T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İNSAN BİLGİSAYAR ETKİLEŞİMLİ GÖRÜNTÜ İŞLEME UYGULAMALARI

YÜKSEK LİSANS TEZİ

Hüseyin KUTLU

Anabilim Dalı: Elektronik ve Bilgisayar Eğitimi
Programı: Bilgisayar Sistemleri Eğitimi
Tez Danışmanı: Doç. Dr. Engin AVCI

HAZİRAN-2013

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İNSAN BİLGİSAYAR ETKİLEŞİMLİ GÖRÜNTÜ İŞLEME UYGULAMALARI

YÜKSEK LİSANS TEZİ

Hüseyin KUTLU
111131104

Tezin Enstitüye Verildiği Tarih:26 Haziran 2013
Tezin Savunulduğu Tarih:23 Temmuz 2013

Tez Danışmanı: Doç.Dr. Engin AVCI (F.Ü)
Diğer Jüri Üyeleri: Yrd. Doç. Dr. Mustafa KAYA (F.Ü)
Yrd. Doç. Dr. Erkan TANYILDIZI (F.Ü)

HAZİRAN-2013

ÖNSÖZ

Gittikçe karmaşıklaşan dünyamızda insanoğlunun her şeyi kontrol altına alması daha da zorlaşmakta ve bilgisayar sistemlerine ihtiyaç duyulmaktadır. İnsanların ihtiyaçlarını karşılayan, yaşamını kolaylaştıran çalışmalardan bazıları ise Image Processing" (Görüntü işleme) ve "Computer Vision" (Bilgisayarlı Görü) konularıdır. Bu tez çalışmasında gerçek zamanlı görüntüdeki el hareketleri ile insan bilgisayar etkileşimi sağlanmıştır. Yapılan çalışmada, görüntü işleme aracı olarak OpenCV kullanılmıştır. Gerçek zamanlı görüntülerdeki hareketli nesneleri tanımlamada ve belirlemede en çok kullanılan genel bir yaklaşım olan arka plan çıkarım yöntemleri ve arka plan çıkarım yöntemlerinin OpenCV ile uygulaması açıklanmıştır.

Çalışmam esnasında her türlü hoşgörü, destek ve bilgisiyle bana katkıda bulunan değerli danışmanım Doç Dr. Engin AVCI' ya, uygulama geliştirme esnasında yardımını esirgemeyen tüm hocalarıma, her zaman bana destek veren ve yanımda olan çok kıymetli aileme, arkadaşlarıma teşekkürü bir borç bilirim.

