır [44].



Şekil 4.1. Görüntü işleme

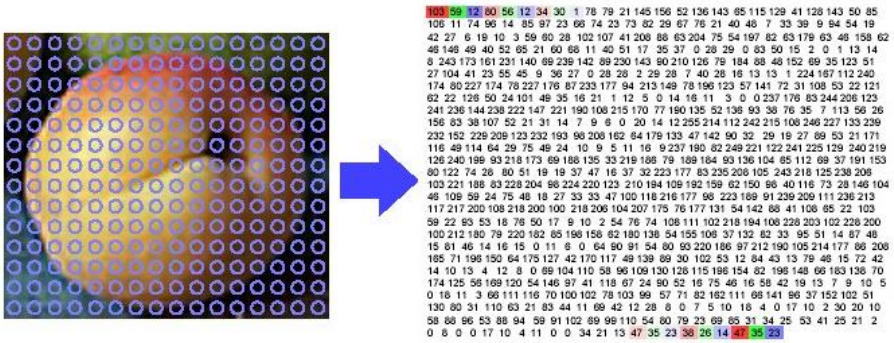
4.2. Görüntü

Görüntü çeşitli yollarla elde edilen bilgilerin görüntüsel olarak saklanmasına ve gösterimine olanak sağlayan resimlerdir. Her türlü iki boyutlu bilgi görüntü olarak ele alınabilir [45].

4.3. Sayısallaştırılmış Görüntü

Bir görüntünün işlenebilmesi için öncelikle bilgisayarın anlayacağı şekle dönüştürülmesi gerekmektedir. Bu dönüştürme işlemine sayısallaştırma da denilmektedir.

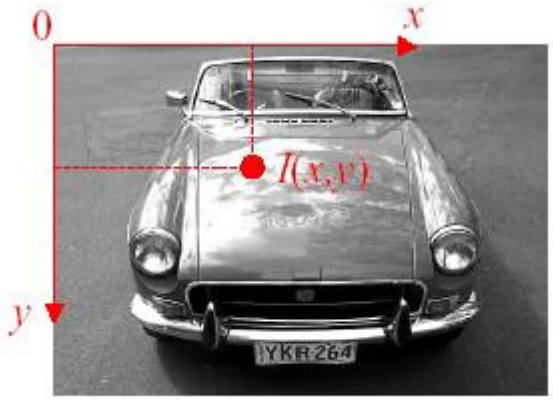
Sayısallaştırılmış bir görüntü yani bilgisayarın anlayabileceği dijital görüntü üzerinde görüntü işleme gerçekleştirilebilir. Sayısallaştırma işleminde görüntünün her bir pikseli bir sayı olarak hafızada depolanacak olan karelere bölünmektedir. Her piksel değerinde görüntünün parlaklığını ve koyuluğunu temsil eden bir tamsayı bulunmaktadır. Bütün pikseller böyle dönüştürüldüğünde görüntü tamsayılardan oluşan matris şekline dönüşmektedir ve bu şekildeki görüntü de yazılım tarafından işlenmeye hazırdır.



Gerçek Resim Dijital Resim

Şekil 4.2. Gerçek bir resmin dijital görüntüsü

Görüntü iki boyutlu (2D) bir ışık yoğunluk fonksiyonu *f(x,y)* olarak gösterilebilir. *"x"* ve *"y"* görüntünün koordinatlarını belirtirken *f(x,y)* fonksiyonunun değeri ise *(x,y)* noktasındaki pikselin sayısal olarak ışıklık değerini temsil etmektedir.

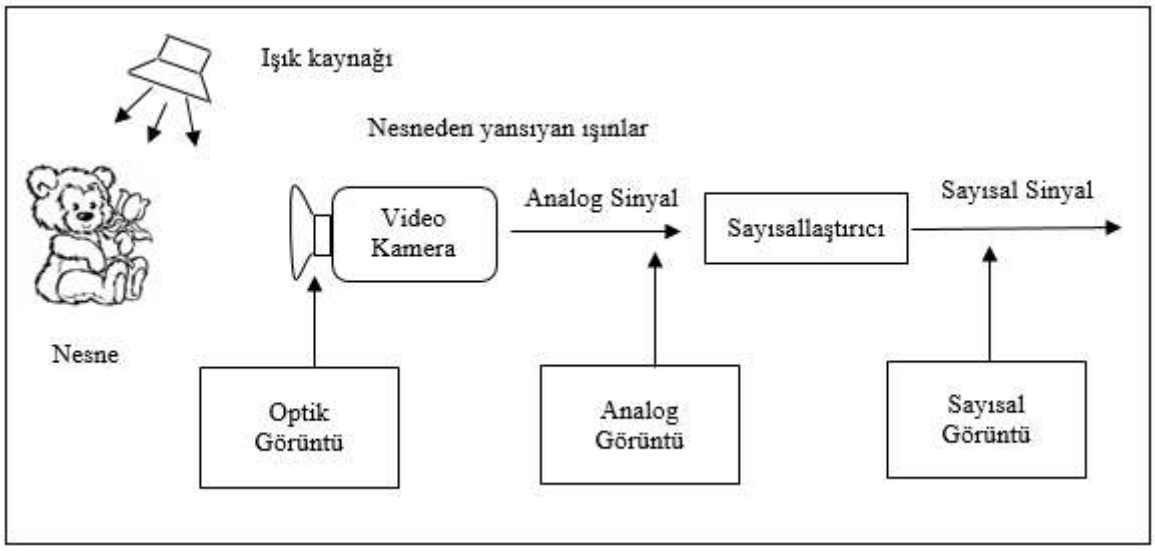


Şekil 4.3. Dijital resim fonksiyonu

Görüntünün, görüntü işleme teknikleri ile bilgisayar ortamında işlenebilmesi için analog görüntü türünden dijital görüntü türüne çevrilmesi gerekmektedir. Bu işlem için görüntünün örnekleme ve nicemleme aşamalarından geçirilmesi gerekir [45].

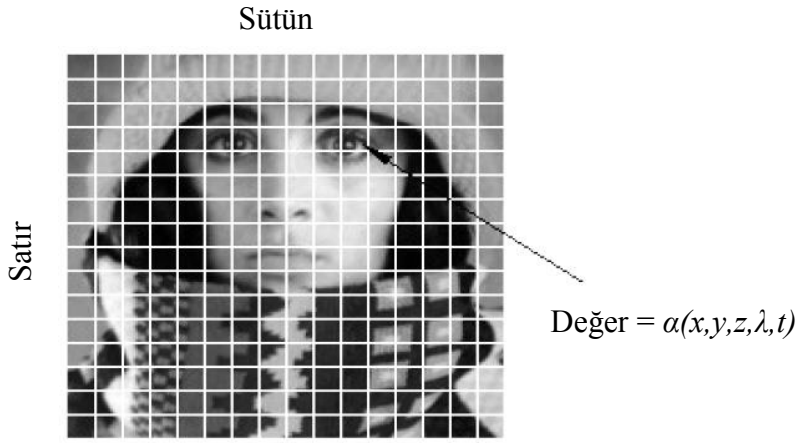
- **Örnekleme:** Görüntüden belirli zamanlarda örnekler alınmasıdır.
- **Nicemleme:** Genlik seviyelerinin sadece belirli değerleri alması işlemidir.

SAYISALLAŞTIRMA = ÖRNEKLEME + NİCEMLEME



Şekil 4.4. Analog görüntünün dijitale çevrilmesi

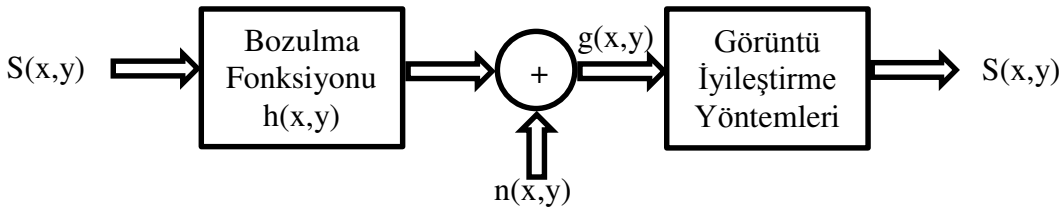
Tüm bu işlemelerden sonra dijital görüntümüz *M* sütun, *N* satır sayılarından oluşmaktadır ve bu satır ve sütunların kesiştiği yerde de piksel adı verilmektedir. Pikseldeki değer ise derinlik (z), renk (λ) ve zamanın bir fonksiyonudur [44].



Şekil 4.5. Dijital görüntüde satır ve sütunlar

Her pikselin değeri o pikselin parlaklık değerinin en yakın tam sayıya yuvarlatılmış halidir. Resimlerin analog ortamdan dijital ortamlara geçirilmesi sırasında yeni oluşan

görüntüde bir takım bozulmalar meydana gelebilir. Bu bozulmalar gürültü (noise) olarak adlandırılmaktadır. Görüntü üzerinde işlem yapmadan önce bu bozulmaların yani gürültülerin giderilmesi gerekir.



Şekil 4.6. Görüntüde bozulma ve iyileştirme işlemlerinin genel yapısı

4.4. Görüntü İşlemede Ön İşlem

Bilgisayara alınmış sayısal görüntü üzerinde işlem yapabilmek için sırasıyla eşikleme ve kenar çıkarma gibi aşamaların görüntü üzerine uygulanması gerekir [4].

4.4.1. Histogram Eşitleme

Histogram bir görüntüdeki renk değerlerinin sayılarını gösteren bir grafik olarak ifade edilebilir. Histogram eşitleme ise bir görüntüdeki renk değerlerinin belli bir yerde kümelenmiş olmasından kaynaklanan renk dağılımı bozukluğunu gidermek için kullanılan bir yöntemdir. Görüntü işlemede histogramlar kullanılarak görüntü daha belirgin bir hale getirilebilmektedir [46]. Bir görüntünün parlaklığını ve kontrastını arttırmak için histogram eşitleme yapılır [47]. Histogram eşitleme;

$$Sk = T(rk) = \sum_{j=0}^{k} \frac{n_j}{n} (L-1) \qquad (1)$$

Formülü ile ifade edilir. Bu formülde:

k = 0, 1, 2, ……. L-1

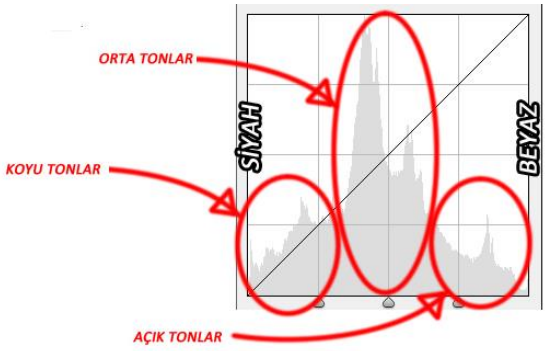
L = Görüntünün gri seviyelerinin sayısı

n_j *= Görüntüdeki j gri pikselin değeri*

n = Görüntüdeki toplam piksel sayısı

olarak kullanılmaktadır [48].

Histogram okuma işleminde ise histogram grafiğinin en solu koyu rengi yani siyah rengi temsil ederken sağ tarafı ise aydınlığı yani beyaz rengi temsil eder.



Şekil 4.7. Histogram grafiği ve histogram okuma

(a) (b)

Şekil 4.8. Histogram eşitleme uygulaması (a) orijinal resim, (b) histogram eşitlemeden sonraki resim

4.4.2. Eşikleme

Eşikleme işleminden amaç, görüntü içerisindeki nesneleri görüntü arka planından ayırmaktır [47]. Eşikle işlemiyle görüntü iki renkle ifade edilebilecek şekle dönüştürülür. Belirli bir eşik değeri belirlenir ve bu eşik değerinden daha yüksek gri seviye değerine sahip piksellere "1" değeri, daha düşük gri seviye değerine sahip olan piksellere ise "0" değeri verilmektedir.

Eşiklemenin temel ifadesi;

$$G(i,j) = 1 \qquad f(i,j) \geq T \; için$$
$$G(i,j) = 0 \qquad f(i,j) < T \; için \tag{2}$$

olarak verilmektedir.

Burada:

T → *Eşik değeri*

G(i,j) = 1 → *Objenin görüntü elemanları*

G(i,j) = 0 → *Objenin görüntü elemanlarıdır.*

Doğru bir eşik değeri seçimi başarılı bir sonuç için çok önemlidir. Bu seçim çeşitli eşik değeri belirleme algoritmalarıyla yapılmaktadır. Görüntünün tamamı için tek bir eşik değeri kullanımı fazla tercih edilen bir durum değildir. Tek bir eşik değeri kullanımının global eşikleme denilmektedir [50]. Bölütleme eşik değeri bölgesel görüntü karakteristiklerinin bir fonksiyonu olarak görüntü üzerinde değiştiğinde, değişken eşiker kullanılarak bu durumlara çözüm üretilir. Bu bölütleme de adaptif bölütleme olarak adlandırılır [51].

(a) (b)

Şekil 4.9. Eşikleme uygulaması (a) orijinal resim, (b) global eşikleme uygulanmış resim

4.4.3. Kenar Çıkarma

Kenar çıkarmadaki amaç uygulanan görüntüdeki renk geçişlerini keskinleştirerek görüntüde bulunan objelerin elde edilmesini kolaylaştırmaktadır. Kenar belirlenmesi resimleri anlamada çok önemlidir çünkü resimlerin anlamlı özelliklerini barındırır ve bu özelliklerin, bilgilerin ortaya çıkmasını sağlarlar [52]. Kenarlar genellikle nesnelerin sınırlarına ve gölge geçişlerine uyarlanır [53].

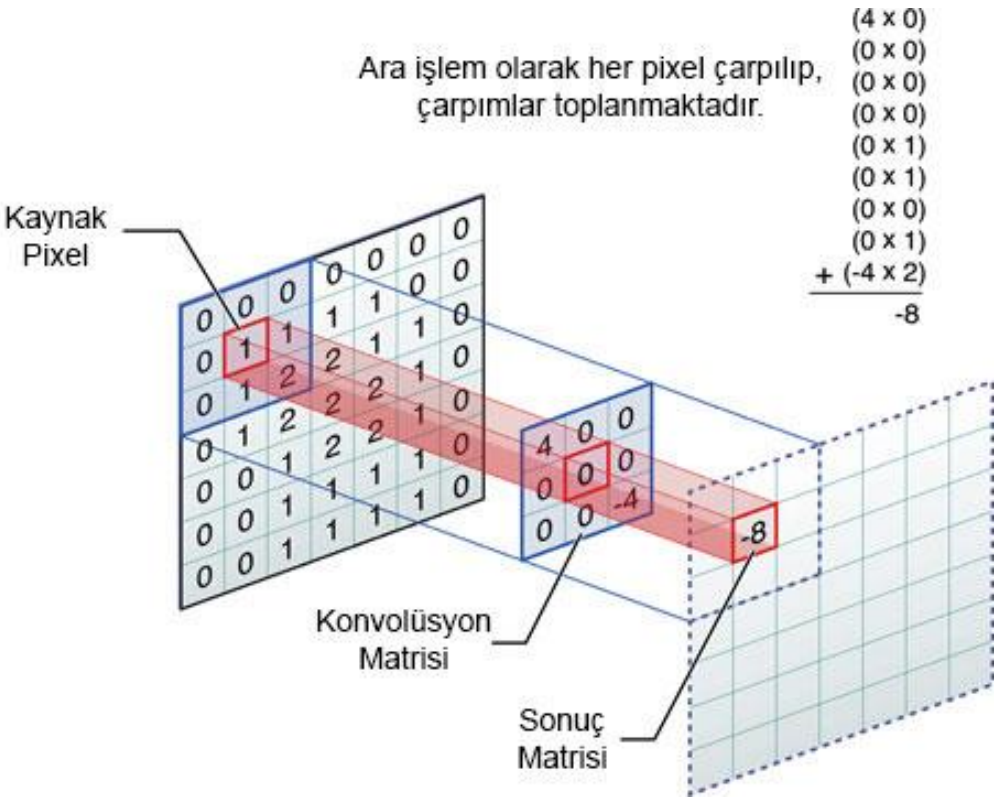
Kenar, bir pikselin bir bölgesinin ve yakın komşularının özelliğidir. Ayrıca büyüklüğü ve yönü veren vektördür. Çok açık renkli resimler, kenar hesaplamalarında kullanılır ve resmin Gradient fonksiyonu kenarları hesaplar. Kenar belirleme gri rengin geçişlerini anlamlı bir şekilde ölçmemize yarar [54].

Görüntüdeki kenarlar farklı algoritmalar yani filtreler kullanılarak bulunabilir. Bu filtrelerle veya algoritmalarla resmin üzerinde bir filtre varmış gibi her piksel değeri yeniden hesaplanır. Filtreleme işlemi genel olarak Denklem 3' deki gibi ifade edilir:

$$\sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} h(i,j) x\, f(x-i, y-j) \tag{3}$$

Denklem 3' deki *h* fonksiyonu görüntüye uygulanan filtredir [44]. Filtre resim değerleri ile çarpılıp toplanarak tek bir değer elde edilmekte ve bu değer filtrenin resim üzerinde denk geldiği merkez piksele atanmaktadır. Filtre, resmin ilk satır ve sütunuyla son satır ve sütununa uygulamayacak şekilde hareket edilmelidir [55].

Filtredeki katsayıların ve filtrenin boyutu yapılan işlemde önemlidir. Filtrenin boyutu genelde *3 x 3* matris şeklindedir. Filtrenin boyutu arttıkça daha fazla komşu değerin hesaplanması gerektiği için işlem yükü artar ve ince detaylar kaybolur. Dolayısıyla büyük boyutlu filtreler pek tercih edilmez. Filtreler genel olarak Sobel, Prewit, Roberts, Log, Canny ve Laplace kenar çıkarma algoritma ya da filtreleri olarak bilinmektedir.



Şekil 4.10. Filtreleme uygulaması

4.4.3.1 Sobel Filtresi

Sobel filtresi, verilen eşik düzeyinden kuvvetli olan kenarları bulmak gereksiz piksel bilgilerinin elenmesini sağlar [56].

$$\frac{\partial f(x,y)}{\partial x} = f_x(x,y) = f(x+1,y) - f(x,y) \tag{4}$$

$$\frac{\partial f(x,y)}{\partial y} = f_y(x,y) = f(x,y+1) - f(x,y) \tag{5}$$

$$\frac{\partial f(x,y)}{\partial x} \cong [f(x+1,y+1) - f(x-1,y+1)] + 2[f(x+1,y) - f(x-1,y)] + [f(x+1,y-1) - f(x-1,y-1)] \cong g_x \tag{6}$$

$$\frac{\partial f(x,y)}{\partial y} \cong [f(x-1,y+1) - f(x-1,y-1)] + 2[f(x,y+1) - f(x,y-1)] + [f(x+1,y+1) - f(x+1,y-1)] \cong g_y \tag{7}$$

Sobel filtresinde, Denklem 4'de verilen *x* yönündeki kısmi türeve Denklem 6, Denklem 5'de verilen *y* yönündeki kısmi türeve ise Denklem 7 uygulanmaktadır.

$$z(x,y) = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{8}$$

Denklem 8'de orijinal bir görüntünün *3 x 3* piksellik boyutlarında bir pencere örneği verilmektedir. Denklem 8'deki pencere örneğine Denklem 6 ve Denklem 7 uygulanarak Denklem 9'daki gradyant elde edilmektedir.

$$\nabla f = grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{9}$$

Denklem 9 elde edildikten sonra Denklem 10'daki yaklaşık genlik hesabı Denklem 11'deki gibi yapılmaktadır.

$$M(x,y) \approx |g_x| + |g_y| \tag{10}$$

$$\begin{aligned} M(x,y) &\cong |g_x| + |g_y| \\ &= |(g + 2h + i) - (a + 2b + c)| + |(c + 2f + i) - (a + 2d + g)| \end{aligned} \tag{11}$$

Sobel filtresinde yukarıda verilen denklemlerden yola çıkarak g_x ve g_y değerlerini hesaplamak için Denklem 12' de verilen matrisler kaydırma tekniğiyle uygulanıp toplanarak filtrelenmiş görüntü elde edilmektedir.

$$W_{sgx} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad W_{sgy} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{12}$$

4.4.3.2 Prewitt Filtresi

Prewitt filtresi, görüntüler üzerinde yatay ve dikey yönlere ait olan kernel matrislerini kullanarak eğimlere odaklanıp sonuç veren bir algoritmadır [57].

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &\cong [f(x+1,y+1) - f(x-1,y+1)] + [f(x+1,y) - f(x-1,y)] \\ &\quad + [f(x+1,y-1) - f(x-1,y-1)] \cong g_x \end{aligned} \tag{13}$$

$$\begin{aligned} \frac{\partial f(x,y)}{\partial y} &\cong [f(x-1,y+1) - f(x-1,y-1)] + [f(x,y+1) - f(x,y-1)] \\ &\quad + [f(x+1,y+1) - f(x+1,y-1)] \cong g_y \end{aligned} \tag{14}$$

Prewit filtresinde, Denklem 4'de verilen *x* yönündeki kısmi türeve Denklem 13, Denklem 5'de verilen *y* yönündeki kısmi türeve ise Denklem 14 uygulanmaktadır.

Denklem 8'deki pencere örneğine Denklem 13 ve Denklem 14 uygulanarak Denklem 9'daki gradyant elde edilmektedir.

Denklem 9 elde edildikten sonra Denklem 10'daki yaklaşık genlik hesabı Denklem 15'deki gibi yapılmaktadır.

$$M(x,y) \cong |g_x| + |g_y| \\ = |(g+h+i)-(a+b+c)| + |(c+f+i)-(a+d+g)| \quad (15)$$

Prewit filtresinde yukarıda verilen denklemlerden yola çıkarak g_x ve g_y değerlerini hesaplamak için Denklem 16'da verilen matrisler kaydırma tekniğiyle uygulanıp toplanarak filtrelenmiş görüntü elde edilmektedir.

$$W_{sgx} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \qquad W_{sgy} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad (16)$$

4.4.3.3 Roberts Filtresi

Roberts filtresi diyagonal olarak kenar taramaktadır. Roberts filtresinde, Denklem 10'daki yaklaşık genlik hesabı Denklem 17'deki gibi yapılmaktadır.

$$M(x,y) \cong |g_x| + |g_y| = |f(i,j) - f(i+1,j+1)| + |f(i+1,j) - f(i,j+1)| \quad (17)$$

Roberts filtresinde yukarıda verilen denklemden yola çıkarak g_x ve g_y değerlerini hesaplamak için Denklem 18'de verilen matrisler kaydırma tekniğiyle uygulanıp toplanarak filtrelenmiş görüntü elde edilmektedir.

$$W_{sgx} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad W_{sgy} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad (18)$$

4.4.3.4 Log Filtresi

Laplacion of Gaussian – LoG algoritması olarak bilinir. Gaussian filtresinin laplası alınarak işlem yapılır. Log filtresi görüntünün ikinci türevini almaktadır. Log filtreleme işlemi genel olarak Denklem 4' deki gibi ifade edilir.

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}[1-\frac{x^2+y^2}{2\sigma^2}]e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4)$$

4.4.3.5 Laplace Filtresi

Laplace operatörü her yönde keskinleştirme yapmaya yarar. Laplas filtresi basitçe bir resimdeki kenar hatlarını belirlemek için kullanılır. Keskinleştirme Filtresi (Sharpening Filter) ismi ile de anılan laplas filtresi çalışırken bir pencere kullanır.

Denklem 19'da bu işlem sırasında kullanılabilecek *3×3* boyutlarında bir pencere örneği verilmektedir.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (19)$$

Yukarıdaki bu pencerede basitçe her hücreye gelen piksel değeri o hücredeki katsayı ile çarpılmakta ve sonuçlar toplanarak ortada bulunan pikselin yeni değeri bulunmaktadır.