Hüseyin KUTLU

ELAZIĞ- 2013

İÇİNDEKİLER

Sayfa No

ÖNSÖZ ..... **II**

İÇİNDEKİLER ..... **III**

ÖZET ..... **VI**

SUMMARY ..... **VII**

ŞEKİLLER LİSTESİ ..... **VIII**

TABLOLAR LİSTESİ ..... **IXI**

KISALTMALAR LİSTESİ ..... **XI**

1. GİRİŞ **1**

2. GERÇEK ZAMANLI GÖRÜNTÜYÜ OLUŞTURAN BİLEŞENLER **3**

2.1. Piksel 3

2.2. Çerçeve 4

2.3. Çözünürlük 4

2.4. Renk Kanalları 4

2.5. Renk Uzayları 4

2.5.1. Gri Ölçekli Görüntü Renk Uzayları 5

2.5.2. RGB Renk Uzayı 5

2.5.3. YUV Renk Uzayı 6

2.5.4. YIQ Renk Uzayı 7

2.5.5. YCbCr Renk Uzayı 7

2.5.6. HSV, HSB, HSI, HSL Renk Uzayı 8

3. GERÇEK ZAMANLI GÖRÜNTÜLERDE ÖN PLAN ARKA PLAN AYRIMI **9**

3.1. Arka Plan Modellemesi 10

3.1. 1. Özyinelemesiz Yöntemler 11

3.1. 1.1. Çerçeve Farkı Yöntemi 11

3.1. 1.2. Ortanca Süzgeç Yöntemi 12

3.1. 1.3. Doğrusal Tahmin Edici Süzgeç Yöntemi 12

3.1. 1.4. Parametrik Olmayan Model Yöntemi 12

3.1. 2. Özyinelemeli Yöntemler 13

3.1. 2.1. Yaklaşık Ortanca Yöntemi 13

3.1. 2.2. Kalman Süzceç Yöntemi 13

3.1. 2.3. Gauss Fonsyonları Karışımı Yöntemi 14
3.1. 2.4. Öz Arka Plan Yöntemi 15
3.1. 2.5. Kod Çizelgesi (CodeBook) Yöntemi..... 15
4. OPENCV..... 22
4.1. OpenCV'nin Temel Özellikleri..... 23
4.2. OpenCV ile Neler Yapabiliriz..... 24
4.3. OpenCV Bileşenleri..... 26
4.3.1. CV Bileşeni..... 26
4.3.2. ML Bileşeni 26
4.3.3. HighGUI Bileşeni..... 27
4.3.4. CxCore Bileşeni 27
4.3.5. CvAux Bileşeni 27
4.4. OpenCV Kurulumu ve Kullanımı 28
4.5. Temel OpenCV Komutları..... 39
4.5.1. Pencere Yönetimi 39
4.5.2. Girdi Kullanımı 40
4.5.3. OpenCV Temel Veri Yapıları 42
4.6. OpenCV ile Temel Görüntü İşleme Uygulamaları 47
4.6.1. OpenCV ile Resim Okumak 47
4.6.2. OpenCV İle Resmi Matris Olarak Almak Ve İşlemek..... 48
4.6.3. OpenCV ile Morfolojik İşlemler..... 49
4.6.3.1 Genişleme ve Aşınma..... 49
4.6.3.2. Açma ve Kapatma 50
4.6.3.3. Bağlı Bileşenler Analizi..... 51
4.6.3.4. Eşikleme..... 52
4.6. 4 OpenCV ile Görüntüye Filitre Uygulama..... 54
4.6.5. OpenCV ile Kenar Bulma..... 57
4.6.6. Kamera ile Gerçek Zamanlı Görüntü Yakalama..... 57
4.7. OpenCV'de Arka Plan Ayırma 58
4.7.1. Arka Plan Çıkarımı..... 59
4.7.2. Arka Plan Çıkarımının Zayıf Yönleri..... 59
4.7.3. Arka Plan Modelleme..... 60
4.7.3.1. Çerçeve Farkı Yöntemi..... 64

4.7.3. 2. Arka Plan Ortalama Yöntemi 65
4.7.3. 3. Ortalama Varyans Kovaryans Toplanması 70
4.7.3. 4. Kod Çizelgesi (CodeBook) Yöntemi 73
4.7.3.4.1. Kod Çizelgesi Arka Plan Modelinin Kullanımı 82
4.7.3.4.2. Ön Plan Temizliği İçin Bağlantılı Bileşenler 83
4.8. OpenCV'de Arka Plan Çıkarma Yöntemlerinin Karşılaştırılması 89
4.9. Watershed Algoritması 93
4.10. İç Boyama İle Görüntü Onarımı 94
4.11. Ortalama Kayması ile Bölütleme 96
4.12. Delanuay üçgenlemesi Voronoi Tesselasyonu 98
4.12.1. Delenuay yada Voronoi Alt Birim Oluşturma 100
4.12.2. Delanuay Alt Biriminin Navigasyonu 102
4.12.3. Kenarlarda Yürümek 104
4.12.4. Kenarlardan Çıkan Noktalar 107
4.12.5. Kenar yada Tepe Noktanın Bulunması İçin Bir Dış Noktanın Kullanımı 107
4.12.6. Bir Dizi Nokta ve Kenar İçinden Dolaşmak 108
4.12.7. Dış Bükey Görüntüdeki Sınırlayıcı Üçgen Yada Kenar Tanımlaması ve Görüntüde Hareket Etmek 109
5. İNSAN BİLGİSAYAR ETKİLEŞİMİ UYGULAMASI 113
5.1. Uygulamanın Geliştirilmesi 113
6. SONUÇ 118
KAYNAKLAR 120
EKLER 123
ÖZGEÇMİŞ 129

ÖZET

Bu tez çalışması gerçek zamanlı görüntüde değişim halinde ön plan nesnesi olan el hareketleri ile bilgisayarı etkileme ve komut verme ile ilgilidir. Tezin amacı gerçek zamanlı bir video görüntüsünden bilgisayara el işareti ile komutlar vermektir. Gerçek zamanlı görüntüde hareket eden ön plan nesnelerini belirlemek için kullanılan yöntemlerin başında arka plan çıkarımı gelir. Arka plan çıkarımı basit olarak düşünüldüğünde ardışık görüntü karelerindeki aynı pikselleri arka plan olarak kabul edip yok ederek, sadece farklı pikselleri ortaya çıkartıp hareketli olan nesneyi tespit etme esasına dayanır. Fakat bu yöntem ile rüzgârın salladığı bir ağaç dahi hareket olarak tespit edileceğinden, arka plan bilgisini sürekli güncelleyen algoritmalar geliştirilmiştir. Bu tez çalışmasında arka plan çıkarım yöntemleri ve açık kaynak kodlu görüntü işleme kütüphanesi olan OpenCV görüntü işleme aracı ile arka plan çıkarım yöntemleri incelenmiştir. Yöntemler incelenmiş ve etkin bellek kullanımı, hızlı sonuç üretimi, dış ortam şartlarında çalışabilmesi ve gerçek zamanlı video görüntülerindeki kullanılabilirliği dolayısıyla yapılan uygulamada arka plan çıkarım metotlarından Kod Çizelgesi (Codebook) metodu uygulanmıştır.

Anahtar Kelimeler: Görüntü İşleme, Gerçek Zamalı Görüntü, OpenCV, Arka Plan Çıkarımı, Kod Çizelgesi (Codebook) metodu

SUMMARY

Human Computer Interactive Image Processing Applications

The aim of study is related to giving commands and influence of computer with hand gestures which is the foreground object in real-time image in flux. The aim of the thesis is to give commands to computer with hand marks from a video image in real-time. Background extraction comes to the fore at the top of the most important methods used to determine the moving foreground objects in real time image. When simply considered, background extraction accepts and then exterminates the same pixels in the sequential image captures as the background, detects just different pixels and is based on the detection of the moving object. However, with this method, as even a tree in the wind shaking will be determined as the act, algorithms updating the background information continuously have been developed. In this study, an open source image processing library OpenCV and background extraction methods were examined. After the methods have been examined and due to the efficient use of memory, the production of fast results, its working in ambient conditions, and its application in real-time video displays, the availability of background extraction methods Code Schedule (Codebook) method was applied.

Keywords: Image Processing, Real Time Images, OpenCV, Background Extraction, Codebook method

ŞEKİLLER LİSTESİ

Sayfa No

Şekil 2.1 Gerçek Zamanlı Görüntüyü Oluşturan Bileşenler 3
Şekil 2.2. Çözünürlük Piksel Sayısı İlişkisi 4
Şekil 2.3. RGB Renk Uzayı Bileşenleri 5
Şekil 2.4. YUV Renk Uzayı Bileşenleri 6
Şekil 2.5. YIQ Renk Uzayı Bileşenleri. 7
Şekil 2.6. YCbCr Renk Uzayı Bileşenleri 7
Şekil 2.7. Altıgen Huni Şeklindeki HSV Renk Uzayı 8
Sekil 3.1. Ön Plan Arka Plan Ayrımında Kullanılan Yöntemlerin Temel Algoritması 9
Sekil 3.2. Kod Çizelgesinde Renk Değişimi ve Parlaklık Hesaplama Modeli 9
Şekil 4.1. OpenCV Kütüphanesini Oluşturan Bileşenler 26
Şekil 4.2. OpenCV Kurulum Sihirbazı 28
Şekil 4.3. OpenCV Kurulum Sihirbazı 28
Şekil 4.4. OpenCV Kurulum Sihirbazı 29
Şekil 4.5. OpenCV Kurulum Sihirbazı 29
Şekil 4.6: OpenCV Kurulum Sihirbazı 30
Şekil 4.7: OpenCV Kurulum Sihirbazı 30
Şekil 4.8: OpenCV Kurulum Sihirbazı 31
Şekil 4.9: Visual Studio 2010'da C++ Projesi Oluşturma 32
Şekil 4.10: Visual Studio 2010'da C++ Projesi Oluşturma 32
Şekil 4.11: Projenin Properties Menüsünü Açma 33
Şekil 4.12: Projenin Properties Menüsünü Açma 33
Şekil 4.13: Properties Penceresi Ayarları 34
Şekil 4.14. Properties Penceresi Ayarları 35
Şekil 4.15. Properties Penceresi Ayarları 36
Şekil 4.16: Properties Penceresi Ayarları 37
Şekil 4.17: Basit Bir Programla Kurulumu Test Etme 38
Şekil 4.18: OpenCv de Bir Resmin Görüntülenmesi 47
Şekil 4.19: Resmi Matris Olarak Açma ve İşlem Yapma 49
Şekil 4.20: OpenCV'de Morfolojik İşlemler 51
Şekil 4.21: OpenCV'de Morfolojik İşlemler ve Thresholding 53
Şekil 4.22: OpenCV'de Resme temel Filitreler Uygulama 56