Örnek olarak mevcut değerleri Denklem 20'deki gibi verilmiş olan bir matris ele alınmaktadır.

$$\begin{bmatrix} 4 & 5 & 6 \\ 3 & 3 & 8 \\ 2 & 1 & 7 \end{bmatrix} \quad (20)$$

Bu matrisin ortasındaki yeni değer şu formülle hesaplanmaktadır:

(1 x 4)+(1 x 5)+(1 x 6)+(1 x 3)+(-8 x 3)+(1 x 8)+(1 x 2)+(1 x 1)+(1 x 7) = 12 (21)

Bu hesaplama işleminden sonra matrisin filtre uygulanmış hali;

$$\begin{bmatrix} 4 & 5 & 6 \\ 3 & 12 & 8 \\ 2 & 1 & 7 \end{bmatrix} \quad (22)$$

olarak bulunur.

4.4.3.6 Canny Filtresi

Görüntünün türevi alınmadan önce yumuşatma filtresi uygulanır. Tek piksel kalınlığında kenarlar üretir ve kırık çizgileri birleştirir [58].

Amaçlar:

- Gürültüye karşı düşük duyarlılık
- İyi sınırlama
- Tek kenardaki birden çok karşılığı elemek

Metotlar:

- Gaussian yumuşatması
- İnceltme
- Histheresis eşik değerini belirleme

Canny kenar belirleme algoritması şu adımlardan oluşmaktadır:

1. Gauss filtresi ile görüntünün yumuşatılması.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{23}$$

2. Kısmi türevler için sonlu fark yaklaşımı kullanılarak gradyanın büyüklüğünün ve yönünün hesaplanması.

$$G(x,y) = \frac{\partial G}{\partial n} = n.\nabla G \qquad n = \frac{\nabla(G * g)}{|\nabla(G * g|} \tag{24}$$

G_n G'nin birinci türevidir n, g resminin yönüdür

3. Büyük gradyanların maksimum olmayan değerlerinin bastırılması.

$$\frac{\partial^2}{\partial_{n^2}} G * g = 0 \tag{25}$$

4. Kenarları algılamak için bir çift eşikleme algoritmasının kullanılması.

Canny algoritması diğer kenar belirleme algoritmalarına göre daha iyi sonuç veren bir algoritmadır. Bu nedenle en çok tercih edilen kenar çıkarma algoritmasıdır.

4.5. Görüntü İşlemede Özellik Çıkarma

Özellik çıkarımı, bir görüntüyü bütün bir bilgi olarak değilde bu görüntüyü oluşturan objelerin bazılarının çıkarılıp işleme hazır hale getirilmesidir. Özellik çıkarma işlemi bir boyut azaltma işlemidir. Buna göre karmaşık olan bir verinin boyutları azaltılarak daha basit bir problem haline indirgenir. Boyut azaltma işlemi için önerilmiş yöntemlerden en çok kullanılanlar PCA (Principle Components Analysis) ve LDA (Linear Discriminant Analysis) yöntemleridir.

PCA yönteminde verinin özellik vektörleri arasındaki ilişkiyi temsil eden kovaryans matrisinin eigen-vektör ve eigen-değer çarpımına eşit olduğu varsayılmaktadır. Eigen-vektörleri yeni temel bileşenler olarak kabul edilerek verinin yeni bileşenleri hesaplanmaktadır. PCA, boyut azaltma için kullanılıyorsa sapma değeri az olan boyut silinmekte ve gerekiyorsa veri kendi boyutuna geri dönüştürülmektedir.

LDA, veri içerisinde bulunan farklı sınıflara ait grupların doğrusal ayrılabilirliğini maksimize ederek boyut azaltması yapan bir yöntemdir. Her grup içerisindeki varyansı minimum ve grupların ortalamalarını birbirlerinden maksimum düzeyde uzak tutarak boyut küçültmesi işlemini gerçekleştirmektedir.

Doğru yapılmış bir özellik çıkarımı ve bu özelliklere uygun bir sistem tasarımı sonucun başarılı olması ve performansını etkileyen unsurlarıdır. Özellik çıkarma sonucunda elde edilen birden fazla özelliğin karşılığını tutan veri yapısına özellik vektörü (feature vector) adı da verilmektedir [59]. Özellik çıkarma işlemi için genellikle istatistiksel özellik çıkarma metodu, dalgacık dönüşümü tabanlı özellik çıkarma metodu ve moment tabanlı özellik çıkarma metotları kullanılmaktadır [60].

4.5.1. İstatistiksel Özellikler

Belirli bir bölgedeki nesnenin özelliklerinin tespit edilebilmesi için kullanılan en temel özellik çıkarma yöntemlerinden birisidir.

Görüntü histogramına dayanan istatistiksel özellik çıkarma işlemi için kullanılan özellikler aşağıdaki denklemlerde verilmektedir [61,62].

$$M = \tau = \sum_{i=1}^{n} ih(i) \quad (26)$$

$$STD = \sigma = \sum_{i=1}^{n} (i-\tau)^2 h(i) \quad (27)$$

$$TM = \sum_{i=1}^{n} (i-\tau)^3 h(i) \quad (28)$$

$$E_1 = -\sum_{i=1}^{n} h(i) \log\big(h(i)\big) \quad (29)$$

burada *h* iki boyutlu görüntü karesinin histogramını, *M* histogramın ortalamasını, *STD* histogramın standart sapmasını, *TM* üçüncü moment ve E_1 birinci dereceden entropi değerini belirtmektedir.

Gri seviye birlikte oluşma matrisi kullanılarak elde edilen istatistiksel özellik çıkarma işlemi için kullanılan özellikler ise aşağıdaki denklemlerde verilmektedir [63].

$$H = \frac{1}{N_c^2} \sum_i \sum_j \big(M_c(i,j)\big)^2 \quad (30)$$

$$M_c(i,j) = \sum_x \sum_y \{I(x,y) == i \& I(x+\Delta x, y+\Delta y) == j\} \quad (31)$$

$$Cont = \frac{1}{N_c(L-1)^2} \sum_{K=0}^{L-1} k^2 \sum_{|i-j|=k} M_c(i,j) \quad (32)$$

$$E_2 = 1 - \frac{1}{N_c(L-1)^2} \sum_i \sum_j M_c(i,j) log(M_c(i,j)) \quad (33)$$

$$Corr = \frac{1}{N_c \sigma_x \sigma_y} \left| \sum_i \sum_j (i-m_x)(j-m_y)(M_c(i,j)) \right| \quad (34)$$

$$m_x = \frac{1}{N_c} \sum_i \sum_j i M_c(i,j) \quad (35)$$

$$m_y = \frac{1}{N_c} \sum_i \sum_j j M_c(i,j) \quad (36)$$

$$\sigma_x^2 = \frac{1}{N_c} \sum_i \sum_j (i-m_x)^2 \, M_c(i,j) \quad (37)$$

$$\sigma_y^2 = \frac{1}{N_c} \sum_i \sum_j (i-m_y)^2 \, M_c(i,j) \quad (38)$$

$$LH = \frac{1}{N_c} \sum\nolimits_i \sum\nolimits_j \frac{1}{1+(i-j)^2} M_c(i,j) \quad (39)$$

$$Dg = \frac{1}{N_c}\sum_{i} M_c(i,i) \tag{40}$$

$$U = \frac{1}{N_c^2}\sum_{i} M_c(i,i)^2 \tag{41}$$

burada *i* ve *j [100]* ile belirtildiği gibi gri ton seviyeleri, Δx ve Δy *[-1 0 1]* değerlerini alan değişim miktarları *[100]* M_c birlikte oluşma matrisi *[100],* N_c ise M_c'nin elemanlarının toplamı, *H* türdeşlik, *Cont* kontrast, E_2 ikinci derece entropi, *Corr* korelasyon, *LH* yerel türdeşlik, *Dg* yönlülük, *U* ise eşbiçimlilik değerlerini belirtmektedir.

4.5.2. Dalgacık Dönüşümü Tabanlı Özellik Çıkarma Yöntemi

Dalgacık dönüşümü farklı frekans çözünürlüğüne sahip bilgilerin elde edilmesi sayesinde özellik çıkarma yöntemi olarak kullanılmaktadır [60]. Dalgacık dönüşümü tabanlı özellik çıkarma yöntemi, dalgacık dönüşümden gelen görüntülerin ağırlıklandırılmış standart sapmalarının hesaplanıp bir özellik vektörü oluşturulmasına dayanmaktadır [64]. Özellik vektörü Denklem 42'de görülmektedir.

$$x = \left\{ \begin{matrix} \sigma_1^{LH}, \sigma_1^{HL}, \sigma_1^{HH}, \frac{1}{2}\sigma_1^{LH}, \frac{1}{2}\sigma_1^{HL}, \frac{1}{2}\sigma_1^{HH}, \\ \dots, \frac{1}{2^{L-1}}\sigma_1^{LH}, \frac{1}{2^{L-1}}\sigma_1^{HL}, \frac{1}{2^{L-1}}\sigma_1^{HH}, \frac{1}{2^{L-1}}\sigma_1^{LL}, \tau^{LL} \end{matrix} \right\} \tag{42}$$

Uygulamalarda bütün görüntü yerine ayrıştırılmış tanınması beklenen belirli bir şablonun olduğu örüntü üzerinde bu dönüşümler uygulanmaktadır [60].

4.5.3. Geometrik Momentler

Moment özellikleri görüntüye ait dönme ve konumdan değişimsiz özelliklerinin elde edilmesini sağlar [60]. Boyut değişikliği ve açısal olarak dönmeden bağımsız olmayan momentler, Hu tarafından geometriden bağımsız hale getirilmiştir [60]. Görüntü üzerinde ele alınan iki boyutlu momentler aşağıdaki formül ile elde edilmektedir [64].

$$m_{pq} = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} x^p y^q g(x,y) \qquad p,q = 0,1,2,3 \ldots \tag{43}$$

burada *(x,y)* görüntü koordinatları, *g* görüntü, m_{pq} ise $p+q \leq n$ iken *n.* seviye moment ifadeleri, ve *N* ile *M* ise görüntünün boyutlarıdır. Momentlerin referans noktalarının merkeze taşınması durumunda moment ifadesi Denklem 44'deki gibi olacaktır. Bu durumda oluşan momentlere, merkez momentler adı verilmektedir. Merkez momentler görüntünün normal momentlerinin görüntü merkezine ötelenmesi ile elde edilmektedir [61]. Bundan dolayı merkez momentler, görüntünün ötelenmesinden etkilenmemektedir. Normalizasyon işlemi Denklem 45'de gösterilmektedir [61].

$$\varphi_{pq} = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} (x-\bar{x})^p (y-\bar{y})^q g(x,y), \quad \bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{44}$$

$$\eta_{pq} = \frac{\varphi_{pq}}{\varphi_{00}^{\gamma}}, \qquad \gamma = \frac{(p+q+2)}{2}, p+q = 2,3 \ldots \tag{45}$$

Hu tarafından ötelemeden bağımsız hale getirilen normalize merkez momentler aşağıdaki denklemlerde verilmiştir [61].

$$\emptyset_1 = \eta_{20} + \eta_{02} \tag{46}$$

$$\emptyset_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{47}$$

$$\emptyset_3 = (\eta_{30} - \eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{48}$$

$$\emptyset_4 = (\eta_{30} - \eta_{12})^2 + (\eta_{21} - \eta_{03})^2 \tag{49}$$

$$\emptyset_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} - \eta_{12})((\eta_{30} - \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2) + (3\eta_{21} - \eta_{03})(\eta_{21} - \eta_{03})((\eta_{30} - \eta_{12})^2 - (\eta_{21} - \eta_{03})^2) \tag{50}$$

$$\emptyset_6 = (\eta_{20} - \eta_{02})((\eta_{30} + \eta_{12})^2 - (\eta_{12} - \eta_{03})^2) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \tag{51}$$

$$\emptyset_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})^2((\eta_{30} - \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2) + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} - \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2) \tag{52}$$

4.5.4. Tchebichef Momentleri

Tchebichef momentleri, doğrudan görüntü koordinat uzayı üzerinden hesaplanmaktadır[65,66]. Tchebichef momentleri aşağıdaki gibi tanımlanmaktadır [65,66].

$$T_{pq} = \frac{1}{\varsigma(p,N)\varsigma(q,M)} \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} t_p(x - x_0) q(y - y_0) g(x,y) \tag{53}$$

burada *g(x, y) (N ×M)* boyutlarındaki görüntünün gri seviye yoğunluk fonksiyonu, *(x_0, y_0)* görüntünün merkez noktası ve t_n ise ölçeklenmiş tchebichef polinomlarıdır. Ölçeklenmiş tchebichef polinomları Denklem 54'te verilmektedir [65,66,67].

$$t_p(x) = \frac{(2p-1)t_1(x)t_{p-1} - (p-1)(1 - \frac{(p-1)^2}{N^2})t_{p-2}(x)}{p} \tag{54}$$

burada *p* Tchebichef polinom derecesi olup *p=2, 3 , ..., N-1* değerlerini içerir. t_0 ve t_1 ise Denklem 55'de gösterilmektedir [65].

$$t_0(x) = 1, \quad t_1(x) = \frac{2x + 1 - N}{N} \tag{55}$$

Denklem 53'de Tchebichef momentlerinin hesabında kullanılan *ζ (p, N)* Denklem 56'da gösterilmektedir ve

$$\zeta\,(p,N) = \sum_{x=0}^{N-1} \left| t_p(x) \right|^2 \tag{56}$$

şeklinde hesaplanmaktadır [65,66,67].

4.5.5. Zernike Momentler

Khotanzad ve Hong tarafından geliştirilen Zernike momentleri, değişimsiz özellik çıkarma yöntemi olarak sunulmuştur [68]. Zernike momentleri hesabında kullanılan Zernike karmaşık polinom seti tamamen dikgen olan bir kümede *x* ve *y* uzunlukları belirtmek üzere $x^2 + y^2 = 1$ olan bir eğri içerisinde geçerlidir [68]. Karesel bir şablonda elde edilen disk dışında kalan pikseller Zernike momentlerinin hesaplanmasında kullanılmayacaktır. Denklem 57'deki gibi belirtilen Zernike polinomlarında, $V_{pq}(x,y)$, *p* moment derecesini ve *q* moment tekrarlama derecesini belirtmektedir [68,69,70-72].

$$V_{pq}(x,y) = V_{pq}(p,\theta) = R_{pq}(\rho)\exp(jq\theta) \tag{57}$$

burada *p* tamsayı olup değeri $p \geq 0$'dır, *q* moment tekrarlama derecesi olup aldığı değerler *p-q=çift tamsayı* ve $|q| \leq p$ şartlarını sağlamalıdır, ρ parametresi *(x,y)* noktasının şablonun merkezine olan uzaklığıdır, θ değeri *r* vektörü ve *x* ekseni arasındaki saat yönü tersindeki açı değeridir Denklem 58 ve $R_{pq}(\rho)$ radyal polinom fonksiyonu olup Denklem 59'da gösterilmektedir [68].

$$\theta = \arctan\left(\frac{y}{x}\right) \tag{58}$$

$$R_{pq}(\rho) = \sum_{s=0}^{p-\frac{|q|}{2}} (-1)^s \frac{(p-s)!}{s!\left(\frac{p+|q|}{2}-s\right)!\left(\frac{p-|q|}{2}-s\right)!}\rho^{p-2s} \tag{59}$$

Denklem 59'dan de görüldüğü gibi $R_{n,-m}(\rho) = R_{n,-m}(\rho)$'dir. $f(x,y)$'nin iki boyutlu görüntü karesini temsil ettiği varsayılarak Zernike momentleri görüntü işlemlerinde ayrık olarak ele alınacağından Denklem 60'daki gibi gösterilebilir[68],

$$A_{pq} = \frac{p+1}{\pi}\sum_x\sum_y f(x,y)V'_{pq}(\rho,\theta) \tag{60}$$

burada $V'_{pq}(x,y)$ ise $V_{pq}(x,y)$'nin eşleniğidir ve A_{pq} ise *p* dereceli *q* tekrarlamalı Zernike momentleridir [60].

4.5.6. Değiştirilmiş Zernike Momentleri

Değiştirilmiş Zernike momentleri birim çember içerisindeki noktalar için hesaplama yapmaktadır [68,73]. Değiştirilmiş Zernike momentleri Denklem 61'deki gibi hesaplanmaktadır [68,74].

$$Z_{pq} = \frac{p+1}{\pi} \sum_{x} \sum_{y} f(x,y) V'_{pq}(\rho, \theta) \tag{61}$$

burada Z_{pq} p dereceli ve q tekrarlamalı değiştirilmiş Zernike momentidir, $V_{pq}(x,y)$ değiştirilmiş Zernike polinomu olup Denklem 57'deki gibi gösterilmektedir [68,74]. Fakat farklı olan değiştirilmiş Zernike momentlerde kullanılan $R_{pq}(\rho)$ radyal polinomudur. Radyal polinom Denklem 62'de gösterilmektedir [68,74].

$$R_{pq}(\rho) = \sum_{s=0}^{p-|q|} (-1)^s \frac{(2p+1-s)!}{s!\,(p+|q|+1-s)!\,(p-|q|-s)!} \rho^{p-s} \tag{62}$$

burada $0 \leq |q| \leq p$ ve $p \geq 0$ şartları sağlanmalıdır ve Zernike momentlerde de değiştirilmiş Zernike momentlerde de hesaplama yaparken $x^2 + y^2 > 1$ ise $V_{pq}(x,y) = 0$ sağlanmalıdır [68,74].

5. OPENCV

OpenCV, bir resim ya da video içindeki anlamlı bilgileri çıkarıp işleyebilmek için INTEL tarafından C ve C++ dilleri kullanılarak geliştirilmiş açık kaynak kodlu bir "Bilgisayarla Görme" kütüphanesidir [75]. OpenCV özellikle gerçek zamanlı uygulamalar için geliştirilmiştir. OpenCV' nin ticari kullanımı da dâhil ücretsiz kullanım sunması OpenCV' yi diğer görüntü işleme araçlarından üstün yapmaktadır.

Açık kaynak kodlu görüntü işleme kütüphanesi olan OpenCV'nin gelişimi dünyanın her tarafından yazılımcılarca sağlandığından projenin ismi "Open Source Computer Vision Library" nin yani Türkçesi ile "açık kaynak kodlu bilgisayarlı görü" 'nün kısaltılışından gelmiştir.

OpenCV kütüphanesi, BSD lisansı ile lisanslanmıştır. BSD lisansında kodu alan kişi, istediği gibi kullanma özgürlüğüne sahiptir [76]. Akademik ve ticari kullanımı ücretsiz olan bu kütüphane Windows, Linux, MacOS X gibi farklı platformlarda kullanılabilir [4].

OpenCV geliştirilirken C ve C++ dilleri seçilmiştir ve çok çekirdekli mimarilerin avantajları da burada kullanılmaya çalışılmıştır. Bu projenin ana takımını Intel oluşturmuştur. OpenCV'nin Intel işlemcilerde yüksek performansla çalışması için Integrated Performance Primitives (Gömülü Performans Tipleri) (IPP) geliştirilmiştir.

Yaklaşık 500'e yakın fonksiyon barındıran OpenCV Kütüphanesi makine öğrenmesinden robotiğe, kamera kalibrasyonundan tıpta görüntü işleme, otomatik tanımlama ve takip etme teknolojilerine kadar çok geniş alanda kullanıma hazır fonksiyonlar bulundurmaktadır.

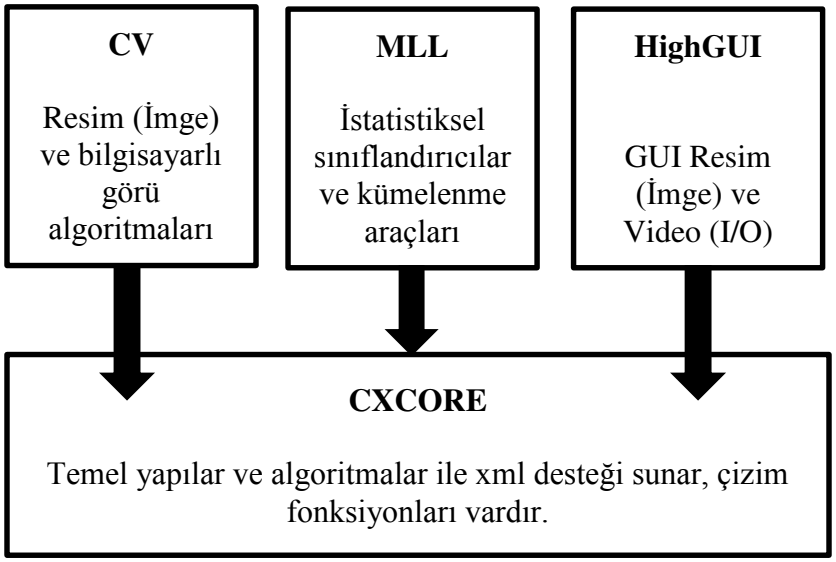
OpenCV'nin ayrıca çok gelişmiş makine öğrenme kütüphanesi Machine Learning Library (MLL) mevcuttur. Bu kütüphanede birçok fonksiyon ve algoritmalar bulunur.

5.1. OpenCV Bileşenleri

OpenCV kütüphanesi beş temel bileşenden oluşmaktadır;

- CV bileşeni
- MLL bileşeni
- HighGUI bileşeni

- CXCore bileşeni
- CvAux bileşeni



Şekil 5.1. OpenCV kütüphanesini oluşturan bileşenler

5.1.1. CV Bileşeni

Temel resim işleme fonksiyonları ve Bilgisayarla Görü/Görme için kullanılan yüksek seviyeli algoritmaları bünyesinde barındıran beş temel kütüphaneden biridir. CV bileşeni ile basit resim işleme, filtre işlemleri, geometrik dönüşümler, renk dönüşümleri, özellik seçimi, morfolojik işlemler, kontur alma, histogram işlemleri, şekil tanımlama işlemleri ve kamera kalibrasyonu gibi işlemler gerçekleştirilebilmektedir.

5.1.2. MLL Bileşeni

Makina öğrenmesi dalı için gerekli istatistiksel verilere ulaşmak, mevcut verileri sınıflandırmak için kullanılan fonksiyonları/araçları içeren diğer bir kütüphanedir. İstatistiksel model işlemleri, destek vektörü makine işlemleri, yapay sinir ağları işlemleri gibi birçok işlem bu bileşen sayesinde rahat bir şekilde gerçekleştirilebilmektedir.

5.1.3. HighGUI Bileşeni

Kayan form gibi pek çok nesneyi yaratabilmemizi sağlayan bir grafik arabirimi olmakla beraber, resim ve videoları kaydetmek, yüklemek, hafızadan silmek için gerekli giriş/çıkış (I/O) fonksiyonlarını da içeren bir kütüphanedir [77]. Görüntü/video elde etme ve basit arayüz elde etme gibi işlemler bu bileşen sayesinde gerçekleştirilmektedir.

5.1.4. CXCore Bileşeni

OpenCV kütüphanesine ait çeşitli veri yapılarını (cvPoint, cvSize, IplImage, cvHistogram, cvMat) bünyesinde barındıran, XML desteği de sağlayan bir kütüphanedir. Matris işlemleri, matematiksel işlemler, 2D grafik çizimi ve karmaşık dizilerdeki işlemler bu bileşen sayesinde basit bir şekilde gerçekleştirilebilmektedir.