Şekil 4.23: OpenCv de Temel Kenar Bulma Fonksyonları.....57
Şekil 4.24: Rüzgarda Hareket Eden Ağaç.....61
Şekil 4.25: Ortalama Farklılıkları.....65
Şekil 4.26: Kod Çizelgeleri Yoğunluk Değerleri 74
Şekil 4.27: Kod Çizelgeleri Yoğunluk Değerleri 75
Şekil 4.28: Çerçeve Farkı.....90
Şekil 4.29: Çerçeve Farkı.....91
Şekil 4.30: Ortalama Yöntemi İle Arka Plan Çıkarma 92
Şekil 4.31: Watershed Algoritması.....93
Şekil 4.32. İç Boyama.....94
Şekil 4.33. İç Boyama.....95
Şekil 4.34: Mean – Shift Bölütleme.....98
Şekil 4.35: Delenuay üçgenlemesi.....99
Şekil 4.36: Voronoi Tesselasyonu 100
Şekil 4.37: Bağlı Kenar.....103
Şekil 4.38: Dörtlü Kenar 104
Şekil 4.40: e Kanarı 105
Şekil 5.1: Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 1 komutu verme.....114
Şekil 5.2: Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 2 komutu verme.....115
Şekil 5.3: Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 3 komutu verme.....115
Şekil 5.4: Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 4 komutu verme.....116
Şekil 5.5. Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 5 komutu verme.....116
Şekil 5.6. Web Cam'den alınan gerçek zamanlı görüntüde el işaretinin görüntü içnde hareket ettirilmesi.....116

TABLOLAR LİSTESİ

Tablo 2.1. RGB renk uzayı karışım oranlarına göre elde edilen renk tonları 6

KISALTMALAR LİSTESİ

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,

MIT : Massachusetts Teknoloji Enstitüsü

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

MoG: Gauss fonksiyonları karışımı yöntemi

OpenCV: Open Source Computer Vision Library

1. GİRİŞ

Gittikçe karmaşıklaşan dünyamızda insanoğlunun her şeyi kontrol altına alması daha da zorlaşmakta dolayısıyla otomasyon sistemlerine ihtiyaç duyulmaktadır [1]. Bu konuda insanların ihtiyaçlarını karşılayan, yaşamını kolaylaştıran çalışmalardan bazıları ise Image Processing" (Görüntü İşleme) ve "Computer Vision" (Bilgisayarlı Görü) konularıdır. Görüntü İşleme ölçülmüş veya kaydedilmiş olan elektronik (dijital) görüntü verilerini, bilgisayar ve yazılımlar yardımı ile elektronik ortamda amaca uygun şekilde değiştirmeye yönelik yapılan bilgisayar çalışmasıdır [2-5]. Bilgisayarlı görü ise üç boyutlu dünyadan elde edilen görüntülerin verdiği bilgileri yorumlamak ve karakterize etmektir [3-4].

Bilgisayara bağlı bir kamera ile görü, insan ve bilgisayar etkileşimini geliştirecek potansiyeli yüksek bir tekniktir [9-10]. Çeşitli algoritmalarla nesnelerin takibi yapılabilir, nesneler tanımlanabilir ve bu imkânlar üzerine birçok ilginç uygulama bina edilebilir. Çeşitli ticari amaçların yanı sıra savunma sanayi, tıp, robot endüstrisi, biyometrik uygulamalar, web siteleri ve engellilerin yaşamlarını kolaylaştırmak gibi çokça uygulamaları mevcuttur.

Gerçek zamanlı görüntü işleme teknikleri ile yapılan çalışmalar bilgisayar bilimlerin insanoğlunun hayatını ne derece kolaylaştırdığını ortaya koymaktadır. Bu çalışmalardan bazıları yol gözetleyici kameralarla yapılan trafik ve güvenlik uygulamaları, nesne takibi, nesne sayımı, nesne tanımlama, yüz tanıma, plaka okuma, metin okuma, tarım arazilerinin kontrolü gibi uygulamalardır. Aritmetiksel ve mantıksal işlemleri yapabilen, karar verebilen, yapay sinir ağları ile öğrenebilen, elektrik sinyalleri ile hareket edebilen bilgisayarlara adeta bir göz olan bu alanda her geçen gün algoritmalar güçlendirilmektedir.

Bilgisayarlı görü ve gerçek zamanlı görüntü işleme işlemci gücü isteyen uygulamalardır. Bu durumun öneminin farkında olan işlemci firması Intel gerçek zamanlı uygulamaları hedef alarak "OpenCV" isimli proje ortaya çıkarmıştır.

OpenCV Massachusetts Teknoloji Enstitüsü (MIT) ve bazı diğer üniversitelerde geliştirilmeye başlanan temel görüntü işleme kütüphaneleri, İntel firmasının da desteğiyle geliştirilen C ve C++ dilleri ile yazılmış, dünyanın her tarafından geliştiricileri olan, açık kaynak kodlu BSD lisansı ile lisanslanmış büyük ve ünlü bir “Bilgisayarla Görü/Görme” kütüphanesidir. Akademik ve ticari kullanımı ücretsiz olan bu kütüphane Windows, Linux, MacOS X gibi farklı platformlarda kullanılabilir [3].

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 [11].

Gerçek zamanlı bir görüntüdeki hareketlerden anlamlı veriler çıkarılabilmesi için hareketli ön plan nesnelerinin hareketsiz arka planlardan nesnelerinde ayrılması gerekir ve bu

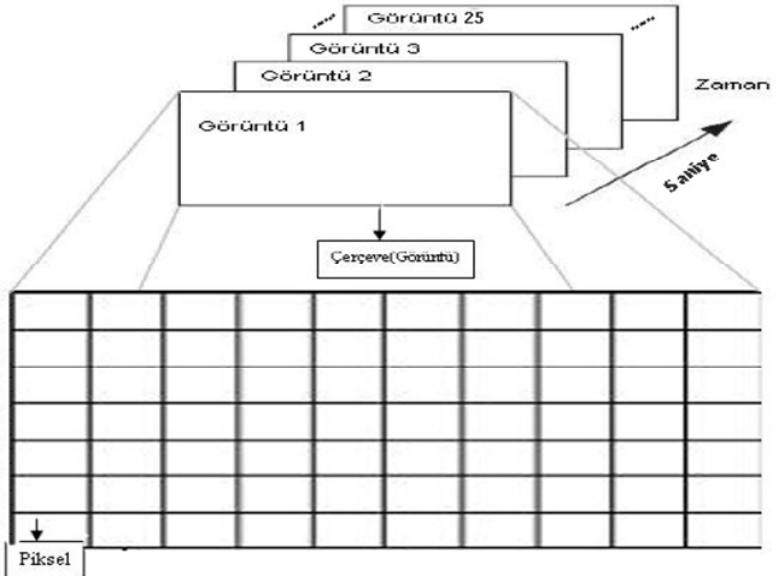
işlem ön plan ve arka plan ayrımı olarak bilinir. Ön plan ve arka plan ayrımı nesnelerin tanımlanması ve takibi gibi uygulamalarda en çok kullanılan genel yaklaşımdır [11].

Bu tez çalışmasının amacı, gerçek zamanlı bir görüntüdeki anlamlı hareketlerden komut alan bir uygulama gerçekleştirmektir. Bu uygulama gerçekleştirilirken OpenCV görüntü işleme kütüphanesinden faydalanılmıştır. Tez çalışmasında öncelikle OpenCV görüntü işleme kütüphanesi tanıtılmış, ardından video görüntülerinde hareketli nesnelerin tanınması ve belirlenmesinde en çok kullanılan genel bir yaklaşım olan arka plan çıkarımının OpenCV ile gerçekleştirilen yöntemleri incelenmiş ve bu yöntemler içerisinden etkin bellek kullanımı, hızlı sonuç üretimi, dış ortam şartlarında çalışabilmesi ve gerçek zamanlı video görüntülerindeki kullanılabilirliği dolayısıyla yapılan uygulamada arka plan çıkarım metotlarından Kod çizelgesi (Codebook) metodu uygulanmıştır.

2. GERÇEK ZAMANLI GÖRÜNTÜYÜ OLUŞTURAN BİLEŞENLER

İnsan gözü saniyede arka arkaya geçen 25 resimi canlı bir görüntüymüş gibi algılamaktadır. Saniyede arka arkaya gösteren 25 adet sayısal olarak derlenmiş resim dizisine gerçek zamanlı görüntü veya video denilir. Video kelimesi köken olarak Latince Videre kelimesinden gelmektedir ve video *görüyorum* demektir [2].

Aşağıda videoyu oluşturan temel bileşenler. Sırasıyla anlatılmaktadır.