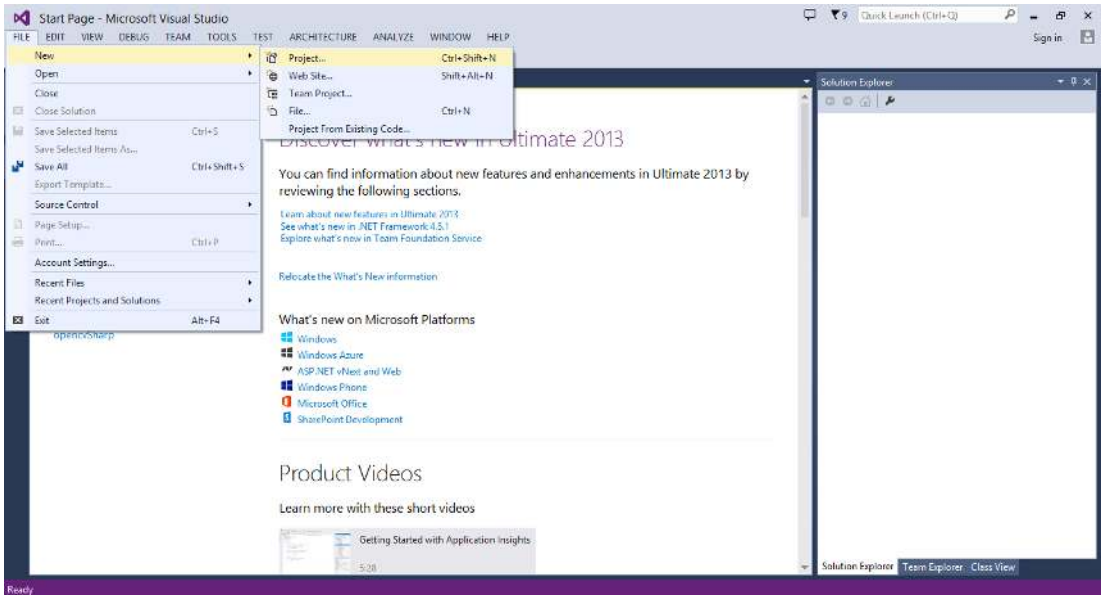
5.1.5. CvAux Bileşeni

Pek çok deneysel algoritmaları barındırır. CvAux bileşeni, şablon eşleştirme (template-matching), şekil eşleştirme (shape matching), bir objenin ana hatlarını bulma (finding skeletons), yüz tanıma (face-recognition), ağız hareketleri izleme (mouth-tracking), vücut hareketlerini tanıma (gesture recognition) ve kamera kalibrasyonu gibi daha pek çok deneysel algoritmaları bünyesinde barındıran kütüphanedir [78].

5.2. OpenCV Kurulumu

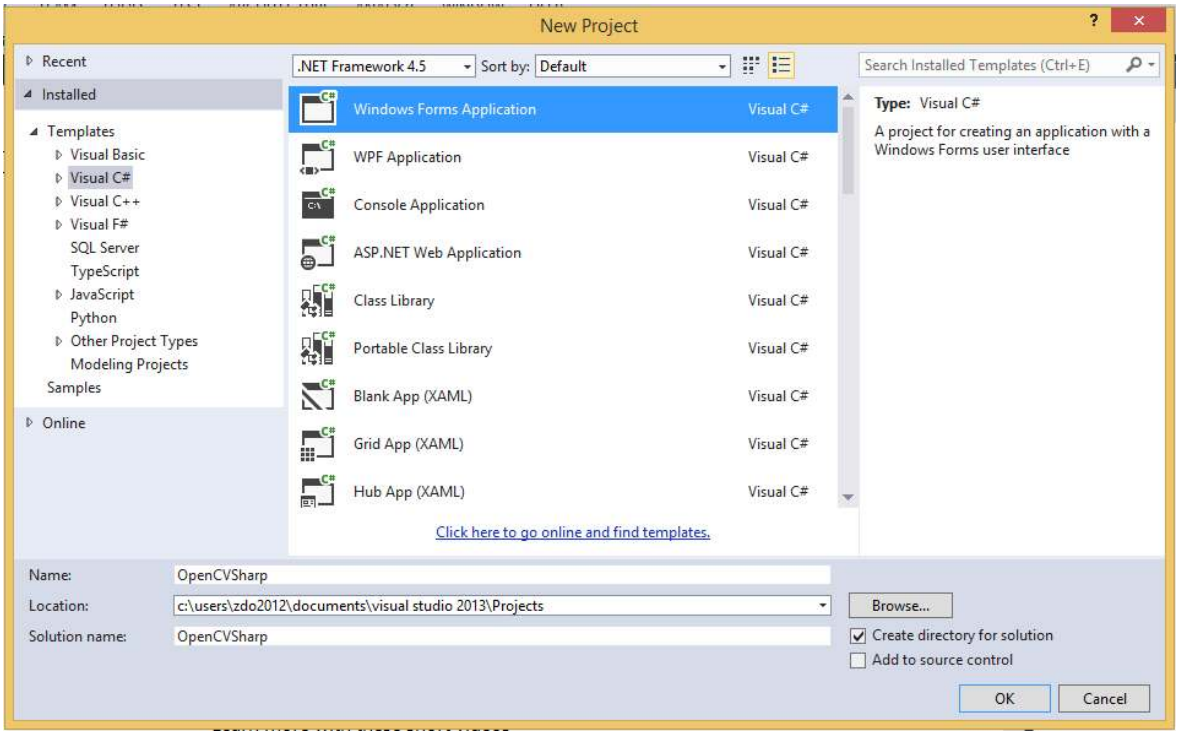
OpenCV kütüphanesi Microsoft Visual Studio, Phython, Delphi gibi birçok platform üzerine eklenerek kullanabilmektir. Uygulamamızda OpenCV kütürphanesi Microsoft Visual Studio C# ile birlikte kullanılacaktır. Bu nedenle OpenCV kütüphanesi Microsoft Visula Studio 2013 plarformunda C# Windows uygulamasına entegre edilecektir.

OpenCV kütüphanesinin Visual Studio 2013 C# uygulamasına ekleyebilmek için öncelikle Visual Studio 2013’ ü açıp bir C# projesi oluşturulmalıdır.



Şekil 5.2. Visual Studio 20013' de yeni C# projesi oluşturma

Şekil 5.2' deki gibi Visual Studio 2013 programı açıldıktan sonra File → New → Project (Ctrl + Shift + N) yolu kullanılarak yeni bir proje oluşturmak için ilk adım uygulanmaktadır.

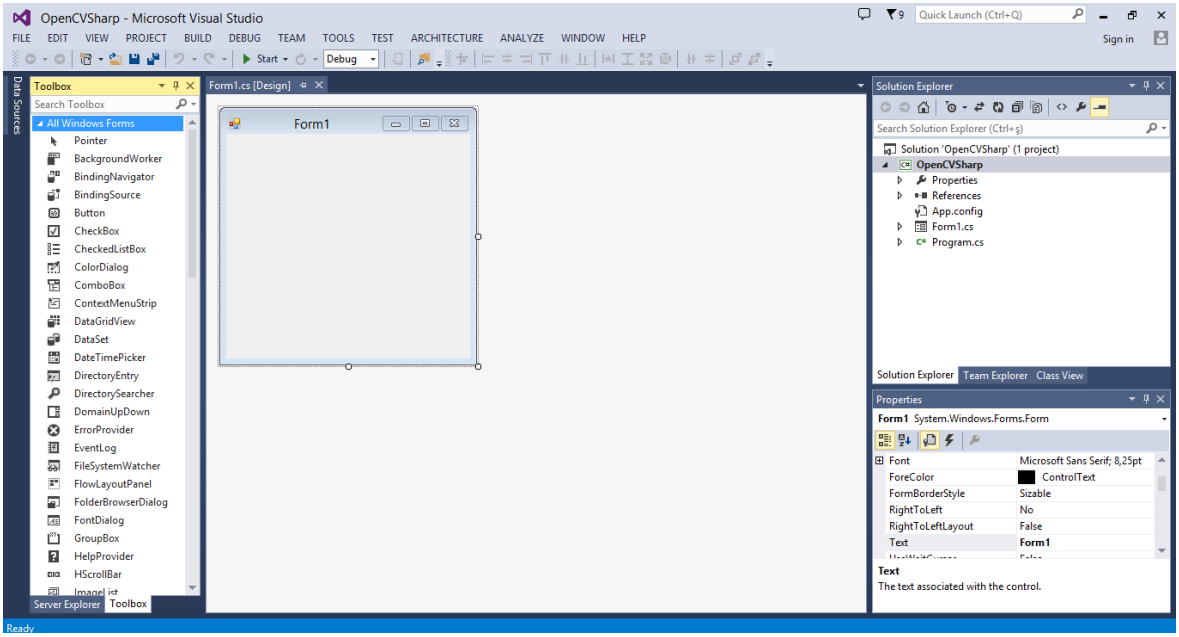


Şekil 5.3. Visual Studio 20013' de C# projesine isim verme ve kayıt yeri belirleme

İkinci adım olarak Şekil 5.3' deki pencere geldikten sonra "Template" kısmından Visual C# ve pencerenin orta bölmesinde yer alan ve form uygulaması oluşturmak için kullanılan "Windows Forms Application" seçilmektedir. Oluşturulacak olan C# form uygulamasına pencerenin alt kısmında bulunan "Name" metin alanına proje için bir isim yazılmaktadır. Son olarak da projenin nereye kaydedileceğini belirlemek için yine pencerenin alt kısmındaki "Location" metin alanına proje yolunun yazılabileceği gibi "Browse" butonuna basılarak açılacak pencere yardımıyla da bir yol belirlenebilmektedir. İlgili ayarlar yapıldıktan sonra "OK" butonuna basılıp yeni bir C# Windows form uygulaması oluşturulur.

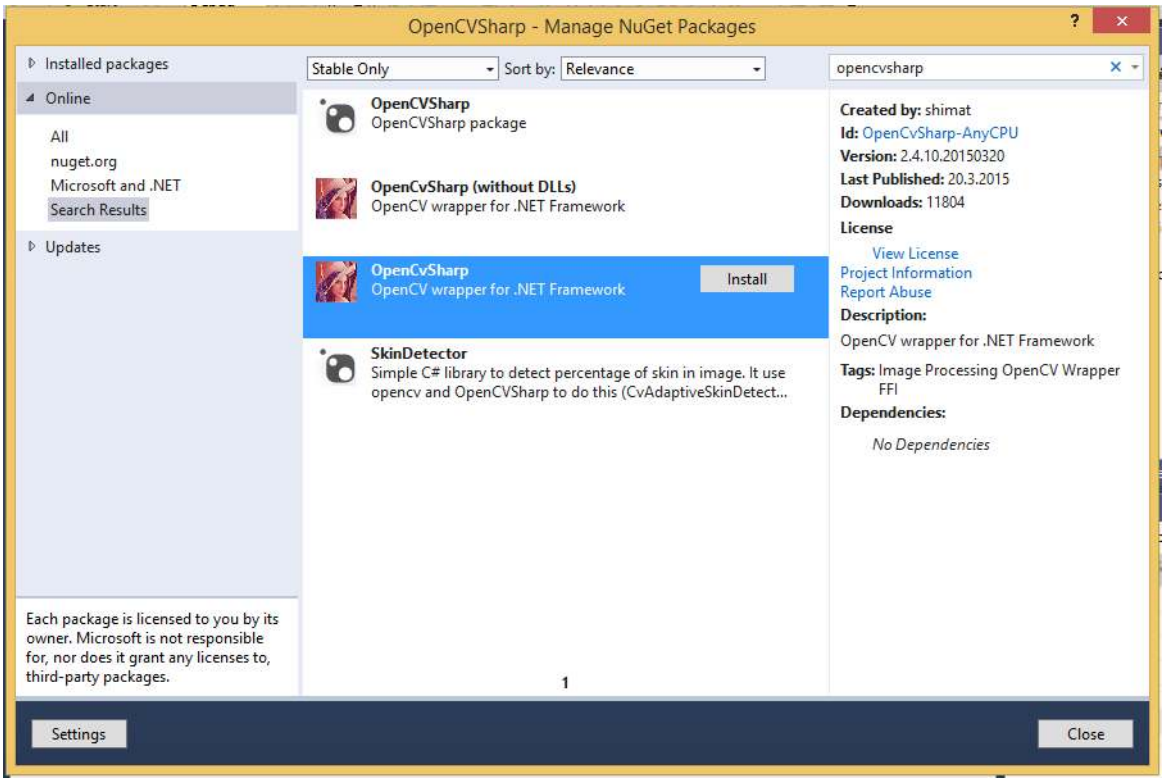
Şekil 5.4' deki gibi yeni bir C# Windows form uygulaması açılmaktadır. OpenCv kütüphanesinde bulunan görüntü işleme elemanlarını kullanabilmek için OpenCv kütüphanesini açılan bu C# Windows form uygulamasına entegre etmek gerekir. Bunun için karşımıza iki seçenek çıkmaktadır. Birinci seçenekte gerekli dll dosyaları indirilip el ile ekleme işlemi gerçekleştirilir. İkinci olarak ise NuGet kullanarak otomatik olarak eklenebilir.

Bu bölümde OpenCv kütüphanesini C# Windows form uygulamasına eklemek için ikinci yöntem olan NuGet kullanılıp otomatik ekleme işlemi gösterilmektedir.



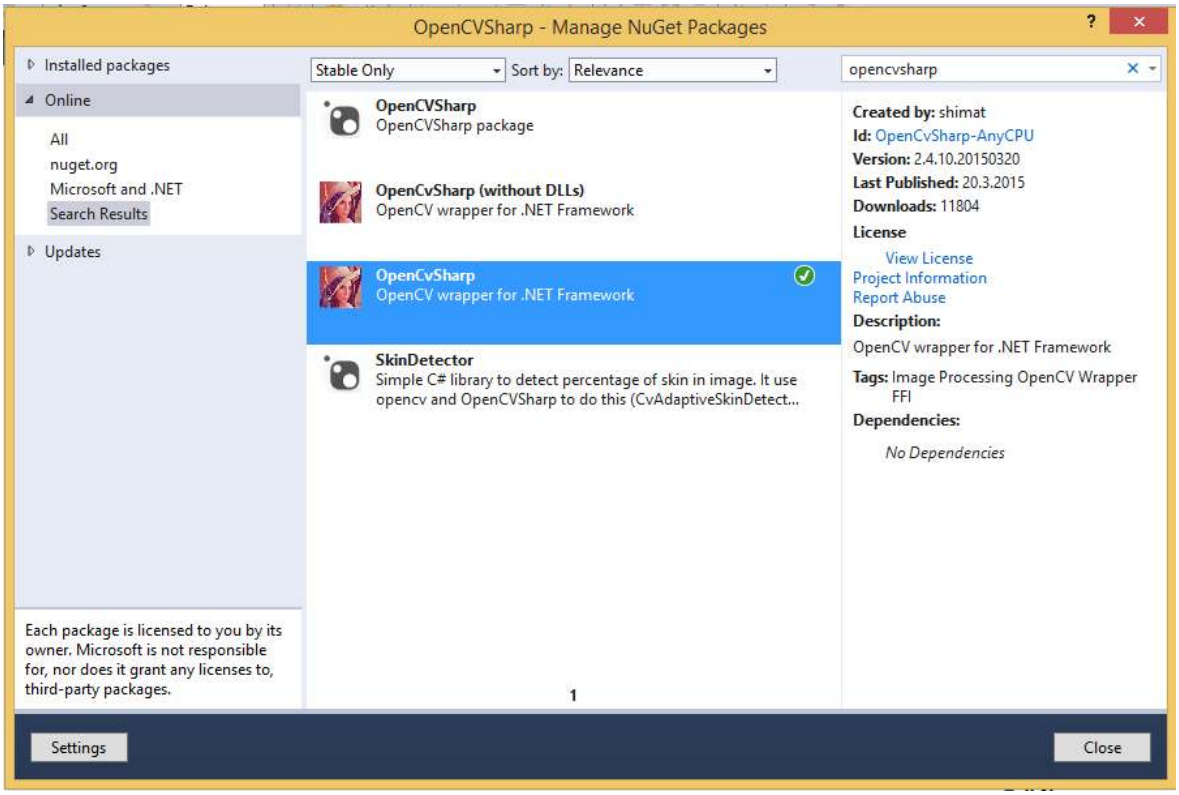
Şekil 5.4. Visual Studio 20013' de açılmış yeni bir C# projesi

OpenCV kütüphanesini ekleme adımına geçmek için Şekil 5.4' deki pencere açıldıktan sonra Project → Manage NuGet → Packages yolu kullanılarak Şekil 5.5' deki pencere açılmaktadır.



Şekil 5.5. C# projesine OpenCV kütüphanesinin eklenmesi

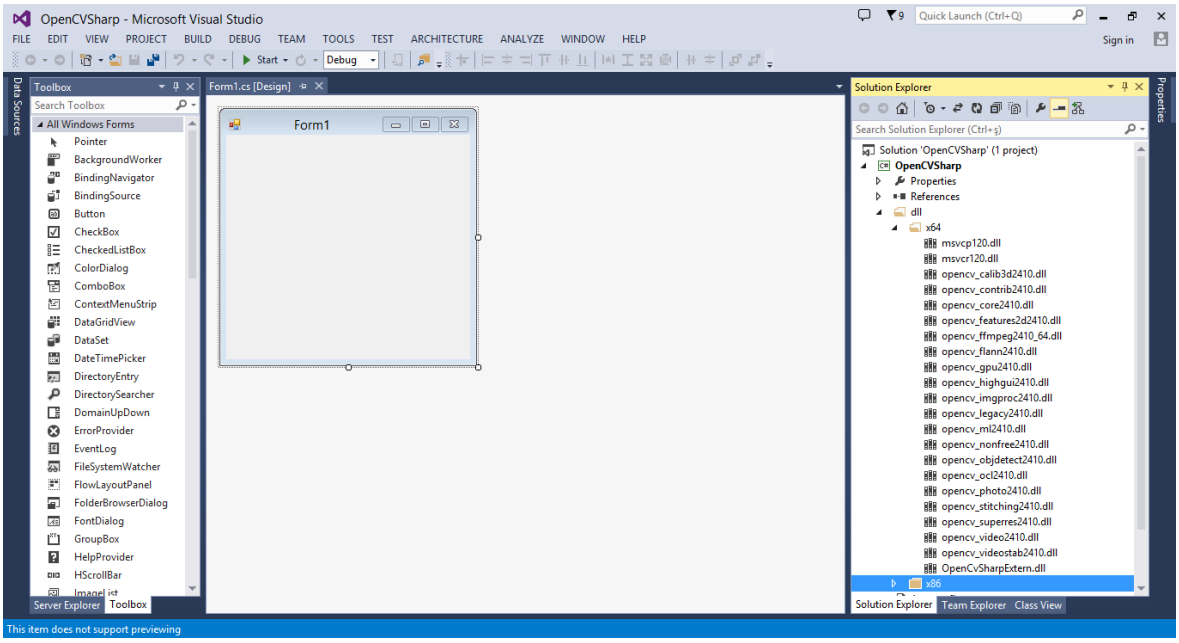
Şekil 5.5' deki pencerede sol bölmede bulunan "Online" seçildikten sonra "Search" metin kutusuna "opencvsharp" yazıp arama işlemi gerçekleştirilir. Dikkat edilmesi gereken husus bilgisayarın bu işlemleri gerçekleştirirken internete bağlı olmasıdır. Arama gerçekleştirildikten sonra Şekil 5.5' deki pencerede orta kısımda görünen seçeneklerden "OpenCvSharp" seçeneğinin yanındaki "Install" butonuna basarak OpenCv kütüphanesi projeye yüklenir.



Şekil 5.6. OpenCvSharp yüklemesinin tamamlanması

Install butonuna basıp kısa bir süre bekledikten sonra pencerenin orta kısmında bulunan ve daha önce Install butonun bulunduğu yerde artık yeşil renkte kütüphanenin yüklendiğini gösteren onay simgesi görünmektedir.

Şekil 5.7' de görüldüğü gibi Solution Explorer penceresinde yüklü dll dosyaları görünmektedir. Böylelikle OpenCV Kütüphanesi C# Windows form uygulamasına eklenmiştir.



Şekil 5.7. OpenCv dll dosyalarının Solution Explorer'da görüntülenmesi

Kod yazarken öncelikle OpenCv kütüphanesini çağırmak için C# kod ekranının en başındaki using satırlarına OpenCv kütüphanelerini çağırıp komutları kullanabilmek için;

```
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.DebuggerVisualizers2013;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
```

kod satırları eklenmelidir. Bu kod satırları eklendikten sonra artık OpenCv kütüphanesinde bulunan komutlar C# uygulamasında rahatlıkla kullanılabilir.

5.3. OpenCV'de Temel Görüntü İşleme Komutları

OpenCV optimize edilmiş ve gerçek zamanlı uygulamalar tasarlanması amacı ile ortaya çıkmıştır. Bunlardan bazı özellikler aşağıda listelenmiştir.

Özellikleri:

- Görüntü verileri işleme (tanımlama, kaydetme, kopyalama, özellik ayarlama, dönüşüm).
- Resim ve video Giriş / Çıkış (I / O, video dosyası ve kamera tabanlı girişi, resim).
- Matris ve vektör manipülasyonu ve lineer cebir rutinleri.
- Çeşitli dinamik veri yapıları (listeler, kuyruklar, ağaçlar, grafikler).
- Temel görüntü işleme (filtreleme, kenar algılama, köşe tespit, örnekleme ve interpolasyon, renk dönüşümü, morfolojik işlemler, histogram, görüntü piramitleri).
- Kamera kalibrasyonu (bulma ve izleme kalibrasyon desenleri, kalibrasyon temel matris tahmini).
- Hareket analizi (optik akış, hareket segmentasyonu, izleme).
- Nesne tanıma (öz – yöntemleri (eigen – methods)).
- Temel GUI (ekran görüntü / video, klavye ve fare kullanımı, kaydırma çubukları)
- Resim etiketleme (çizgi, çokgen, konik, metin yazma).

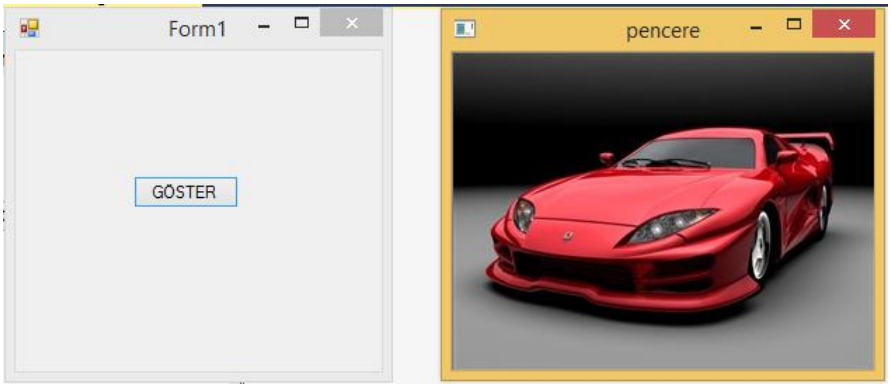
5.3.1. OpenCV ile Resmin Görüntülenmesi

Resimler **IplImage** tipindeki değişkenlerde tutulur. IplImage veri yapısı, çeşitli resim dosyalarını hafızaya alabilmek için oluşturulmuş özel bir veri yapısıdır. Bu değişkenlere **LoadImage** fonksiyonu ile resim yüklenir. Bu fonksiyon, BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF ve TIF uzantılı resim dosyalarını okuyabilmektedir[62].

NamedWindow() fonksiyonu pencere ismi ve flag değişkeni alarak, yeni bir pencere oluşturur.

ShowImage() fonksiyonu içine daha önceden oluşturulmuş pencere ismi ve resim kaynağı verilerek, resmin ekranda gösterilmesi sağlanır.

Şekil 5.8' deki gibi pir form tasarlandıktan sonra Ek I'de verilen program *"GÖSTER"* butonuna yazılarak butona tıklandığında resmin bir IplImage tipindeki değişkene yüklenip oluşturulan bir pencere yardımıyla ekranda görüntülenmesi sağlanmaktadır.



Şekil 5.8. OpenCv'de resmin görüntülenmesi

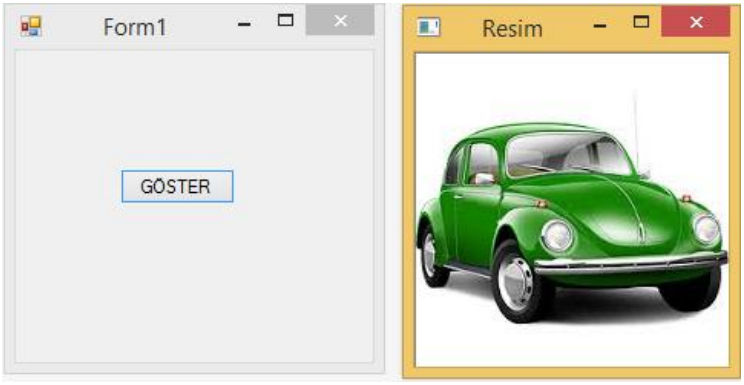
5.3.2. OpenCV ile Resmin Matris Olarak Alınıp Görüntülenmesi

OpenCV' de resmi bir matris olarak tutup matris kütüphanesindeki fonksiyonlardan faydalanılabilinir. **Imread** fonksiyonu ile resim okunur. Imread fonksiyonun döndürmüş olduğu değer bir matrisi temsil eder. Matris olarak alınan resim üzerinde toplama, çarpma, bölme gibi temel matris işlemleri ile birlikte çeşitli filtreleme işlemleri yapılabilmektedir.

Ek II' de verilen program kodları, bir resmin OpenCv kütüphanesi ile matris olarak nasıl alınacağını göstermektedir. Şekil 5.9' daki form tasarlanıp *"GÖSTER"* butonuna basıldığında resim bir matris olarak alınıp ekranda gösterilmektedir.

ImRead() matris türündeki değişkenlere resmi yüklemek için kullanılan bir fonksiyondur. Bu fonksiyonla yüklenen resimler "Mat" türündeki değişkenlerde tutulmaktadır.

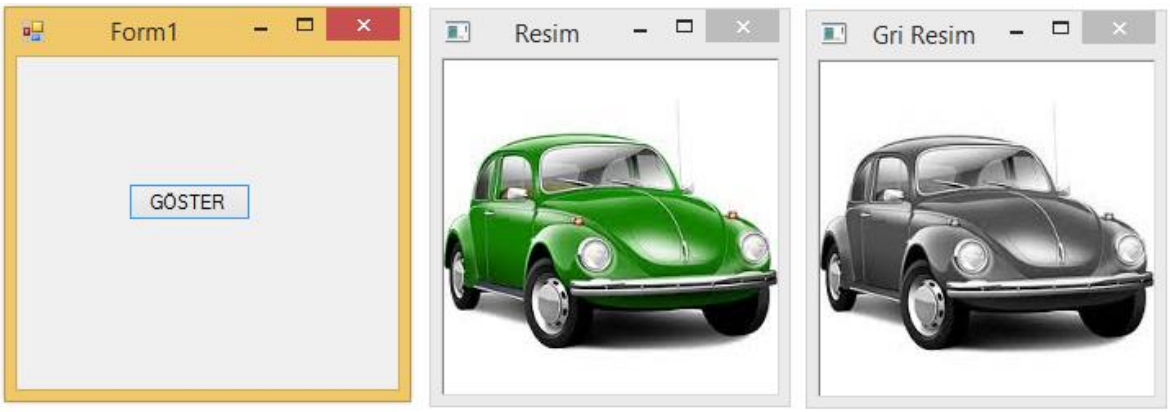
ImShow() matris türündeki değişkenlerin içerisine yüklenen resimler bu fonksiyonla ekranda görüntülenmektedir.



Şekil 5.9. OpenCv'de resmin matris olarak alınması ve görüntülenmesi

5.3.3. OpenCV'de Matris Olarak Alınan Resmin Renk Dönüşümünü Yapmak

OpenCV' de matris olarak alınan bir resmin renk dönüşümünü yapmak için **CvtColor** fonksiyonu kullanılmaktadır. Ek III' de verilen program kodlarıyla matris olarak alınan RGB uzay modelindeki renkli bir resim gri tona çevrilmiştir. İlgi kodlar *"GÖSTER"* butonuna yazılıp basıldığında Şekil 5.10' daki gibi bir sonuç vermektedir.

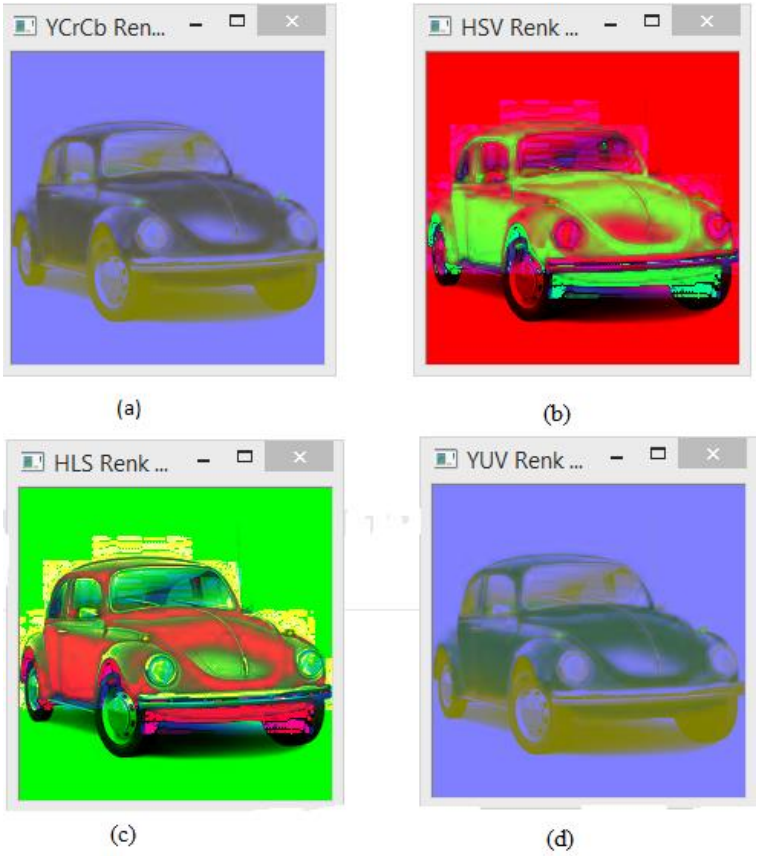


Şekil 5.10. OpenCv'de renkli resmin gri resme dönüştürülmesi

CvtColor fonksiyonu renk dönüşümü işlemini yaparken kaynak, hedef ve kod parametreleriyle çalışır. Kaynak renk dönüşümü yapmak istediğimiz resim değişkeni belirtmektedir. Hedef kaynak resmin dönüşüm yapıldıktan sonra atama yapılarak yeni resmin içerisinde tutulduğu değişkeni göstermektedir. Kod ise yapılmak istenen renk dönüşümünü ifade etmektedir. Bu uygulamada yapılmak istenen renk dönüşümü RGB renk modelinden gri tonlamalı renge dönüştürme işlemidir. Bu dönüşüm için **BgrToGray** parametresi kullanılmıştır. Remin görüntülenmesi için ise **ImShow** fonksiyonu kullanılmıştır.

- **RGB ←→ GRAY:** BgrToGray, GrayToBgr parametreleri ile RGB renk uzayı ile GRAY renk uzayı arasında dönüşüm yapılabilmektedir.
- **RGB ←→ YCrCb:** BgrToCrCb, CrCbToBgr parametreleri ile RGB renk uzayı ile YCrCb renk uzayı arasında dönüşüm yapılabilmektedir.
- **RGB ←→ HSV:** BgrToHsv, HsvToBgr parametreleri ile RGB renk uzayı ile HSV renk uzayı arasında dönüşüm yapılabilmektedir.

- **RGB ←→ HLS:** BgrToHls, HlsToBgr parametreleri ile RGB renk uzayı ile HLS renk uzayı arasında dönüşüm yapılabilmektedir.
- **RGB ←→ YUV:** BgrToYuv, YuvToBgr parametreleri ile RGB renk uzayı ile YUV renk uzayı arasında dönüşüm yapılabilmektedir.

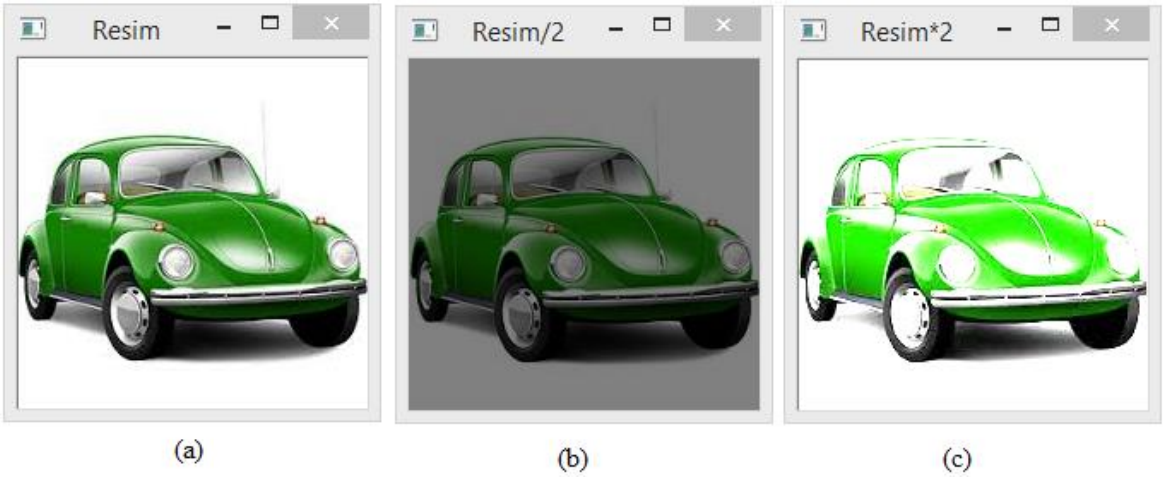


Şekil 5.11. Resmi matris olarak alma ve üzerinde renk dönüşümü işlemlerini yapma. (a) YCrCb renk uzayı, (b) HSV renk uzayı, (c) HLS renk uzayı, (d) YUV renk uzayı

5.3.4. Aritmetiksel İşlemlerle Resmin Kontrastını Arttırmak ve Azaltmak

Resim matris olarak alındıktan sonra basit aritmetiksel işlemlerle resmin kontrastı düşürülüp arttırılabilmektedir. Bir gri resmin data kısmı *unsigned char* olarak tutulup piksel değerleri *0 – 255* arasındadır. 0 siyahı 255 beyazı göstermektedir. Bu değerler arasındaki diğer değerler ise grinin diğer tonlarını temsil etmektedir. RGB renk uzayıyla oluşturulmuş bir resimde aynı mantıkla *0 – 255* arası değerler alır fakat gri tondan farklı olarak üç kanal için ayrı ayrı değerler bulunmaktadır. *(0, 0, 0)* siyah *(255, 255, 255)* beyazı temsil eder. Bu değerlere çarpma bölme gibi aritmetiksel işlemler uygulanarak kontrast değeri

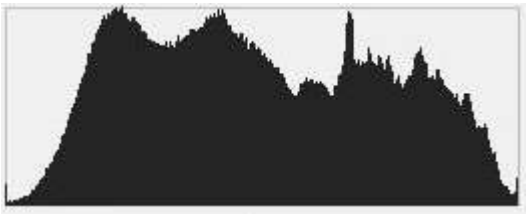
değiştirtilebilir. *Resim/2* gibi basit bir işlem ile resmin datasındaki değerler düşürülerek resmin kontrastı azaltılmış olur, *Resim x 2* gibi bir işlem uygulandığında ise resmin datasındaki değerler arttırılarak resmin kontrastı arttırılır. Ek – IV' deki kodlar *"GÖSTER"* butonuna yazılıp basıldığında Şekil 5.12' daki gibi bir sonuç vermektedir.



Şekil 5.12. Resmi matris olarak alma (a) orjinal resim, (b) kontrastı azaltılmış resim, (c) kontrastı arttırılmış resim

5.3.5. OpenCV' de Histogram İşlemleri

Histogram resmi oluşturan piksellerin ışık değerleri ile oluşturulmuş grafiktir. Histogram grafiği *X* ekseninde ışık değerleri, *Y* ekseninde ise piksel sayısı olacak şekilde oluşturulmaktadır. Histogramda sol kısımdan sağ kısma doğru gidildiğinde ışık değeri 0 dan 255 'e doğru artmaktadır. Şekil 5.13' de bir histogram grafiği örneği görülmektedir.



Şekil 5.13. Histogram grafiği

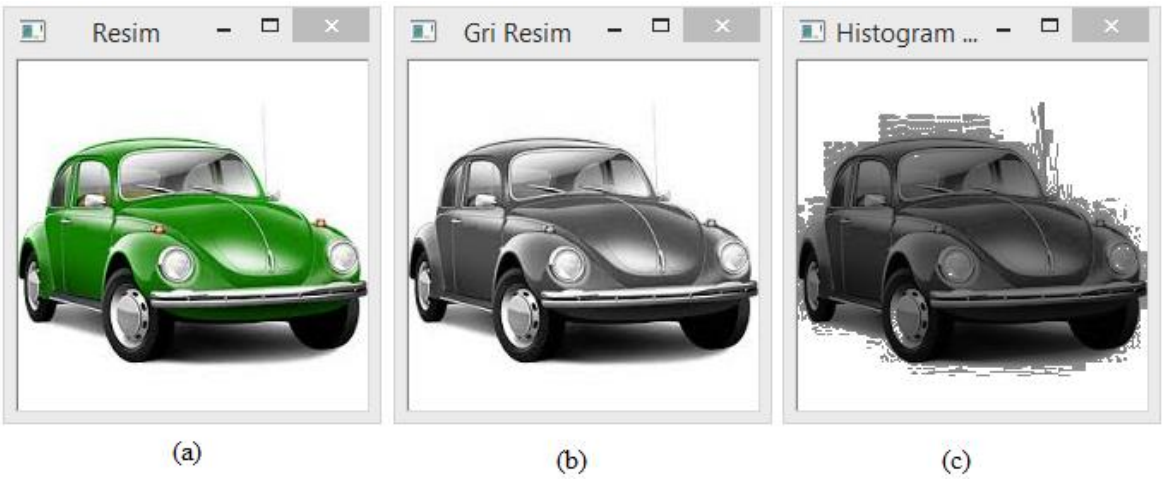
Görüntü işlemede histogramlar kullanılarak görüntü daha belirgin hale getirilebilmektedir. Görünü iyileştirme için kullanılan histogram türlerinden en çok kullanılanlar;

- Histogram Eşitleme (Histogram Equalization)
- Adaptif Histogram

olarak bilinmektedir.

5.3.5.1 Histogram Eşitleme (Histogram Equalization)

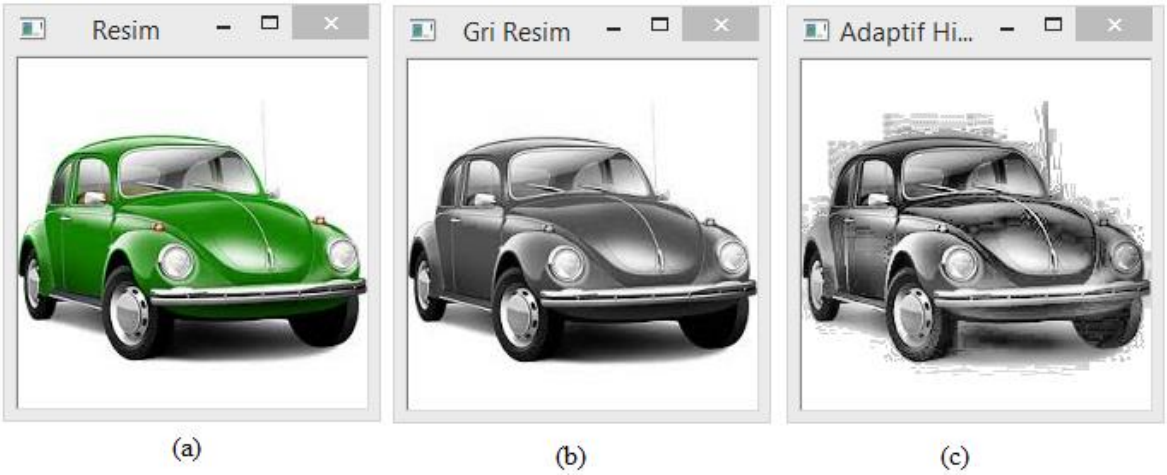
Histogram eşitleme (histogram equalization), bir görüntüdeki renk değerlerinin belli bir yerde kümelenmiş olmasından kaynaklanan renk dağılımı bozukluğunu gidermek için kullanılan bir yöntemdir. Ek – V' de verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme histogram eşitleme işlemi uygulanarak Şekil 5.14' deki resimler elde edilmiştir.



Şekil 5.14. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) histogram eşitleme uygulanmış resim

5.3.5.2 Adaptif Histogram

Adaptif histogram eşitleme değiştirilmiş bir histogram eşitleme işlemidir. Yani yerel veri üzerinde iyileştirme işlemini gerçekleştirmektedir. Buradaki ana düşünce görüntü ızgara şeklinde dikdörtgen bölgelere bölünür ve her bir bölgeye standart histogram eşitleme işlemi uygulanır [80]. Ek – VI' da verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme adaptif histogram eşitleme işlemi uygulanarak Şekil 5.15'deki resimler elde edilmiştir.



Şekil 5.15. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) adaptif histogram eşitleme uygulanmış resim

5.3.6. OpenCV' de Filtrelerin Kullanılması

Resimdeki gürültüyü azaltmak, işlem süresini ve bilgisayar kaynaklarını verimli kullanmak gibi amaçları vardır. Parlaklık ve gürültü gibi resimdeki istenmeyen öğeleri yok etmektedirler.

Lineer ve lineer olmayan filtre versiyonları mevcuttur. Filtreler genel olarak, bir kernel (çekirdek) matrisin tüm resim boyunca gezdirilip, her piksel için çarpıların toplamının esasına dayanır. Böylelikle her piksel değeri için yeni bir değer hesaplanır. Matrisin büyüklüğü ve içerdiği değerlere göre resmi yumuşatma, keskinleştirme, kenar tespiti gibi birçok işlem yapılabilmektedir.

Genel olarak filtreler *pre – processing (ön işleme)* olarak kullanılır ve asıl yapılacak işlerin daha rahat yapılmasını sağlarlar. Örneğin; çok parlak bir resimde ya da gürültünün (noise) çok bulunduğu bir resimde ne kadar isterseniz de istediğiniz nesneyi tespit etmeniz çok zorlaşacaktır [81].

5.3.6.1 OpenCV' de Median (Ortanca) Filtrenin Kullanılması

Median filtre genellikle *salt & pepper* denilen bozulma türünü gidermek için kullanılmaktadır. Bu bozulmayı düzeltmek için bir kernel matris uygulanmaktadır.

4	5	6
1	7	8
3	2	9

Şekil 5.16. 3x3' lük örnek kernel matrisi

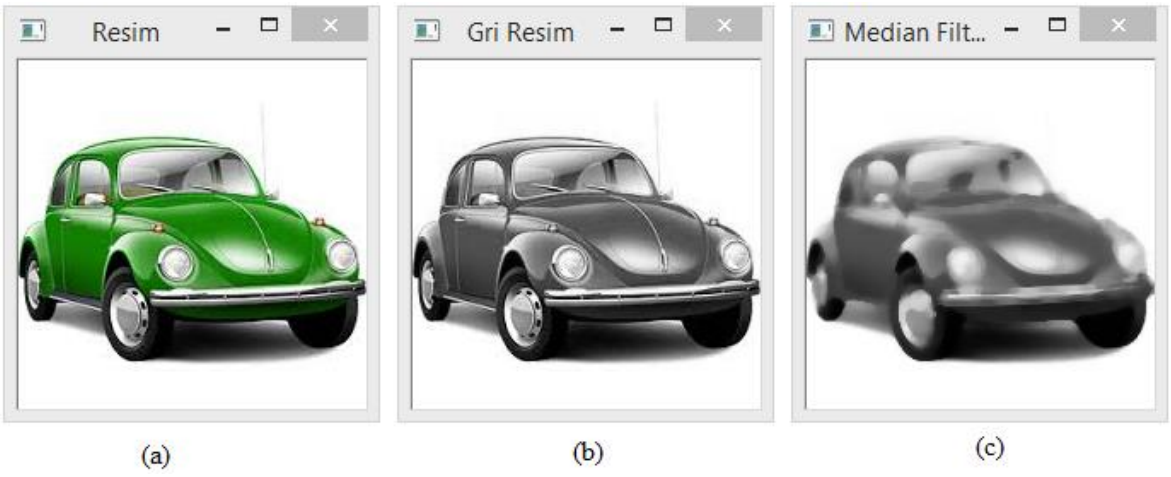
Şekil 5.16' daki gibi *3 x 3* lük bir kernel matrise median filtre uygulandığında filtre, matris elemanlarını büyükten küçüğe doğru sıralayacak ve ortanca elemanı matrisin ortasına yani *(2,2)* numaralı indise yerleştirecektir.

1	2	3
4	5	6
7	8	9

Şekil 5.17. 3x3' lük örnek kernel matrisi median filtre uygulandıktan sonra

Matris elemanlarını sıralayacak olursak *1, 2, 3, 4, 5, 6, 7, 8, 9* olduğu görülmektedir ve ortanca elemanın *5* olması nedeniyle *(2,2)* indisine *5* yerleşecektir. Bu şekilde görüntüde iyileştirme yapılmış olur.

Ek – VII'de verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme median filtre işlemi uygulanarak Şekil 5.18'deki resimler elde edilmiştir.



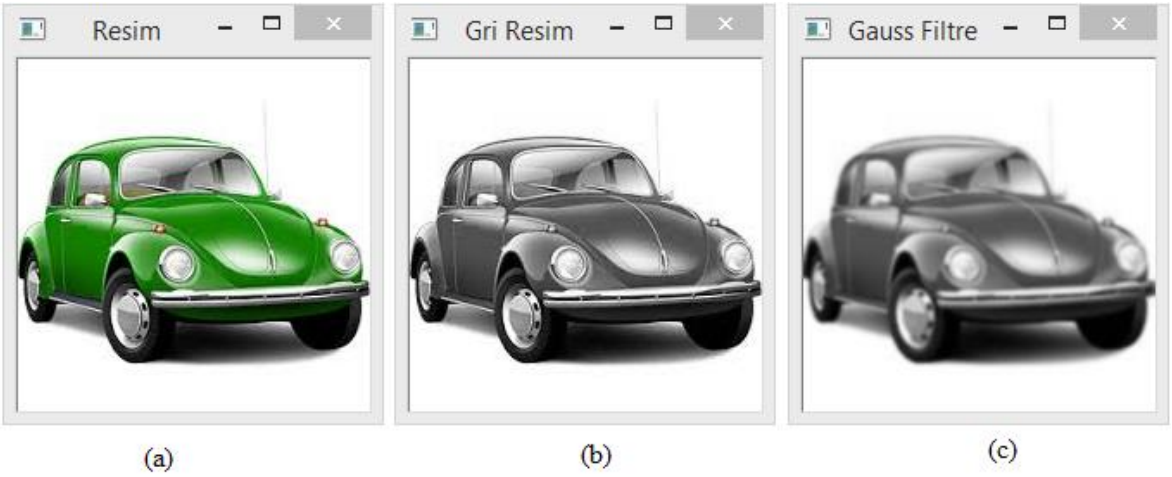
Şekil 5.18. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) median filtre uygulanmış resim

5.3.6.2 OpenCV' de Gauss Filtresinin Kullanılması

Median filtrenin Gaussian dağılımını kullanarak biraz daha değiştirilmiş hali Gaussian filtresi olarak bilinir. Gaussian filtreleme aynı zamanda bir fourier dönüşümüdür. Gauss filtre ile sonsuz bir transfer fonksiyonuna karşılık mekânsal alanda sonlu bir pencerede (tarama penceresi) filtreleme yapılabilmektedir. Bu da filtrelemenin temel problemini daha kolay çözülebilir hale getirir. Gauss filtresi uygulandığı resme belirtilecek maske boyutunda bir matris ile tarama işlemi yapmakta ve her bir matriste ağırlıklı ortalamayı hesaplamaktadır.

Gauss yumuşatmasının avantajları, filtreleme önce yatay ardından çıkan sonuçla düşey eksende gerçekleştirilebilir.

Ek – VIII' de verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme gauss filtreleme işlemi uygulanarak Şekil 5.19' daki resimler elde edilmiştir.



Şekil 5.19. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) gauss filtre uygulanmış resim

5.3.7. OpenCV' de Morfolojik İşlemler, Eşikleme ve Bağlı Bileşenler Analizi

Morfolojik işlemler çok geniş bir kullanım alanına sahiptir; gürültünün kaldırılması, bazı elementlerin izole edilmesi, kopuk elementlerin birleştirilmesi örnek olarak verilebilir. Morfolojik işlemler genellikle binary resimlerde kullanılmaktadır. Resmi binary resme dönüştürebilmek için resme eşikleme (thresholding) işlemi uygulanmaktadır.

5.3.7.1 OpenCV' de Eşikleme

Eşikleme (thresholding) genellikle resmi binary resme (ikili yani siyah / beyaz resim) çevirmek için kullanılan bir tekniktir. Gürültüyü önlemek veya nesne belirlemek gibi farklı amaçlarla da kullanılabilmektedir. Genel olarak belirlenen bir sınır değeriyle resmin tüm piksellerinin teker teker karşılaştırılması ve belirli bir kurala göre resmi *0* (sıfır) ve *1* (bir)'e dönüştürme uygulamasıdır. Farklı seçenekleri bulunmaktadır.

OpenCV içerisinde eşikleme için kullanılan fonksiyonlar:

- **THRESH_BINARY**

Resimdeki piksel sınır değerinden (thresh) büyükse maksimum olarak belirlenen *255* değeri verilir, küçükse minimum olarak *0* değeri verilir. Bu durumda sınırdan büyükler beyaz olurken sınırdan küçükler siyah olacaktır.

Ek – IX' da verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme eşikleme işlemi uygulanarak Şekil 5.20' deki resimler elde edilmiştir.

Şekil 5.20. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) Binary tipinde eşikleme yapılmış resim

- **THRESH_BINARY_INV**

THRESH_BINARY' nin tam olarak zıttıdır. Piksel değeri sınır değerinden büyük ise *0* (sıfır), piksel değeri sınır değerinden küçük ise *255* değeri verilir. Yani piksel sınırdan büyükse siyah küçükse beyaz olacaktır.

Ek – X' da verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme eşikleme "binary" işlemi uygulanarak Şekil 5.21' deki resimler elde edilmiştir.

Şekil 5.21. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) BinaryInv tipinde eşikleme yapılmış resim

- **THRESH_TOZERO**

Bu yönteme göre piksel değeri sınırdan büyük ise kendi değerinde kalacak, eğer piksel değeri sınırdan küçük ise *0* (sıfır) olarak değiştirilecektir. Yani piksel değeri sınırdan büyük olduğu yerlerde aynen kalacak, küçük olduğu yerlerde ise siyah olacaktır.

Ek – XI' de verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme eşikleme "binary" işlemi uygulanarak Şekil 5.22' deki resimler elde edilmiştir.

Şekil 5.22. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) ToZero tipinde eşikleme yapılmış resim

- **THRESH_TOZERO_INV**

THRESH_TOZERO' nun tam tersidir. Yani piksel değerinin sınırdan büyük olduğu yerler siyah olurken küçük olduğu yerler aynen kalacaktır. İki durumda da sonuç tam binary olmasa da kullanışlı olduğu yerler vardır.

Ek – XII' da verilen program kodlarıyla resim matris formatıyla alınıp gri tona çevrilmiş ve gri tonlu resme eşikleme (thresholding) "binary" işlemi uygulanarak Şekil 5.23' deki resimler elde edilmiştir.

Şekil 5.23. Resmi matris olarak alma (a) orjinal resim, (b) gri resim, (c) ToZeroInv tipinde eşikleme yapılmış resim

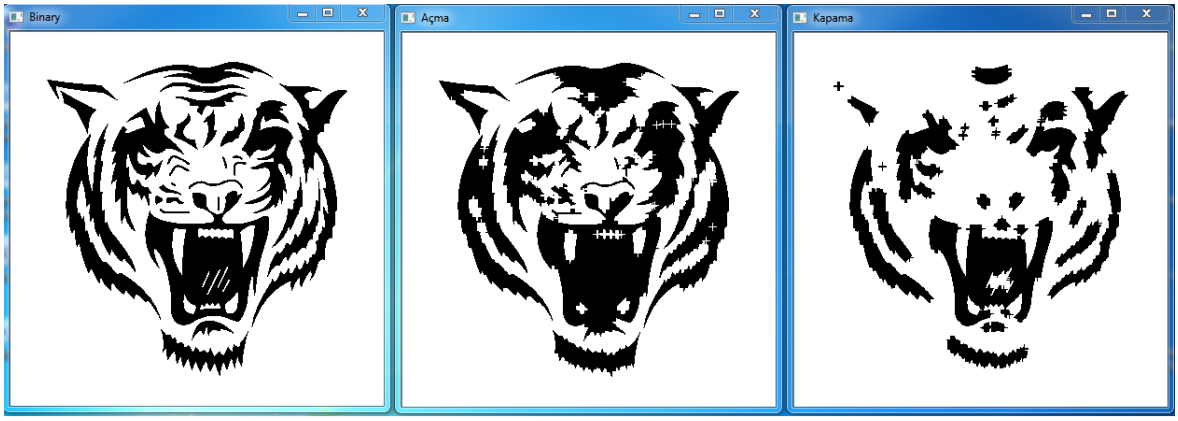
5.3.7.2 Açma (Opening) ve Kapama (Closing)

Bir görüntüye önce aşınma daha sonra genişleme uygulanırsa bu görüntüye açma (opening) işlemi uygulanmış olunur. Açma işlemi görüntüde ince çıkıntıları, dışa doğru olan sivri sınır düzensizliklerini, ince birleşimleri ve izole olmuş küçük nesneleri yok etmektedir. Açma işlemi ile birbirine yakın iki nesne görüntüde fazla değişime sebebiyet vermeden ayrılabilmektedir.

Bir görüntüye önce genişleme daha sonra aşınma uygulanırsa bu görüntüye kapama (closing) işlemi uygulanmış olunur. Kapama işlemi görüntüde ince girintileri, içeri doğru olan sivri sınır düzensizliklerini ve küçük boşlukları yok etmektedir. Kapama işlemi ile birbirine yakın iki nesne görüntüde fazla değişime sebebiyet vermeden birbirine bağlanabilmektedir.

OpenCV' de açma ve kapama işlemlerini yapabilmek için *MorphologyEx* fonksiyonu bulunmaktadır. Bu fonksiyon tip olarak bir parametre almaktadır. Parametre olarak açma işlemi için *MorphologyOperation.Open* kapama işlemi için ise *MorphologyOperation.Close* parametreleri kullanılmaktadır.

Ek XIII' de verilen kodlarla resim matris formatından alınıp gri tona çevrildikten sonra öncelikle ikili seviyeye indirgenmiştir. Daha sonra ikili seviyedeki resme açma ve kapama işlemleri sırasıyla uygulanmış ve sonuç Şekil 5.24' de gösterilmiştir.



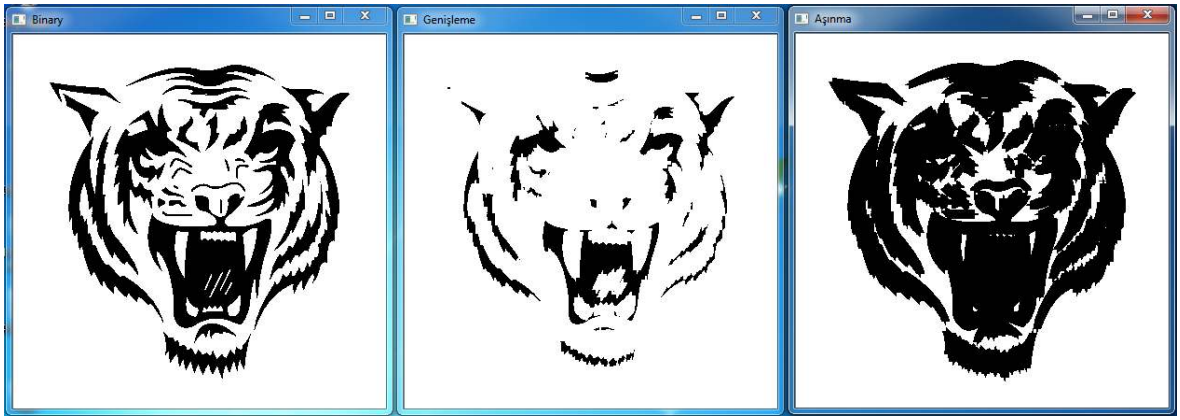
(a) (b) (c)

Şekil 5.24. Morfolojik işlemler (a) ikili binary resim, (b) açma işlemi uygulanmış resim, (c) kapama işlemi uygulanmış resim

5.3.7.3 Genişleme (Dilation) ve Aşınma (Erosion)

Genişleme ve aşınma işlemi için de tıpkı açma ve kapama işlemlerinde olduğu gibi bir tane öncelikle kernel belirlenmektedir. Genişleme ve aşınma işlemlerinin ikisinde de bu kernel tüm pikseller boyunca dolaştırılır. Genişleme de dışarı taşmalar eklenirken, aşınmada sadece resmin içinde kalanlar alınmaktadır. Bir başka deyişle genişleme işleminde resme yeni pikseller eklenirken, aşınma işleminde resimden bazı pikseller silinmektedir.

Ek – XIV'de verilen kodlarla resim matris formatından alınıp gri tona çevrildikten sonra öncelikle ikili seviyeye indirgenmiştir. Daha sonra ikili seviyedeki resme genişleme ve aşınma işlemleri sırasıyla uygulanmış ve sonuç Şekil 5.25' de gösterilmiştir.



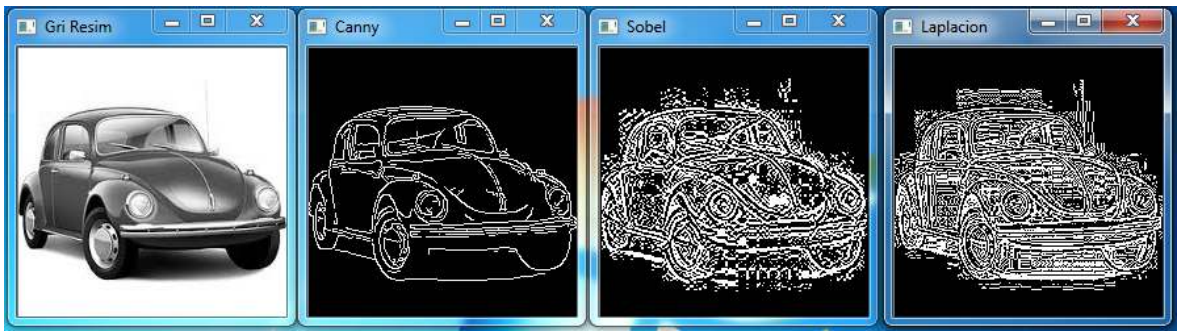
(a) (b) (c)

Şekil 5.25. Morfolojik işlemler (a) ikili binary resim, (b) genişleme işlemi uygulanmış resim, (c) aşınma işlemi uygulanmış resim

5.3.8. OpenCV' de Kenar Bulma

Bir resimde kenarlar sınırları karakterize etmektedir. Bir görüntüdeki kenarlar yoğunluk zıtlığı olan bölgelerdir ve bir yoğunluktan diğerine sıçrama noktasıdır. Bir görüntünün kenarlarını bulma görüntüdeki önemli yapısal özellikleri korurken verinin büyük bir kısmını önemli ölçüde azaltır ve gereksiz bilgiyi filtreler. En sık kullanılan kenar bulma algoritmaları Canny, Sobel ve Laplace olarak bilinmektedir.

Ek XV' de verilen kodlarla resim matris formatından alınıp gri tona indirgenmiştir. Daha sonra gri seviyedeki resme Canny, Sobel ve Laplace kenar belirleme algoritmaları sırasıyla uygulanmış ve sonuç Şekil 5.26' de gösterilmiştir.



(a) (b) (c) (d)

Şekil 5.26. Kenar belirleme (a) gri resim, (b) canny algoritması uygulanmış resim, (c) sobel algoritması uygulanmış resim (d) laplace algoritması uygulanmış resim

5.3.9. OpenCV' de Şekil Bulma (Contour Detection)

Kontürler (contours), basitçe aynı renk ve şiddete sahip bir sınır boyunca sürekli noktaları birleştiren bir eğri olarak tanımlanmaktadırlar. Kontürler şekil analizi, nesne algılama ve tanıma için kullanılabilmektedirler. Kontür işlemi uygulanırken daha iyi sonuçlar elde edebilmek ve eldeki veriyi kaybetmemek için dikkat edilmesi gereken bazı noktalar vardır.

Bunlar;

- Daha doğru bir sonuç elde edebilmek için ikili görüntü kullanılmalıdır.
- Kontür bulma işlemi kaynak görüntüyü değiştirmektedir. Kontür bulma işlemi gerçekleştirildikten sonra kaynak görüntü kullanılmak isteniyorsa kontür bulma işlemi gerçekleştirilmeden önce başka bir değişkene kaynak görüntünün yedeği alınmalıdır.
- OpenCV kontürleri bulurken resmin arka planını siyah nesnenin ise beyaz olması daha iyi sonuç vermektedir.

Kontür bulma işlemi tamamlandıktan sonra bulunan kontürler çizilerek istenilen şekil veya nesne ortaya çıkarılabilmektedir.

Bir resimden bir nesnenin görüntüsünü kontür yöntemiyle bulmak için aşağıdaki adımlar sırasıyla takip edilebilir;

1. Önce renkli görüntü gri tonlu görüntüye çevrilmelidir.
2. Kenar bulma algoritmalarıyla gri tonlu görüntünün kenarları tespit edilmelidir.
3. Görüntünün şekil hatları (contorus) bulunur.
4. Son olarak kontür çizme işlemi ile bulunan kontürler istenen renkte (kırmızı, mavi… vb.) gösterilmektedir.

OpenCV' de kontür bulma işlemini gerçekleştirmek için ***cv2.FindContours()*** fonksiyonu kullanılmaktadır. Bulunan kontürleri çizmek için ise ***cv2.DrawContours()*** fonksiyonu kullanılmaktadır.

5.3.9.1 Kontür Bulma (Cv2.FindContours())

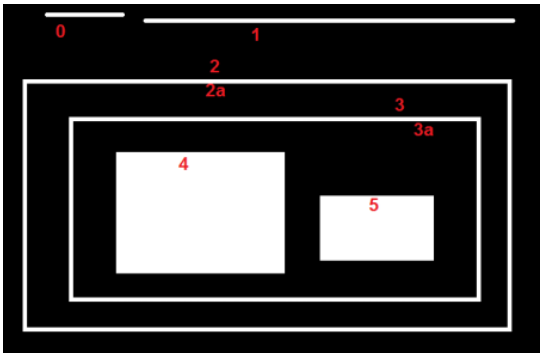
OpenCV' de kontürleri yani nesnenin hatlarını bulmak için *cv2.FindContours()* fonksiyonu kullanılmaktadır. OpenCv bu fonksiyon ile ikili bir görüntüdeki hatları kolaylıkla tespit edebilmektedir.

Bu fonksiyonun C#' da kullanımı aşağıdaki gibidir.

C# : void **Cv2.FindContours**(InputOutputArray **image**, out Mat[] **contours**, OutputArray **hierarchy**, ConturRetriveal **mode**, ContourChain **method**, [OpenCvSharp, CPlusPlus.Point? **offset** = null])

Bu kodla birlikte kontürleri daha düzgün ve istenilen düzeyde bulabilmek için özellikle bazı parametrelerin değerlerinin dikkatli bir şekilde ayarlanması gerekmektedir. Bu fonksiyonla birlikte kullanılacak parametreler;

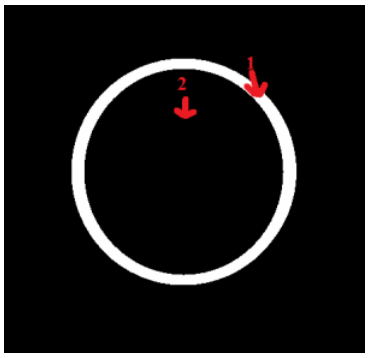
- **image:** Bu parametre öncelikle kaynak görüntüyü temsil edip işlem bittikten sonra ise hedef yani sonuç görüntüsü olarak kullanılmaktadır. Kaynak görüntü ikili (8 – bitlik) görüntü olmalıdır. Renkli bir görüntüyü ikili görüntüye dönüştürmek için öncelikle gri tonlu görüntüye çevirmek gerekir. Daha sonra gri tonlu görüntü *compare(), inRange(), threshold(), adaptiveThreshold()* gibi fonksiyonlar kullanılarak ikili görüntü elde edilebilmektedir.
- **contours:** Tespit edilen kontür noktalarının vektör olarak depolandığı yerdir. Yani her bağlı alan için o alanı temsil edecek bir değişken olarak tanımlanmaktadır. Her bir contours[i] alanı temsil eden noktaları içermektedir [82].
- **hierarchy:** Temsil edilen piksel guruplarının (contours) birleri arasındaki yapılanmayı ve bağı göstermektedir. Bazen nesneler farklı yerlerde bulunabilmekte ve şekiller içiçe yer alabilmektedir. Bu tür durumlarda dıştaki kontür ebeveyn içteki kontür ise çocuk olarak adlandırılmaktadır.



Şekil 5.27. Kontür hiyerarşi yapısı

Şekil 5.27' de görüldüğü gibi 0, 1 ve 2 en dıştaki kontürlerdir. Bunlar aynı düzeyde kontürler olup ebeveyn kontür denilmekte ve hiyerarşi – 0 olarak adlandırılmaktadırlar. Daha sonra gelen kontür 2a kontür 2' nin çocuğudur. Bu yüzden onada hiyerarşi – 1 denilmektedir. Benzer şekilde kontür 3 kontür 2a' nın çocuğu, kontür 4 ve 5 ise kontür 3a' nın çocukları olarak adlandırılmaktadır [83].

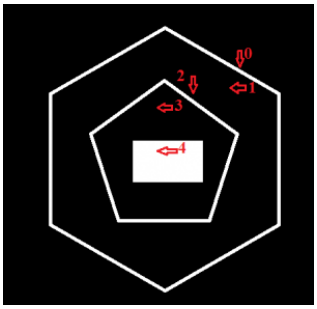
- **mode:** Kontür alma modunu belirlemek için kullanılan parametredir. Alabileceği değerler;
    1. **EXTRERNAL:** Bu mod kullanılırsa sadece dıştaki kontürler alınır. Bütün çocuk kontürler gözardı edilir. Yani bir nevi ailenin en yaşlı üyesi seçilir. Şekil 5.27' de 0, 1 ve 2 numaralı kontürler ailenin en yaşlı üyeleridir, bu mod kullanılırsa sadece 0, 1 ve 2 seçilir.
    2. **LIST:** Bu mod kullanılırsa herhangi bir hiyerarşik ilişki kurmadan tüm kontürleri alır. Ebeveynler ve çocuklar bu mode altında eşit düzeydedirler.
    3. **CCOMP:** Bu mod kullanılırsa tüm yapılar 2 seviyeye ayrılmış olarak kabul edilmektedir. Dış yapılar hiyerarşi 1 düzeyinde iç kısımda kalan piksel toplulukları ise 2 düzeyinde olarak kabul edilmektedir.



Şekil 5.28. CV_RETR_CCOMP mod yapısı

Şekil 5.28' deki gibi siyah fonda büyük bir beyaz çember ve bununda içerisinde siyah bir piksel topluluğu düşünülürse *CCOMP* paramtresi kullanılarak yapılan etiketlemede sıfırın dış kısmında kalan beyaz piksel yapısına 1. seviye, diğer iç kısımda kalan çember ise 2. seviye olarak etiketlenmektedir[84].

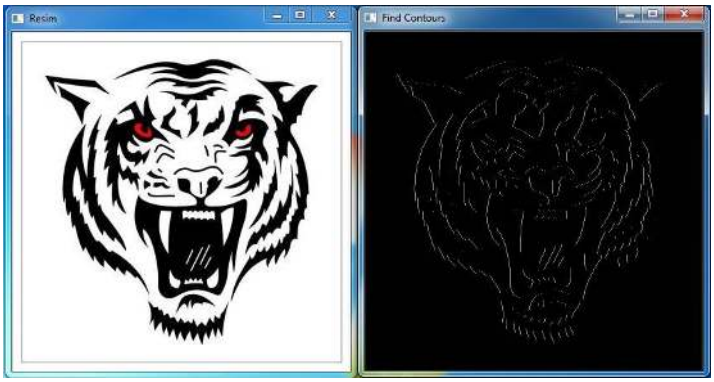
4. **TREE:** Bu mod tüm yapıları ve yapıların kendi arasındaki ilişkilerini söndürmektedir. Yani tam bir aile hiyerarşi listesi oluşturmaktadır.



Şekil 5.29. CV_RETR_TREE mod yapısı

- **method:** Kontür yaklaşım yöntemini belirleyen parametredir. Alabileceği değerler;
    1. **APROXNONE:** Bu method herhangi iki noktanın yatayda ve dikeyde diyagonal komşularına sahip olmak ister.
    2. **APROXSIMPLE:** Bu method yatay, dikey ve diyagonal segmentleri sıkıştırır ve sadece kendi bitiş noktalarını bırakır.
    3. **APPROXTC89L1, APPROXTC89KCOSS:** Bu method uygulama için zincir yaklaşım algoritmasına başvurmaktadır.
- **offset:** Bu parametre ile isteğe bağlı olarak her kontür noktası kaydırılır.

Ek XVI' da verilen kodlarla resim matris formatından alınıp gri tona indirgenmiştir. Daha sonra gri seviyedeki resme ikili resim haline dönüştürülmüş ve *Cv2.FindContours()* fonksiyonu kullanılarak kontürler bulunmuş sonuç Şekil 5.30' da gösterilmiştir.



(a) (b)

Şekil 5.30. Kontür bulma (a) orjinal resim, (b) FindContours fonksiyonuyla kontürleri bulunmuş resim

5.3.9.2 Kontür Alanı (Cv2.ContourArea())

Kontür alanlarını hesaplamak için *Cv2.ContourArea()* fonksiyonu kullanılmaktadır. Bu fonksiyon C#' da aşağıdaki gibi kullanılmaktadır.

C#: double **Cv2.ContourArea**(InputArray **contour**, [bool **oriented** = false])

Bu fonksiyon ile birlikte kullanılacak parametreler;

- **contour:** Girdi vektörü (kontörlerin saklandığı vektör).
- **oriented:** Kontürler birbirine bağlı ise doğru (true) değerini gönderir, bağlı değil ise varsayılan değer olarak da kullanılan yanlış (false) değerini gönderir.

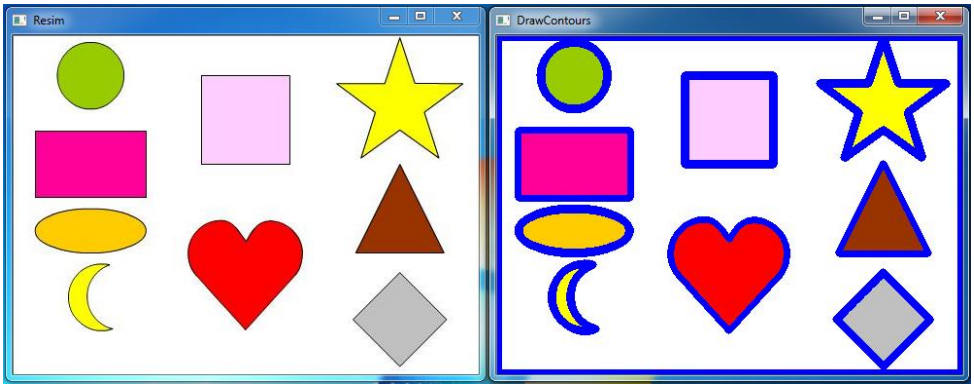
5.3.9.3 Kontür Çizme (Cv2.DrawContour())

Bulunan kontürleri çizmek için *Cv2.DrawConturs()* fonksiyonu kullanılmaktadır. Bu fonksiyon C#' da aşağıdaki gibi kullanılmaktadır.

C#: void **Cv2.DrawContours** (InputArray **image**, IEnumerable<Mat> **contours**, int **contourIdx**, Scalar **color**, [int **thickness** = 1], [LineType lineType = **LineType.**Link8], [Mat **hierarchy** = null], [int **maxLevel** = 2147483647], [OpenCvSharp.CPlusPlus?**offset** = null])

Bu fonksiyon ile birlikte kullanılacak parametreler;

- **image:** Bu parametre hedef görüntü anlamına gelmektedir. Çizilecek kontörlerin üzerinde bulunduğu kaynak görüntüdür.
- **contours:** Giriş kontörleridir. Her kontörün saklandığı vektördür.
- **contourIdx:** Bu parametre ile çizmek için kontör gösterilmektedir. Eğer negatif ise tüm kontörler çizilir.
- **color:** Çizilecek kontörlerin rengi için kullanılır.
- **thickness:** Çizilen kontörün kalınlığını belirlemek için kullanılır.
- **lineType:** Hat bağlantısı.
- **hierarchy:** Hiyerarşi hakkında isteğe bağlı bilgiler verilmektedir.
- **maxLevel:** Çizilmiş kontür için maksimum seviyeyi belirlemek için kullanılmaktadır.
- **offset:** Opsiyonel olarak kontür değiştirmek için kullanılmaktadır.



(a) (b)

Şekil 5.31. Kontür çizme (a) orjinal resim, (b) DrawContours fonksiyonuyla kontürleri çizilmiş resim

Ek XVII' de verilen kodlarla resim matris formatından alınıp gri tona indirgenmiştir. Daha sonra gri seviye indirgenmiş resim ikili resim haline dönüştürülmüş ve *Cv2.FindContours()* fonksiyonu kullanılarak kontürler bulunmuştur. Kontürleri bulunan resim *Cv2.DrawContours()* fonksiyonu kullanılarak bulunan kontürler mavi renkte orijinal resim üzerine çizilmiş ve sonuç Şekil 5.31' de gösterilmiştir.

6. RETİNA GÖRÜNTÜSÜNDEN DAMAR ÇIKARMA UYGULAMASI

Bu bölüm tez çalışması kapsamında retina görüntülerinden damarların çıkarılması için kullanılan şekil bulma uygulamasının ayrıntılarını içermektedir. Retina görüntülerinden damarların çıkarılması için daha önce yapılmış çalışmalarda Kirsch, eşleştirme filtresi gibi bazı yöntemler kullanılmış olup bu yöntemlerin kodları Matlab ortamında gerçekleştirilmiştir.

Uygulamalarda hız önemlidir, bunun için makine diline en yakın üst düzey programlama dilleri tercih edilmelidir [33]. Bu nedenle bu tez çalışmasında kullanılan yazılımlar geliştirilirken Intel şirketinin görsel bilgisayar uygulamaları için geliştirdiği OpenCV Kütüphanesi kullanılmıştır.

C ve C++ arayüzüne sahip bir kütüphane olan OpenCV' nin Matlab'a göre en büyük avantajı açık kaynak kodlu ve tamamen ücretsiz bir kütüphane olmasıdır. Görüntü hızı konusunda OpenCV Matlab'a göre çok daha hızlıdır. Bunun nedeni bilgisayarımızda bir Matlab programı çalıştırıldığında bilgisayar bütün Matlab kodlarını yorumlamaktadır. OpenCV ise kısa zamanda kodu bilgisayara gönderir. Çok sayıda algoritmaya sahip olduğu için ekstra yorumlanmaya ihtiyacı yoktur. Bu yüzden Matlab'da yazılmış aynı programa göre çok daha hızlı çalışır. Anlık görüntü alma hızı çok daha fazladır. Eğer insanların gülümsemesini yakalayacak bir video görüntüsü için küçük bir program yazılacak olursa Matlab' da saniye de *3-4* fps yakalayabilirken, OpenCV' de *25-30* fps yakalanabilmektedir. Yine hızla yakından alakalı olarak, Matlab' da yazılan kodla görüntü işleme uygulaması gerçekleştirirken bilgisayar OpenCV 'ye oranla daha fazla bellek kullanımına ihtiyaç duymaktadır. OpenCV'nin bir diğer avantajı ise farklı platformlarda çalışabilmesi ve bu platformlarda gerçekleştirilecek form tasarımları sayesinde kullanıcıların kod bilmeden programı kullanabilmelerini sağlamaktadır ve tasarlanan formlar sayesinde program kullanımı oldukça basite indirgenebilmektedir.

Tez çalışması kapsamında öncelikle Kirsch operatörü yardımıyla Matlab ortamında gerçekleştirilen damar çıkarma uygulaması OpenCV Kütüphanesi kullanılarak Visual Studio 2013 C# ortamına aktarılmıştır. Daha sonra bu tez çalışması için geliştirilen uygulamada ise kontür bulma, kontür alanı belirleme ve kontör çizme fonksiyonları kullanılarak retina görüntüsündeki damarların belirlenmesi daha etkin bir şekilde gerçekleştirilmiştir.

6.1. Kirsch Operatörü Kullanılarak Retina Görüntüsündeki Damarların Belirlenmesi

Yapılan çalışma; OpenCV görüntü işleme kütüphanesi kullanılarak Visual Studio 2013 ortamında C# programlama dilinde gerçekleştirilmiştir. Çalışma gerçekleştirilirken Intel ® Core™ i5 M 450 @ 2.40 GHz işlemcili, Windows 7 Home Premium 64 bit işletim sistemine sahip, 8 GB RAM bellek, 1GB harici ekran kartlı kişisel bilgisayar kullanılmıştır.

Matlab ortamında Kirsch operatörü kullanılarak gerçekleştirilen uygulama bu tez çalışması için C# programlama diliyle OpenCV kütüphanesi kullanılarak yazılmıştır.

Bu uygulamada öncelikle fundus kamerayla çekilmiş olan retina görüntüsü matris formatında alınıp bir değişkene aktarılmaktadır. Daha sonra okunan bu retina görüntüsü renkli durumdan gri tonlamalı duruma indirgenmiştir.

(a) (b)

Şekil 6.1. Görüntünün matris formatında alınması ve gri seviyeye indirgenmesi (a) orjinal resim, (b) gri resim

Gri görüntü üzerinde kirsch operatörü uygulanmıştır. Kullanılan kirsch kernel matrisleri aşağıda gösterilmektedir.

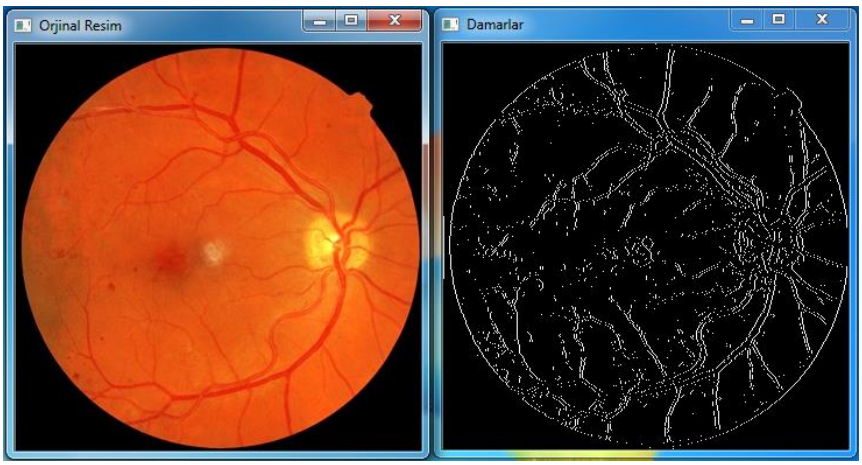
$$h1 = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \qquad h2 = \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

$$h3 = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \qquad h4 = \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix}$$

$$h5 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \qquad h6 = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

$$h7 = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \qquad h8 = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$$

Sekiz adet kernel matrisinin görüntüye uygulanmasının ardından elde edilen sekiz görüntünün pikselleri üzerinde gezinip belirtilen eşik değerinden büyük olan piksel değerleri oluşturulmuş olan görüntü matrisine aktarılıp son olarak da elde edilen görüntü siyah / beyaz yani ikili görüntü olarak görüntülenmektedir.

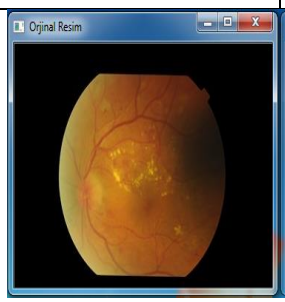
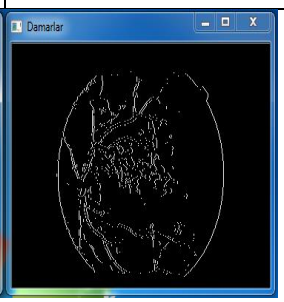
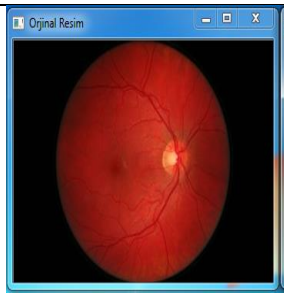
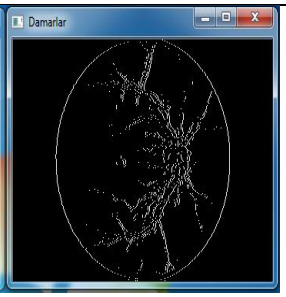
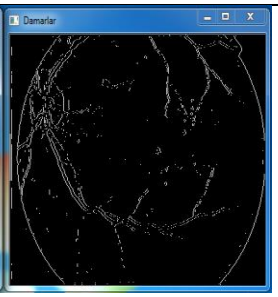
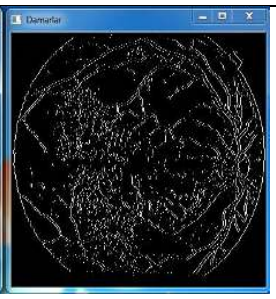


(a) (b)

Şekil 6.2. Retina görüntüsünden Kirsch operatörü kullanılarak damarların çıkarılması (a) orjinal resim, (b) damarları çıkarılmış resim

Tablo 6.1’ de Farklı retina görüntülerinin damarları tespit edilerek gösterilmiştir.

Tablo 6.1. Farklı retina görüntülerinden Kirsch operatörü kullanılarak damarların çıkarılması

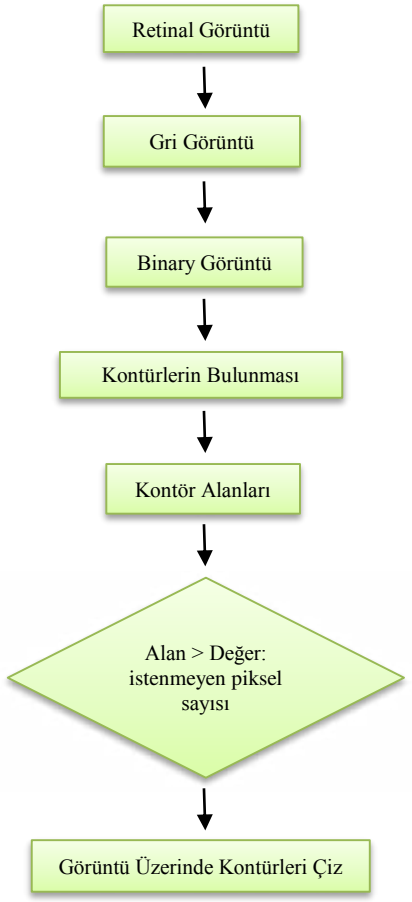
Orijinal Resim	Damarları tespit edilmiş resim
	
	
	
	

Uygulama farklı retina görüntülerine uygulandığında istenilen sonuçların tam olarak alınamadığı görülmektedir. Bu uygulama ile damar olmayan birçok bölgenin de damar gibi görüntülendiği anlaşılmaktadır. Bu nedenle tez çalışması için bu alanda yeni kullanılan bir yöntem olan şekil hatlarını belirleme uygulaması gerçekleştirilmiştir.

6.2. Şekil Hatlarının Belirlenmesi Yöntemi İle Uygulamanın Geliştirilmesi

Çalışması kapsamında geliştirilen uygulamada retina görüntüsündeki damarların belirlenmesi için kontürlerin belirlenmesi, istenilen piksel sayısındaki kontür alanlarının tespit edilip, tespit edilen kontürlerin çizilmesi işlemleri uygulanmıştır. Yapılan çalışma; OpenCV görüntü işleme kütüphanesi kullanılarak gerçekleştirilmiştir.

Retina görüntüsündeki damarların belirlenmesi için bu uygulamada Şekil 6.3' deki yol takip edilmiştir.

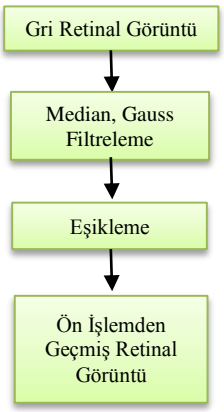


Şekil 6.3. Retina görüntülerinden kan damarlarının belirlenmesi için izlenecek yol

Retinadaki kan damarlarının tespiti için fundus kamerayla çekilmiş retina görüntüsünün gri seviyeye indirgenmesiyle başlamaktadır. Gri seviyeye indirgenmiş retinal fundus görüntülerde yer alan kan damarlarını arkaplana göre daha da belirginleştirmek ve kümeleme işleminde ayırt edilmesini kolaylaştırmak için görüntünün bazı aşamalardan geçmesi gerekmektedir[7]. Bu aşamalar;

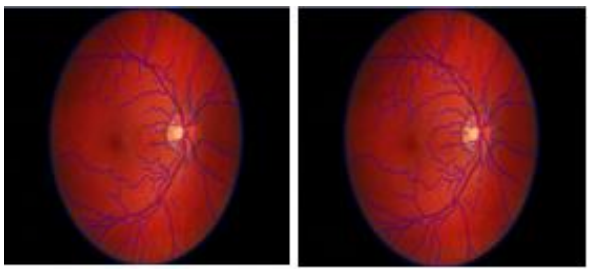
1. Filtreleme işlemi
2. Eşikleme işlemi

olarak kullanılmaktadır. Bu uygulamada filtreleme işlemi için median ve gauss filtreleri, eşikleme işlemi için ise adaptif eşikleme yöntemleri kullanılmıştır. Retina görüntüsüne ön işlem uygulama aşamaları Şekil 6.4'de gösterilmektedir.



Şekil 6.4. Retina görüntülerine uygulanan ön işlem basamakları

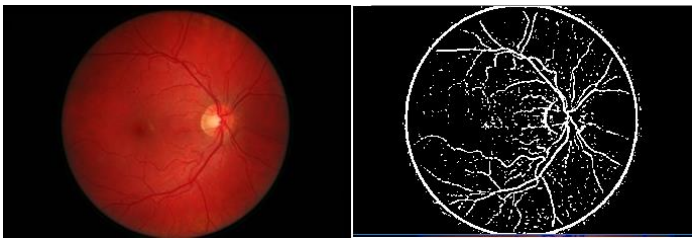
Median filtre genellikle salt & pepper denilen bozulma türünü gidermek için kullanılmaktadır. Median filtrenin Gaussian dağılımını kullanarak biraz daha değiştirilmiş hali Gaussian filtresi olarak bilinir. Retina görüntüsüne median ve gaus filtre uygulanmış sonuçları Şekil 6.5'de gösterilmiştir.



(a) (b)

Şekil 6.5. (a) Median filtre uygulanarak damarları tespit edilen retina görüntüsü, (b) gauss filtre uygulanarak damarları tespit edilen retina görüntüsü

Eşikleme genellikle resmi binary resme çevirmek için kullanılan bir tekniktir. Gürültüyü önlemek veya nesne belirlemek gibi farklı amaçlarla da kullanılabilmektedir. Genel olarak belirlenen bir sınır değeriyle resmin tüm piksellerinin teker teker karşılaştırılması ve belirli bir kurala göre resmi *0 (sıfır)* ve *1 (bir)*'e dönüştürme uygulamasıdır. Retina görüntüsüne eşikleme işlemi uygulanmış sonuçları Şekil 6.6'da gösterilmiştir.



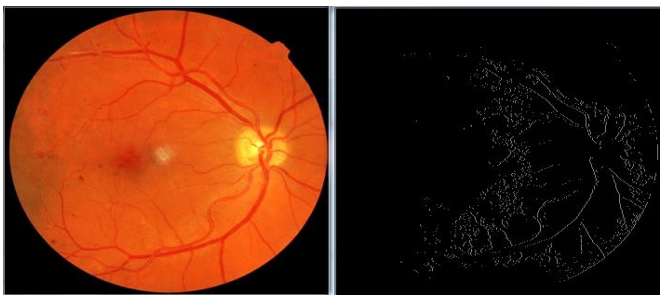
(a) (b)

Şekil 6.6. (a) Orijinal retina görüntüsü, (b) eşikleme işlemi uygulanmış retina görüntüsü

Görüntü üzerinde gerekli ön işlemler yapıldıktan sonra şekil hatlarının belirlenmesi yöntemiyle retina görüntüsündeki damarların tespiti yapılabilmektedir.

Şekil hatlarının belirlenmesi yöntemiyle damarların tespit edilebilmesi üç aşamada gerçekleşmektedir.

İlk aşama olarak görüntü üzerindeki kontürlerin bulunması gerekmektedir. Kontürler, basitçe aynı renk ve şiddete sahip bir sınır boyunca sürekli noktaları birleştiren bir eğri olarak tanımlanmaktadırlar. Şekil 6.7'de bir retina görüntüsü üzerinde kontür bulma işlemi gerçekleştirilmiştir.



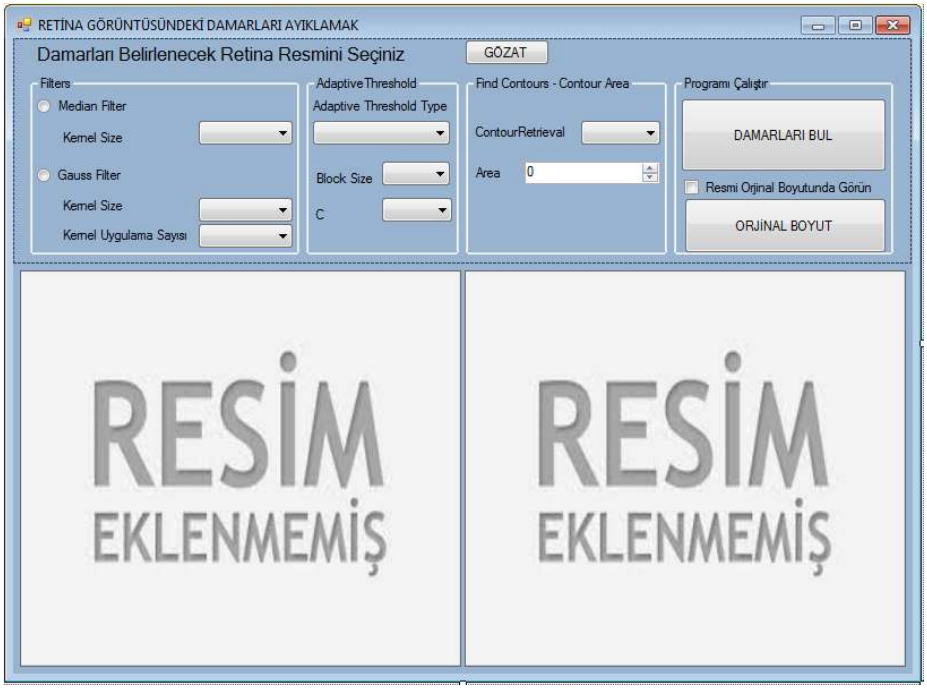
(a) (b)

Şekil 6.7. (a) Orjinal retina görüntüsü, (b) kontürleri bulunmuş retina görüntüsü

İkinci aşama olarak da bulunan kontürlerin alanı hesaplanmaktadır. Kontür alanlarını hesaplamadaki amaç kontür alanları belirlenirken bir sınır boyunca damar olmayan küçük parçalarda bulunmaktadır, bu nedenle küçük olan ve damar olmayan bu alanların istenmeyen bölge olarak belirlemek gerekmektedir. İstenmeyen alanlar belirlenirken girilen piksel sayısından daha küçük bir alana sahip kontürler üçüncü aşama ile işleme tabi tutulmamaktadırlar.

Üçüncü ve son aşamada ise kontürler bulunup istenmeyen alanlar yani belirli bir piksel sayısı altında kalan alanlar devre dışı bırakıldıktan sonra bu aşamada görüntü üzerinde tespit edilen kontürlerin üzeri çizilmekte ve belirgin bir hale getirilmektedir. Böylelikle orijinal retinal görüntüsü üzerindeki kan damarları belirgin bir hale getirilmiş olmaktadır.

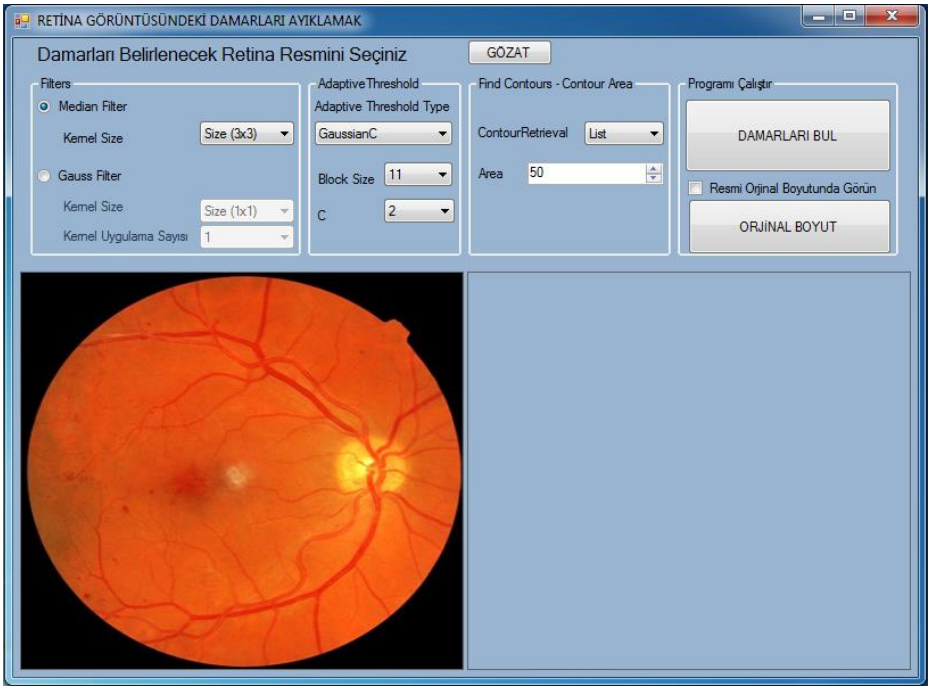
Uygulama için bir Visual Studio 2013 C# Windows Form Application ortamında Şekil 6.8' deki gibi bir form tasarlanmıştır.



Şekil 6.8. Damar çıkarma uygulaması için tasarlanan form

Damarların bulunması için gerekli bilgisayarda kayıtlı olan fundus kamerayla çekilmiş retina görüntüsünü programa yüklemek gerkir. Görüntüyü yüklemek için formda bulunan "*GÖZAT*" butonuna tıklanarak bilgisayardaki retina görüntüsünün yolu seçilip retina görüntüsü Şekil 6.9' da görüldüğü gibi programa yüklenmiş olur. Yüklenmiş olan retina

görüntüsü bir değişkene aktarılmaktadır. Değişkene aktarılan görüntü matris formatında olup üzerinde işlem yapılmaya hazır durumdadır.



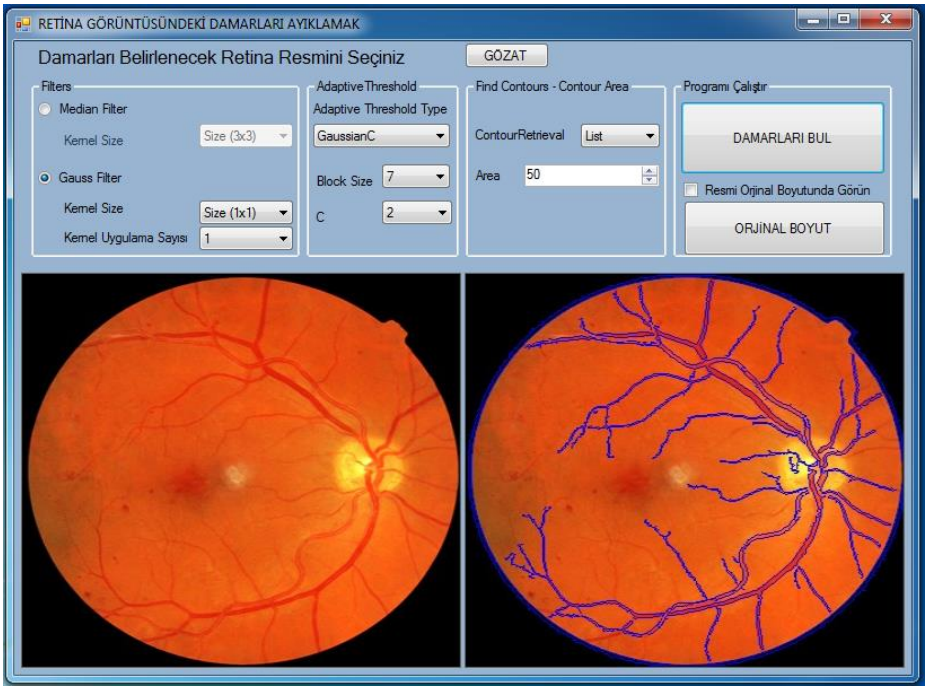
Şekil 6.9. Programa retina görüntüsünün yüklenmesi

Resim yüklendikten sonra damarları tespit etmek için form üzerinde bulunan "*DAMARLARI BUL*" butonuna tıklamak gerekmektedir ancak öncelikle bazı ayarlamaların yapılması gerekmektedir. Eğer damarlar bulunduktan sonra istenen sonuç elde edilmemiş ise form üzerindeki diğer seçeneklerin değerleri değiştirilerek daha iyi sonuçlar elde edilebilmektedir.

Retina görüntüsü üzerindeki damarların daha belirgin bir şekilde çıkarılabilmesi için öncelikle resim gri tona çevrilmektedir ve gri tondaki resme bazı filtreler uygulanmaktadır. Filtreleri uygulamak için form üzerinde bulunan *"Filters"* gurubundaki filtrelerden biri kullanılabilinir. Bu guruptaki filtreler bu uygulama için *"Median Filter"* ve *"Gauss Filter"* olmak üzere iki tanedir. Bölüm 5' de açıklanan bu filtreler için uygun kernel matris boyutları form üzerine yerleştirilmiş *"ComboBox"* kompenentleri aracılığıyla belirlenebilmektedir. Eğer Gauss Filter seçili ise bu filtrenin görüntüye kaç defa uygulanacağı da yine buradaki "*Kernel Uygulama Sayısı*" şeklinde verilen ComboBox aracılığıyla girilebilmektedir. Daha sonra filtreden geçirilmiş resme *"Adaptive Threshod"* uygulanarak resim ikili resme dönüştürülmektedir. Adaptive Threshod seçenekleri de yine form üzerinde

belirlenebilmektedir. Bu bölümdeki özellikler de belirlendikten sonra artık kontürlerin belirlenmesi olayı gerçekleştirilecektir.

Kontürler belirlendikten sonra damar olmayan hatların yani kontürler belirlenirken bulunan damar gibi görünen kontürlerin (piksel guruplarının) ayrıştırılması için kontür alanları belirlenmektedir. Damar olmayan alanlar ayrıştırıldıktan sonra geriye kalan ve damar olan kontürlerin çizilmesi işlemi gerçekleştirilmiştir. Sonuç olarak Şekil 6.10' daki gibi görüntü elde edilmektedir.

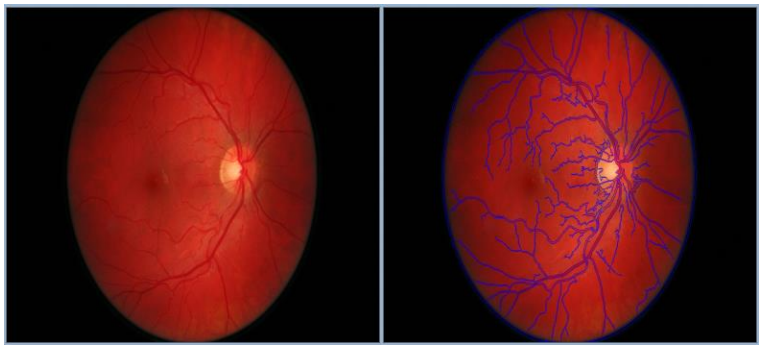


Şekil 6.10. Retina görüntüsünün yüklenmesi ve damarların tespit edilmesi

Eğer damarları tespit edilmiş retina görüntüsü orijinal boyutunda görünmek isteniyorsa form üzerinde bulunan *"Resmi Orijinal Boyutunda Görün"* CheckBox (onay kutucuğu) işaretlenerek "*ORJİNAL BOYUT*" butonuna tıklanmalıdır. Bu butona tıklandığında Şekil 6.11' deki gibi bir görüntü elde edilmektedir.

Şekil 6.11. Retina görüntüsünün orijinal boyutta görüntülenmesi

Uygulamada farklı retina görüntüleri kullanılarak Şekil 6.12' deki gibi görüntüler elde edilmiştir.



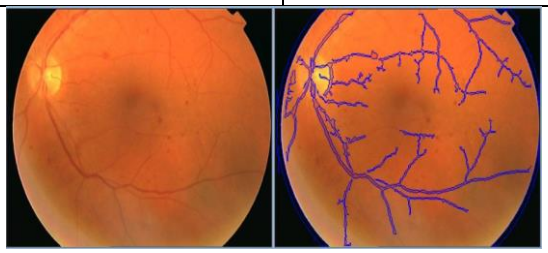
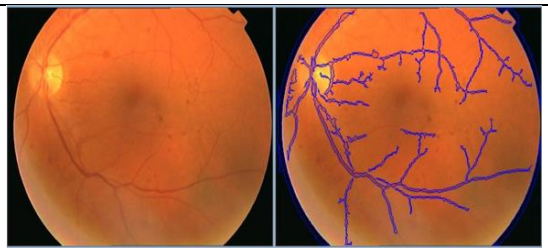
(a) (b)

Şekil 6.12. Retina görüntüsünden damarların tespit edilmesi (a) orijinal görüntü, (b) damarları tespit edilmiş görüntü

Şekil 6.13. Retina görüntüsünün orijinal boyutta görüntülenmesi

Tablo 6.2'de Farklı retina görüntülerinin damarları tespit edilerek gösterilmiştir.

Tablo 6.2. Damarları tespit edilmiş farklı retina görüntüleri

Orijinal resim	Damarları tespit edilmiş resim
	
	
	
	
	

7. SONUÇLAR

İnsan hayatının son yıllarda uzamasının en önemli nedenlerinden bir tanesi hastalıkların tespiti ve erken teşhisi için tıpta bilgisayar bilimleri ve görüntü işleme tekniklerinin kullanılmasıdır. Röntgen ve tomografi başta olmak üzere görüntü işleme tekniklerinin tıpta kullanılmasıyla bir çok hastalığa daha kolay ve erken teşhis konulabilmektedir.

İnsanlarda ortaya çıkan hastalıklardan bazıları retinada bulunan kan damarlarının yapısında bozulmalara (kan damarlarının genişlemesi vb.) neden olmaktadır. Bu nedenle retinada bulunan kan damarlarındaki yapısal bozuklukların önceden teşhisi diyabetik retinopati, glakom, damar sertliği, retinal alter tıkanıklığı gibi bazı hastalıkların tedavisinde önemli rol oynamaktadır.

Retinada bulunan kan damarlarının yapısal bozukluklarının daha kolay teşhis edilebilmesi için retinadaki kan damarlarının retina görüntüsünden ayrıştırılması gerekmektedir. Retina görüntüsünden kan damarlarının ayrıştırılması için bilgisayar bilimleri ve görüntü işleme teknikleri kullanılmaktadır.

Bu yüksek lisans tez çalışmasında fundus kamerayla çekilen retina görüntülerinden kan damarlarının belirlenmesi işlemi gerçekleştirilmiştir. Bu işlem için kullanılan bazı yöntemlerden bahsedilmiştir. Açık kaynak kodlu görüntü işleme kütüphanesi olan OpenCV anlatılmış, OpenCV' nin kurulumu, bazı fonksiyonları anlatılmış, temel uygulamalar yapılmış ve tez çalışmamız kapsamında kullanılan OpenCV kenar bulma ve şekil hatları bulma fonksiyonları anlatılmıştır.

Çalışma kapsamında yapılan uygulamada şekil hatlarını belirleme fonksiyonları, etkin bellek kullanımı için OpenCV kütüphanesi aracı ile sağlanmıştır. C ve C++ arayüzüne sahip bir kütüphane olan OpenCV' nin MATLAB' a göre en büyük avantajı açık kaynak kodlu ve tamamen ücretsiz bir kütüphane olmasıdır. Görüntü hızı konusunda OpenCV Matlab'a göre çok daha hızlıdır. Matlab'da yazılan kodla görüntü işleme uygulaması gerçekleştirirken bilgisayar OpenCV 'ye oranla daha fazla bellek kullanmaktadır.

Uygulama için tasarlanan form sayesinde daha kolay bir kullanım sunulmakta ve bu form sayesinde girdi görüntüsü üzerinde yapılan değişikler çıktı görüntüsü üzerinde anında görüntülenebilmektedir. Bununla birlikte uygulama geliştirilirken OpenCV kullanılması

sayesinde uygulamanın istenilen platformda herhangi bir programa ihtiyaç duyulmadan çalıştırılmasını sağlamaktadır.

Yapılan uygulamada OpenCV kütüphanesinde bulunan filtreleme yöntemleri ve şekil bulma fonksiyonları ile retina görüntüsündeki kan damarlarının sınırları belirlenmiştir.

Bilgisayar sistemlerinin gelişmesiyle birlikte kullanılan bu yöntemler daha da gelişecektir. Böylece daha net ve hızlı sonuçlar elde edilebilecektir.

KAYNAKLAR

[1] Muthukrishnan, R., Radha, M, (2011), "Edge Detection Techniques for Image Segmentation", International Journal of Computer Science & Information Technology (IJCSIT), Vol. 3, No. 6, pp. 259 - 267, DOI: 10.5121/ijcsit.2011.3620.

[2] Bob Z., Lin Z., Lei Z., Fakhri k., "Retinal Vesel Extrection By Matched Filter With First – Order Derivate Of Gaussian", Dept. Of Electrical an Computer Engineering, Universty of Waterloo Aterloo, ON, Canada, N2L3G1, Biometrics Research Center, The Hong Kong Polytechnic Iniversty, Hong Kong, China

[3] Kutlu H., "İnsan Bilgisayar Etkileşimli Görüntü İşleme Uygulamaları", Yüksek Lisans Tezi, T.C. Fırat Üniversiyesi Fen Bilimleri Enstitüsü Elektronik ve Bilgisayar Eğitimi Anabilim Dalı Bilgisayar Sistemleri Eğitimi, Elazığ 2013.

[4] TÜRKOĞLU İ., "T.C. Fırat Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Bilgisayar Eğitimi Anabilim Dalı Örüntü Tanıma Ders Notları", Elazığ.,2010

[5] Ateş E., "Özgür Yazılımlarla Görüntü İşleme", Akademik Bilişim 2013, Antalya

[6] Uğur A., "Görüntü İşlemeye Giriş", 2013.

[7] Yavuz Z., Köse C., "Bulanık C-Kümeleme Algoritması ile Retinal Kan Damarı Bölütleme Retinal Vessel Segmentation with Fuzzy C-Means Clustering Algorithms" Bilgisayar Mühendisliği Bölümü Karadeniz Teknik Üniversitesi, Bursa 2012

[8] Y. A. Tolias and S. M. Panas, "A fuzzy vessel tracking algorithm for retinal images based on fuzzy clustering", IEEE Trans. Med. Imag., vol. 17, pp. 263–273, 1998.

[9] A. Can, H. Shen, J. N. Turner, H. L. Tanenbaum, and B. Roysam, "Rapid automated tracing and feature extraction from retinal fundus images using direct exploratory algorithms", IEEE Trans. Inform. Technol. Biomed., vol. 3, pp. 125–138, 1999.

[10] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum, "Detection of blood vessels in retinal images using two-dimensional matched filters," IEEE Trans. Med. Imaging, pp. 263–269, 1989.

[11] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response", IEEE Trans. Med. Imag., vol. 19, pp. 203–210, 2000.

[12] M. E. Martínez-Pérez, A. D. Hughes, A. V. Stanton, S. A. Thom, A. A. Bharath, and K. H. Parker, “Scale-space analysis for the characterization of retinal blood vessels”, Medical Image Computing and Computer-Assisted Intervention—MICCAI’99, 1999, pp. 90–97.

[13] T. Walter and J. C. Klein, “Segmentation of color fundus images of the human retina: Detection of the optic disc and the vascular tree using morphological techniques”, Medical Data Analysis, pp. 282–287, 2001.

[14] F. Zana and J. C. Klein, “Segmentation of vessel-like patterns using mathematical morphology and curvature evaluation”, IEEE Trans. Image Processing, vol. 10, no. 7, pp. 1010–1019, 2001.

[15] J. Staal, M. D. Abrmoff, M. Niemeijer, M. A. Viergever, and B. V. Ginneken, “Ridge-based vessel segmentation in color images of the retina”, IEEE Trans. Med. Imag., vol. 23, no. 4, pp. 501–509, 2004.

[16] João V. B. Soares, Jorge J. G. Leandro, Roberto M. Cesar Jr., Herbert F. Jelinek, and Michael J. Cree. “Retinal Vessel Segmentation Using the 2-D Gabor Wavelet and Supervised Classification”, IEEE Transactions on Medical Imaging, Vol. 25, No. 9, pp. 1214-1222, 2006.

[17] E. Ricci and R. Perfetti, "Retinal Blood Vessel Segmentation Using Line Operators and Support Vector Classification", IEEE Transactions on Medical Imaging, Vol. 26, No. 10, pp. 1357-1365, 2007.

[18] Marin, D.; Aquino, A.; Gegundez-Arias, M.E.; Bravo, J.M., "A New Supervised Method for Blood Vessel Segmentation in Retinal Images by Using Gray-Level and Moment Invariants-Based Features", IEEE Transactions on Medical Imaging, Volume: 30, Issue: 1, pp. 146-158, 2011.

[19] X. Jiang and D. Mojon, “Adaptive local thresholding by verification-based multithreshold probing with application to vessel detection in retinal images”, IEEE Trans. Pattern Anal. Mach. Intell., vol. 25, no. 1, pp. 131–137, 2003.

[20] K.Jeyasri, P.Subathra and K.Annaram, “Detection of Retinal Blood Vessels for Disease Diagnosis” International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 3, March 2013

[21] D.Siva Sundhara Raja, Dr.S.Vasuki and D.Rajesh Kumar, “Performance Analysıs of Retınal Image Blood Vessel Segmentatıon” Advanced Computing: An International Journal (ACIJ), Vol.5, No.2/3, May 2014

[22] www.fenokulu.net

[23] http://www.uzunhayat.net/duyu-organlari/goz-ve-gorme-duyusu.html

[24] Hüseyin Y. "Hastalık Teşhis ve Takibine Yönelik Bilgisayar Görüş Teknikleriyle Ophtalmoscope Görüntülerinde Görüntü Kayıtlama", Ege Üniversitesi Fen Bilimleri Enstitüsü, 16.09.2008

[25] www.atesveates.com/pdf/diyabetik_retinopati.pdf

[26] http://gozdr.com/?p=114

[27] Your Eyes and Diabetes Turkish, Diabetes UK, Diyabet Bilgi Mart 2009

[28] http://gozsagligi.todnet.org/oct-optical-coherence-tomographynedir-ne-amacla-kullanilir/

[29] Dr. Mine Ö. KURTULMUŞOĞLU, Dr. Şengül C. ÖZDEK, "Güncel Retinal Görüntüleme Yöntemleri", Şubat 2008 GÜTF

[30] Zafer C., Koray A., "Diyabetik Retinopati", İstanbul Üniversitesi, İstanbul Tıp Fakültesi Göz Hastalıkları Anabilim Dalı, İstanbul

[31] Prof. Dr. Murat K., "Diyabetik Retinopati", İ.Ü. Cerrahpaşa Tıp Fakültesi Sürekli Tıp Eğitimi Etkinlikleri, Diabetes Mellitus Sempozyumu, 18-19 Aralık 1997, İstanbul, s. 61-67

[32] http://tr.wikipedia.org/wiki/Piksel

[33] Buğday A., "Gerçek Zamanlı Videolarda Ön Plan Arka Plan Ayrımı", Yüksek Lisans Tezi, 93-94, Ankara 2010.

[34] http://hamithoca.blogspot.com.tr/2010/12/grafik-animasyon-notlar.html

[35] http://helpx.adobe.com/tr/photoshop/using/chanel-basics.html

[36] Yılmaz İ., "Renk Sistemleri, Renk Uzayları ve Dönüşümler", Selçuk Üniversitesi Jeodezi ve Fotogrametri Mühendisliği Öğretiminde 30. Yıl Sempozyumu, Konya 2002.

[37] http://helpx.adobe.com/tr/photoshop/using/color.html

[38] Yrd. Doç. Dr. Harun Hilmi P., "Garfik Tasarım Sürecinde Kullanılan Aygıtların Renk Modelleri"

[39] http://www.grafikhaber.net/bilgi/cmyk-rgb-renk-nedir.html

[40] Deniz T., "Sıkıştırılmış Video Akımının Düzensiz Haritalar ve Başlangıç Kodlarına Dayalı Şifrelenmesi", Trakya Üniversitesi Fen Bilimleri Enstitüsü, Edirne 2007.

[41] Jack K., "Video Demstified", LLH Technology Publishing, 1995.

[42] http://www.cescript.com/2012/07/hsv-rgb-uzay-donusumu.html

[43] Melih Ö., "Dijital Ortamda Renkler"

[44] K. Sinan Y., Cenk., T. Emre K., "Görüntü İşleme", Ege Üniversitesi Bilgisayar Mühendisliği Bölümü

[45] Doç. Dr. Sarp E., Arş. Gör. Oğuzhan URHAN, "İmage İşleme Ders Notları", Kocaeli Üniversitesi Mühendislik Fakültesi Elektronik ve Haberleşme Mühendisliği Bölümü

[46] Mehmet K., "Görüntü İşleme Teknolojileri ve Uygulamaları", Ege Üniversitesi, Uşak 2012.

[47] Daniel L. B., "Mastering OpenCV With Pratical Computer Vision Project"

[48] Atilla K., "Histogram Trasformation In Image Processing And Its Applications", Szeged Üniversitesi.

[49] Yrd. Doç. Dr. Aydın K., "Görüntü Bölütleme", Elektrik-Elektronik Mühendisliği Bölümü, Pamukkale Üniversitesi, Denizli 2008.

[50] Mustafa A., Saime A., "Beyin Emar Görüntülerinin Wtershed Algoritması Yardımıyla Bölütlenmesi", Erciyes Üniversitesi Elektrik-Elktronik Mühendisliği Bölümü, Kayseri.

[51] Wilson, J. A. Noble Adaptive Segmentation of MRI Data D. L.

[52] enes Ç., "Görüntü İşlemeye Dayalı Avuç İçi İzinin Yapay Sinir Ağı İle Tanınması", Elektronik-Bilgisayar Eğitimi Anabilim Dalı Bilgisayar-Kontrol Eğitimi Programı Yüksek Lisans Tezi, Maramara Üniversitesi, İstanbul 2011.

[53] Chang C., Huang C., "Edge Detection Based On Class Ratio", Shenkeng, Taipei, Taiwan 2002.

[54] Gonzales R. C., Woods R. E., "Digital Image Processing", 2nd Edition, ABD, ISBN: 0-13-094650-8, (2001) 1-750.

[55] http://yavuzbugra.wordpress.com/2011/05/01/goruntu-islemede-filtreleme/

[56] Taner D., Adil A., "İnsan Yüzündeki Duygusal İfadenin Tanınması", Bilgisayar Mühendisliği Bölümü, Dokuzeylül Üniversitesi İzmir

[57] Sevdanur G., "Karınca Kolonisi algoritması ile Kenar Tespiti"

[58] Dilek K. S., "İnsan Yapmı Nesnelerin Hava Fotoğrafları ve Uydy Görüntülerinden Belirlenmesi", Ortadoğu Teknik Üniversitesi, Jeodezi ve Coğrafi Bilgi Teknolojileri Bölümü, Şubat 2007.

[59] http://bilgisayarkavramlari.sadievrenseker.com/2008/12/01/ozellik-cikarimi-feature-extraction/

[60] Nevrez İ., "Dönerkanat Tipinde Bir İnsansız Hava Aracıyla Video Tabanlı Üst Düzey İşlevlerin Tasarlanması", Yüksek Lisans Tezi, T.C. TOBB Ekonmi ve Teknoloji Üniversitesi Fen Bilimleri Enstitüsü Elektrik Elektronik Mühendisliği Ana Bilim Dalı, Ankara 2010

[61] Gonzales, R. C., Woods, R. E., Digital Image Processing, Prentice Hall, Upper Saddle River, New Jersey, 2008.

[62] Borchani, M., Stamon, G., Use of Texture Features for Image Classification and Retrieval, In Proc. SPIE Conference on Storage and Retrieval for Image and Video Databases III, 3229, 401-406, 1997.

[63] Bhagavathy, S., Chhabra, K., A Wavelet-based Image Retrieval System", Project Report, University of California, Department of Electrical and Computer Engineering, Santa Barbara.

[64] Tuceryan, M., Moment Based Texture Segmentation, Appeared in Pattern Recognition Letters, 15, 659-668, Temmuz 1994.

[65] Mukundan, R., Ong, S. H., Lee, P.A., Image analysis by Tchebichef Moments, IEEE Trans. on Image Processing, 10, 1357-1364, 2001.

[66] Zhu, H., Shu, H., Xia, T., Luo, L., Coatrieux, J.L., Translation and Scale Invariants of Tchebichef Moments, Pattern Recognition, (9), 2530–2542, Eylül 2007.

[67] Rani, J.S., Devaraj, D., Sukanesh, R., A Novel Feature Extraction Technique for Face Recognition, International Conf. on Computational Intelligence and Multimedia Applications, 2, 428-435, 2007.

[68] Khotanzad, A., Hong, Y. H., Invariant Image Recognition by Zernike Moments, IEEE Trans. on Pattern Analysis and Machine Intelligence, 12, 489-497, 1990.

[69] Prokorp, R. J., and Reeves, A. P., A Survey of Moment-Based Techniques for Unoccluded Object Representation and Recognition, CVGIP: Graphical Models and Image Processing, 54, 438-460, 1992.

[70] Liao, S.X., Image Analysis with Zernike Moment Descriptors, IEEE Canadian Conference on Electrical and Computer Engineering, 2, 700-703, 1997.

[71] Lin, T.W., Chou, Y.F., A Comparative Study of Zernike Moments, Proc. of IEEE/WIC International Conf. on Web Intelligence, 516-519, Halifax, Canada, 2003.

[72] Maaoui, C., Rosenberger, C., Emile, B., Robust Color Object Detection and Recognition, 13th European Signal Processing Conference, 2005.

[73] Teh, C.-H., Chin, R. T., On Image Analysis by the Methods of Moments, IEEE Trans. on Pattern Analysis and Machine Intelligence, 10(4), 496-513, Temmuz 1988.

[74] Chong, C. W., Mukundan, R., Raveendran, P., An Efficient Algorithm for Fast Computation of pseudo-Zernike Moments, International Journal of Pattern Recognition and Artificial Intelligence, 17(6), 1011-1023, 2003.

[75] Erişti E., "Görüntü İşlemede Yeni Bir Soluk, OPENCV", İstanbul Ticaret Ünivesitesi, Bilgisayar Mühendisliği Bölümü, İstanbul.

[76] Deane S., A Comparison of Background Subtraction Techniques COMP6009 Invdividual Reseach Project, 2007.

[77] Aktekin C., OpenCV Temlleri, http://cemalaktekin.blogspot.com/2011/10/2opencv-temelleri.html , 07.06.2012.

[78] Brodski G. And Haehler A., "Learning OpenCV: Computer Vision With the openCV Library", O'Reilly Media, Amerika Birleşik Devletleri, 16-17 (2008).

[79] OpenCV Reference Manuels-HighGUI Reference Manuel.

[80] Kurt B. Nabiyev V. "Dijital Mamografi Görüntülerinin Kontrast Sınırlı Adaptif Histogram Eşitleme ile İyileştirilmesi", Biyoistatistik ve Tıp Bilişimi AD, Karadeniz Teknik Üniversitesi, Trabzon.

[81] http://yusuftas.com/opencv-de-filtreler/

[82] http://www.turkishcoders.com/2013_01_01_archive.html

[83] http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_contours/py_contours_hierarcy/py_contours_hierarchy.html#contours-hiearchy

[84] http://ayciceksamet.com/opencv-parent-child-iliskisi-ve-hole-detection/

EKLER

EK – I: OpenCV' de Resmin Görüntülenmesi

// PROGRAM 1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Cv2.NamedWindow("pencere",WindowMode.AutoSize);
            Cv2.MoveWindow("pencere",100,100);
            IplImage imgee =
Cv.LoadImage("C:/Users/ZDO2012/Desktop/images/oto.jpg");
            Cv.ShowImage("pencere",imgee);
        }
    }
}
```

EK – II: OpenCV ile Resmin Matris Olarak Alınıp Görüntülenmesi

// PROGRAM 2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim",resim);
        }
    }
}
```

EK – III: OpenCV ile Resmin Matris Olarak Alınıp Gri Tona Çevrilmesi

// PROGRAM 3

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim",resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("YCrCb Renk Uzayı", dst1);
        }
    }
}
```

EK – IV: OpenCV ile Resmin Kontrast Değerleri Arttırılıp Azaltılması

// PROGRAM 4

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim",resim);
            Mat dst1 = new Mat();
            Mat dst2 = new Mat();
            dst1 = resim / 2;
            dst2 = resim * 2;
            Cv2.ImShow("Resim/2", dst1);
            Cv2.ImShow("Resim*2", dst2);
        }
    }
}
```

EK – V: OpenCV ile Resme Histogram Eşitleme Uygulaması

// PROGRAM 5

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.EqualizeHist(dst1, dst2);
            Cv2.ImShow("Histogram Eşitleme", dst2);
        }
    }
}
```

EK – VI: OpenCV ile Resme Adaptif Histogram Eşitleme Uygulaması

// PROGRAM 6

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            CLAHE clahe = Cv2.CreateCLAHE();
            clahe.Apply(dst1, dst2);
            Cv2.ImShow("Adaptif Histogram", dst2);
        }
    }
}
```

EK – VII: OpenCV ile Resme Median Filtre Uygulaması

// PROGRAM 7

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.MedianBlur(dst1, dst2, 7);
            Cv2.ImShow("Median Filtre", dst2);
        }
    }
}
```

EK – VIII: OpenCV ile Resme Gauss Filtre Uygulaması

// PROGRAM 8

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO2012/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.GaussianBlur(dst1, dst2, Cv.Size(7, 7), 1);
            Cv2.ImShow("Gauss Filtre", dst2);
        }
    }
}
```

EK – IX: OpenCV ile Resme Eşikleme (Thresholding) Binary Uygulaması

// PROGRAM 9

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.Binary);
            Cv2.ImShow("Binary", dst2);
            Mat dst3 = new Mat();
        }
    }
}
```

EK – X: OpenCV ile Resme Eşikleme (Thresholding) BinaryInv Uygulaması

// PROGRAM 10

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.BinaryInv);
            Cv2.ImShow("ToZero", dst2);
        }
    }
}
```

EK – XI: OpenCV ile Resme Eşikleme (Thresholding) ToZero Uygulaması

// PROGRAM 11

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.ToZero);
            Cv2.ImShow("ToZero", dst2);
        }
    }
}
```

EK – XII: OpenCV ile Resme Eşikleme (Thresholding) ToZeroInv Uygulaması

// PROGRAM 12

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.ToZeroInv);
            Cv2.ImShow("ToZero", dst2);
        }
    }
}
```

EK – XIII: OpenCV' de Açma (Opening) ve Kapama (Closing) İşlemleri Uygulaması

// PROGRAM 13

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/kaplan.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.Binary);
            Cv2.ImShow("Binary", dst2);
            Mat dst3 = new Mat();
            Mat kernel =
Cv2.GetStructuringElement(StructuringElementShape.Cross,
Cv.Size(9, 9));
            Cv2.MorphologyEx(dst2, dst3,
MorphologyOperation.Open,kernel,iterations:1);
```

```
            Cv2.ImShow("Açma", dst3);
            Mat dst4 = new Mat();
            Cv2.MorphologyEx(dst2, dst4,
MorphologyOperation.Close, kernel, iterations: 1);
            Cv2.ImShow("Kapama", dst4);
        }
    }
}
```

EK – XIV: OpenCV' de Genişleme (Dilation) ve Aşınma (Erosion) İşlemleri Uygulaması

// PROGRAM 14

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.Binary);
            Cv2.ImShow("Binary", dst2);
            Mat dst3 = new Mat();
            Mat kernel =
Cv2.GetStructuringElement(StructuringElementShape.Cross,
Cv.Size(9, 9));
            Cv2.Dilate(dst2, dst3, kernel, iterations: 1);
```

```
            Cv2.ImShow("Açma", dst3);
            Mat dst4 = new Mat();
            Cv2.Erode(dst2, dst4, kernel, iterations: 1);
            Cv2.ImShow("Kapama", dst4);

        }
    }
}
```

EK – XV: OpenCV' de Kenar Bulma Algoritmaları

// PROGRAM 15

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;
namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/tosbaga.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst3 = new Mat();
            Cv2.Canny(dst1, dst3, 125, 255);
            Cv2.ImShow("Canny", dst3);
            Mat dst4 = new Mat();
            Cv2.Laplacian(dst1, dst4, MatType.CV_32F, 1, 1);
            Cv2.ImShow("Laplacion", dst4);
            Mat dst5 = new Mat();
            Cv2.Sobel(dst1, dst5, MatType.CV_32F, 1, 1);
            Cv2.ImShow("Sobel", dst5);
        }
    }
}
```

EK – XVI: OpenCV' de Kontür Bulma

// PROGRAM 16

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;
namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/kaplan.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);
            Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.Binary);
            Cv2.ImShow("Binary", dst2);
            OpenCvSharp.CPlusPlus.Point[][] contours;
            OpenCvSharp.CPlusPlus.HiearchyIndex[] hierarchy;
            Cv2.FindContours(dst2, out contours, out
hierarchy,ContourRetrieval.List, ContourChain.ApproxSimple);
            Cv2.ImShow("Find Contours", dst2);
        }
    }
}
```

EK – XVII: OpenCV' de Kontör Çizme

// PROGRAM 17

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;
namespace OpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Mat resim =
Cv2.ImRead("C:/Users/ZDO/Desktop/images/sekiller.jpg");
            Cv2.ImShow("Resim", resim);
            Mat dst1 = new Mat();
            Cv2.CvtColor(resim, dst1, ColorConversion.BgrToGray);
            Cv2.ImShow("Gri Resim", dst1);  Mat dst2 = new Mat();
            Cv2.Threshold(dst1, dst2, 125, 255,
ThresholdType.Binary);
            Cv2.ImShow("Binary", dst2);
            OpenCvSharp.CPlusPlus.Point[][] contours;
            OpenCvSharp.CPlusPlus.HiearchyIndex[] hierarchy;
            Cv2.FindContours(dst2, out contours, out
hierarchy,ContourRetrieval.List, ContourChain.ApproxSimple);
            Cv2.ImShow("Find Contours", dst2);
            Cv2.DrawContours(resim, contours, -1, Scalar.Blue, 5);
            Cv2.ImShow("DrawContours", resim);
        }
    }
}
```

EK – XVIII: OpenCV' de Kirsch Operatörü Kullanılarak Retinadaki Damarların Belirlenmesi

// PROGRAM 18

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Blob;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using OpenCvSharp.Utilities;
using System.Threading;

namespace DenemeOpenCVSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Mat src1 = new Mat();
            src1 =
Cv2.ImRead("C:/Users/ZDO/Desktop/retina_image/goz.jpg");
            Cv2.ImShow("Orjinal Resim",src1);
            Mat dst1 = new Mat();
            Cv2.CvtColor(src1, dst1, ColorConversion.BgrToGray);

            //Cv2.ImShow("Orjinal Resim",src1);
            //Cv2.ImShow("Gri Resim", dst1);

            Mat h1 = new Mat(3, 3, MatType.CV_32F);
            Mat h2 = new Mat(3, 3, MatType.CV_32F);
            Mat h3 = new Mat(3, 3, MatType.CV_32F);
            Mat h4 = new Mat(3, 3, MatType.CV_32F);
            Mat h5 = new Mat(3, 3, MatType.CV_32F);
```

```
Mat h6 = new Mat(3, 3, MatType.CV_32F);
Mat h7 = new Mat(3, 3, MatType.CV_32F);
Mat h8 = new Mat(3, 3, MatType.CV_32F);

h1.Set<float>(1, 1, 0);
h1.Set<float>(0, 0, 5);
h1.Set<float>(0, 1, -3);
h1.Set<float>(0, 2, -3);
h1.Set<float>(1, 0, 5);
h1.Set<float>(1, 2, -3);
h1.Set<float>(2, 0, 5);
h1.Set<float>(2, 1, -3);
h1.Set<float>(2, 2, -3);

h2.Set<float>(1, 1, 0);
h2.Set<float>(0, 0, -3);
h2.Set<float>(0, 1, -3);
h2.Set<float>(0, 2, -3);
h2.Set<float>(1, 0, 5);
h2.Set<float>(1, 2, -3);
h2.Set<float>(2, 0, 5);
h2.Set<float>(2, 1, 5);
h2.Set<float>(2, 2, -3);

h3.Set<float>(1, 1, 0);
h3.Set<float>(0, 0, -3);
h3.Set<float>(0, 1, -3);
h3.Set<float>(0, 2, -3);
h3.Set<float>(1, 0, -3);
h3.Set<float>(1, 2, -3);
h3.Set<float>(2, 0, 5);
h3.Set<float>(2, 1, 5);
h3.Set<float>(2, 2, 5);

h4.Set<float>(1, 1, 0);
h4.Set<float>(0, 0, -3);
h4.Set<float>(0, 1, -3);
h4.Set<float>(0, 2, -3);
h4.Set<float>(1, 0, -3);
h4.Set<float>(1, 2, 5);
h4.Set<float>(2, 0, -3);
h4.Set<float>(2, 1, 5);
h4.Set<float>(2, 2, 5);

h5.Set<float>(1, 1, 0);
h5.Set<float>(0, 0, -3);
h5.Set<float>(0, 1, -3);
h5.Set<float>(0, 2, 5);
```

```
h5.Set<float>(1, 0, -3);
h5.Set<float>(1, 2, 5);
h5.Set<float>(2, 0, -3);
h5.Set<float>(2, 1, -3);
h5.Set<float>(2, 2, 5);

h6.Set<float>(1, 1, 0);
h6.Set<float>(0, 0, -3);
h6.Set<float>(0, 1, 5);
h6.Set<float>(0, 2, 5);
h6.Set<float>(1, 0, -3);
h6.Set<float>(1, 2, 5);
h6.Set<float>(2, 0, -3);
h6.Set<float>(2, 1, -3);
h6.Set<float>(2, 2, -3);

h7.Set<float>(1, 1, 0);
h7.Set<float>(0, 0, 5);
h7.Set<float>(0, 1, 5);
h7.Set<float>(0, 2, 5);
h7.Set<float>(1, 0, -3);
h7.Set<float>(1, 2, -3);
h7.Set<float>(2, 0, -3);
h7.Set<float>(2, 1, -3);
h7.Set<float>(2, 2, -3);

h8.Set<float>(1, 1, 0);
h8.Set<float>(0, 0, 5);
h8.Set<float>(0, 1, 5);
h8.Set<float>(0, 2, -3);
h8.Set<float>(1, 0, 5);
h8.Set<float>(1, 2, -3);
h8.Set<float>(2, 0, -3);
h8.Set<float>(2, 1, -3);
h8.Set<float>(2, 2, -3);

int rows = dst1.Rows;
int cols = dst1.Cols;

//Mat t1 = new Mat();

Mat t1 = new Mat(rows, cols, MatType.CV_32F);
Mat t2 = new Mat(rows, cols, MatType.CV_32F);
Mat t3 = new Mat(rows, cols, MatType.CV_32F);
Mat t4 = new Mat(rows, cols, MatType.CV_32F);
Mat t5 = new Mat(rows, cols, MatType.CV_32F);
Mat t6 = new Mat(rows, cols, MatType.CV_32F);
Mat t7 = new Mat(rows, cols, MatType.CV_32F);
```

```
            Mat t8 = new Mat(rows, cols, MatType.CV_32F);

            Cv2.Filter2D(dst1, t1, -1, h1, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t2, -1, h2, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t3, -1, h3, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t4, -1, h4, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t5, -1, h5, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t6, -1, h6, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t7, -1, h7, Cv.Point(0, 0));
            Cv2.Filter2D(dst1, t8, -1, h8, Cv.Point(0, 0));

            //MessageBox.Show(t1.At<float>(1,2).ToString());

            Mat temp = new Mat(1, 8, MatType.CV_32F);

            temp.Set<float>(0, 0, 0);
            temp.Set<float>(0, 1, 0);
            temp.Set<float>(0, 2, 0);
            temp.Set<float>(0, 3, 0);
            temp.Set<float>(0, 4, 0);
            temp.Set<float>(0, 5, 0);
            temp.Set<float>(0, 6, 0);
            temp.Set<float>(0, 7, 0);

            Mat bloodVessels = new Mat(rows, cols,
MatType.CV_32F);
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    bloodVessels.Set<double>(i, j, 0);
                }
            }

            double minVal;
            double maxVal;

            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    temp.Set<float>(0, 0, t1.At<char>(i, j));
                    temp.Set<float>(0, 1, t2.At<char>(i, j));
                    temp.Set<float>(0, 2, t3.At<char>(i, j));
                    temp.Set<float>(0, 3, t4.At<char>(i, j));
                    temp.Set<float>(0, 4, t5.At<char>(i, j));
                    temp.Set<float>(0, 5, t6.At<char>(i, j));
                    temp.Set<float>(0, 6, t7.At<char>(i, j));
```

```
                    temp.Set<float>(0, 7, t8.At<char>(i, j));

                    //Cv2.MinMaxLoc(temp,out minVal,out maxVal);
                    temp.MinMaxLoc(out minVal, out maxVal);

                    //MessageBox.Show(maxVal.ToString());
                    if (maxVal>100)
                    {
                        bloodVessels.Set<double>(i, j, maxVal);
                    }
                }
            }
            Cv2.ImShow("Damarlar",bloodVessels);
        }
    }
}
```

EK – XIX: OpenCV' de Şekil Hatlarının Belirlenmesi Yöntemi Kullanılarak Retinadaki Damarların Belirlenmesi

// PROGRAM 19

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;
using OpenCvSharp.Utilities;
using OpenCvSharp.Blob;
using OpenCvSharp.DebuggerVisualizers2013;
using OpenCvSharp.Extensions;
using OpenCvSharp.UserInterface;
using System.Threading;

namespace opencvSharp
{
    public partial class Form1 : Form
    {
        Mat src;
        public Form1()
        {
            InitializeComponent();
        }

        static string filename = "";
        private void button1_Click(object sender, EventArgs e)
        {
            if (filename == "")
            {
                MessageBox.Show("Lütfen Bir Resim Seçiniz");
                return;
            }

            //median filter combobax
            int kernelsize = 1; ;
            if (medianCB.Text == "Size (1x1)")
                kernelsize = 1;
            else if (medianCB.Text == "Size (3x3)")
```

```
    kernelsize = 3;
else if (medianCB.Text == "Size (5x5)")
    kernelsize = 5;
else if (medianCB.Text == "Size (7x7)")
    kernelsize = 7;
else if (medianCB.Text == "Size (9x9)")
    kernelsize = 9;
else if (medianCB.Text == "Size (11x11)")
    kernelsize = 11;
else if (medianCB.Text == "Size (13x13)")
    kernelsize = 13;
else if (medianCB.Text == "Size (15x15)")
    kernelsize = 15;

//gauss filter combobax
if (GausKernelCB.Text == "Size (1x1)")
    kernelsize = 1;
else if (GausKernelCB.Text == "Size (3x3)")
    kernelsize = 3;
else if (GausKernelCB.Text == "Size (5x5)")
    kernelsize = 5;
else if (GausKernelCB.Text == "Size (7x7)")
    kernelsize = 7;
else if (GausKernelCB.Text == "Size (9x9)")
    kernelsize = 9;
else if (GausKernelCB.Text == "Size (11x11)")
    kernelsize = 11;
else if (GausKernelCB.Text == "Size (13x13)")
    kernelsize = 13;
else if (GausKernelCB.Text == "Size (15x15)")
    kernelsize = 15;

int uy = 1;
if (GaussGB.Text == "1")
    uy = 1;
else if (GaussGB.Text == "2")
    uy = 3;
else if (GaussGB.Text == "3")
    uy = 5;
else if (GaussGB.Text == "4")
    uy = 7;
else if (GaussGB.Text == "5")
    uy = 9;
else if (GaussGB.Text == "6")
    uy = 11;
else if (GaussGB.Text == "7")
    uy = 13;
else if (GaussGB.Text == "8")
```

```
                uy = 15;
            else if (GaussGB.Text == "9")
                uy = 13;
            else if (GaussGB.Text == "10")
                uy = 15;

            //adaptiveThreshold
            AdaptiveThresholdType att = new
AdaptiveThresholdType();
            if (adaptivettCB.Text == "GaussianC")
                att = AdaptiveThresholdType.GaussianC;
            else if (adaptivettCB.Text == "MeanC")
                att = AdaptiveThresholdType.MeanC;

            //block Size
            int bs = 11;
            if (blocksizeCB.Text == "3")
                bs = 3;
            else if (blocksizeCB.Text == "5")
                bs = 5;
            else if (blocksizeCB.Text == "7")
                bs = 7;
            else if (blocksizeCB.Text == "9")
                bs = 9;
            else if (blocksizeCB.Text == "11")
                bs = 11;
            else if (blocksizeCB.Text == "13")
                bs = 13;
            else if (blocksizeCB.Text == "15")
                bs = 15;
            else if (blocksizeCB.Text == "17")
                bs = 17;
            else if (blocksizeCB.Text == "19")
                bs = 19;
            else if (blocksizeCB.Text == "21")
                bs = 21;

            // c (mean weighted mean)
            int c=2;
            if (cCB.Text == "1")
                c = 1;
            else if (cCB.Text == "2")
                c = 2;
            else if (cCB.Text == "3")
                c = 3;
            else if (cCB.Text == "4")
                c = 4;
            else if (cCB.Text == "5")
```

```
                c = 5;
            else if (cCB.Text == "6")
                c = 6;
            else if (cCB.Text == "7")
                c = 7;
            else if (cCB.Text == "8")
                c = 8;
            else if (cCB.Text == "9")
                c = 9;
            else if (cCB.Text == "10")
                c = 10;

            // contourRetrival
            ContourRetrieval cr = new ContourRetrieval();
            if (cbCr.Text == "List")
                cr = ContourRetrieval.List;
            else if (cbCr.Text == "Tree")
                cr = ContourRetrieval.Tree;
            else if (cbCr.Text == "External")
                cr = ContourRetrieval.External;
            else if (cbCr.Text == "CComp")
                cr = ContourRetrieval.CComp;
            else if (cbCr.Text == "FloodFill")
                cr = ContourRetrieval.FloodFill;

            double x = Convert.ToDouble(numericUpDown1.Value);

            pictureBox1.ImageLocation = filename;

            src = Cv2.ImRead(filename);
            Mat dst = new Mat();
            Cv2.CvtColor(src, dst, ColorConversion.BgrToGray);
            Mat filter = new Mat();
            if (gaussRB.Checked==true)
                Cv2.GaussianBlur(dst, filter,
Cv.Size(kernelsize,kernelsize),uy);
            else if(medinRB.Checked == true)
                Cv2.MedianBlur(dst, filter, kernelsize);
            Mat adapdive_image = new Mat();
            Cv2.AdaptiveThreshold(filter, adapdive_image, 255,
att, ThresholdType.BinaryInv,bs,c);
            OpenCvSharp.CPlusPlus.Point[][] contours;
            OpenCvSharp.CPlusPlus.HiearchyIndex[] hierarchy;
            OpenCvSharp.CPlusPlus.Point[][] intruders = null;
            OpenCvSharp.CPlusPlus.Point[][] con2;
            Cv2.FindContours(adapdive_image, out contours,out
hierarchy, cr, ContourChain.ApproxSimple);
            int contoursize = contours.Length;
```

```
            con2 = contours;
            Array.Resize(ref intruders, 0);
            for (int i = 0; i < contoursize ; i++)
            {
                double area = Cv2.ContourArea(contours[i]);
                if (area > x)
                {
                    Array.Resize(ref intruders, intruders.Length +
1);
                    intruders[intruders.Length - 1] = contours[i];
                }
            }
            Cv2.DrawContours(src, intruders, -1, Scalar.Blue, 1);
            Cv2.ImWrite("D:/a.jpg", src);
            pictureBox2.ImageLocation = "D:/a.jpg";
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            cbCr.SelectedIndex = 0;
            numericUpDown1.Value = 50;
            medianCB.SelectedIndex = 0;
            GaussGB.SelectedIndex = 0;
            GausKernelCB.SelectedIndex = 0;
            adaptivettCB.SelectedIndex = 0;
            blocksizeCB.SelectedIndex = 4;
            cCB.SelectedIndex = 1;
            medinRB.Checked = true;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if(checkBox1.Checked==true)
                Cv2.ImShow("Adaptive Resim", src);

        }

        private void button3_Click(object sender, EventArgs e)
        {
            filename = "";
            Thread newThread = new Thread(new
ThreadStart(ThreadMethod));
            newThread.SetApartmentState(ApartmentState.STA);
            newThread.Start();
            while (newThread.IsAlive)
            {
                if (filename != "")
                    newThread.Abort();
```

```
            }
            pictureBox1.ImageLocation = filename;
            pictureBox2.Image = null;
        }

        static void ThreadMethod()
        {
            OpenFileDialog dlg = new OpenFileDialog();
            dlg.ShowDialog();
            filename = dlg.FileName;

        }

        private void medinRB_CheckedChanged(object sender,
EventArgs e)
        {
            label5.Enabled = true;
            medianCB.Enabled = true;
            label6.Enabled = false;
            label7.Enabled = false;
            GaussGB.Enabled = false;
            GausKernelCB.Enabled = false;
        }

        private void gaussRB_CheckedChanged(object sender,
EventArgs e)
        {
            label5.Enabled = false;
            medianCB.Enabled = false;
            label6.Enabled = true;
            label7.Enabled = true;
            GaussGB.Enabled = true;
            GausKernelCB.Enabled = true;
        }

    }
}
```

ÖZGEÇMİŞ

1982 yılında Elazığ'da doğdu. İlk, orta ve lise öğrenimini Elazığ'da tamamladıktan sonra 2001 yılında Fırat Üniversitesi, Teknik Eğitim Fakültesi, Elektronik Bilgisayar Eğitimi Bölümü, Bilgisayar Öğretmenliğini kazandı. 2005 yılında bu bölümden mezun oldu. Yine 2012 yılında Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik Bilgisayar Eğitimi Bölümü, Bilgisayar Sistemleri Anabilim Dalında yüksek lisans eğitimine hak kazandı. 2006 yılında Milli Eğitim Bakanlığında Bilişim Teknolojileri Teknik Öğretmeni olarak göreve başladı. Halen Adıyaman Besni Osman İsot Mesleki ve Teknik Anadou Lisesinde görevini sürdürmektedir. Yabancı dili İngilizce'dir.