Şekil 2.1. Gerçek Zamanlı Görüntüyü Oluşturan Bileşenler

2. 1. Piksel

Tüm sayısal görüntülerin temel yapı taşıdır. Piksel kelimesi ingilizce **Pic**ture **El**ement (Resim Elemanı) kelimelerinin ilk hecelerinin bileşiminden oluşmuştur. Sayısal bir görüntüdeki belirli bir eni ve boyu veya çapı olmayan tek bir nokta veya karedir. Sayısal bir görüntüye sayır ve sütun olarak sık bir şekilde dizilen pikseller görüntüyü oluştururlar.

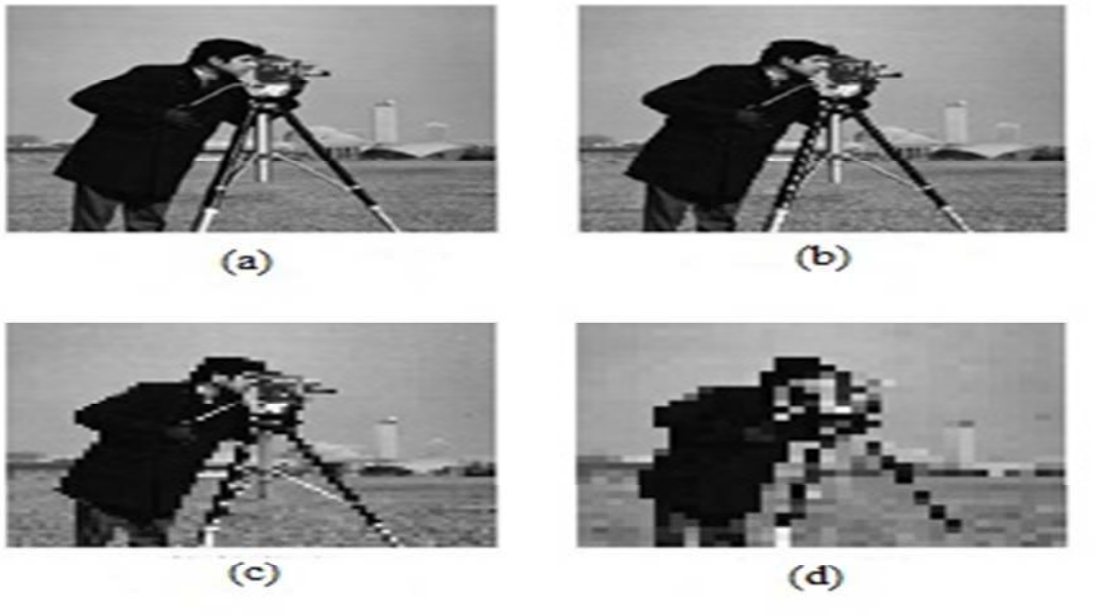
Sayısal görüntü eninde (satır sayırı) ve boyunda (sütun sayısı) bulunan piksel sayısı ile tanımlanır. Görüntüdeki piksel sayısı arttıkça görüntü kaliteside artmaktadır.

2. 2. Çerçeve (Frame)

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

2. 3. Çözünürlük

Bir resmi oluşturan piksellerin birim uzunluktaki sayısına çözünürlük denilir. Başka bir ifadeyle piksellerin birbirine olan uzaklıklarının ölçüsüdür. Aynı fiziksel büyüklüğe sahip görüntüler aynı çözünürlükte olmayabilirler [2]. Aşağıdaki görüntüler örnek olarak verilebilir. Çözünürlük arttıka başka bir değişle çerçevedeki piksel sayısı arttıkça görüntü kalitesi artar.



Şekil 2.2. Çözünürlük Piksel Sayısı ilişkisi (a) 256x256 piksel, (b) 128x128 piksel, (c) 64x64 piksel, 32x32 piksel

2. 4. Renk Kanalları

Renk kanalları sayısal bir görüntüdeki pikselin ana renk seviyelerinin ayrı ayrı tutulduğu kanallardır [2]. Bu renk kanalları bir araya gelerek renkli görüntüler elde edilir [12].

2. 5. Renk Uzayları

Renk uzayları renkleri tanımlamak için kullanılan matematiksel bir modeldir. Bir rengi belirlemek için birbirinden bağımsız 3 değişkene 3 boyuta ihtiyaç vardır. Tüm renk uzayları, kamera ve tarayıcı gibi aygıt kaynaklı RGB renk uzayı bilgisi kullanılarak elde edilmektedir.

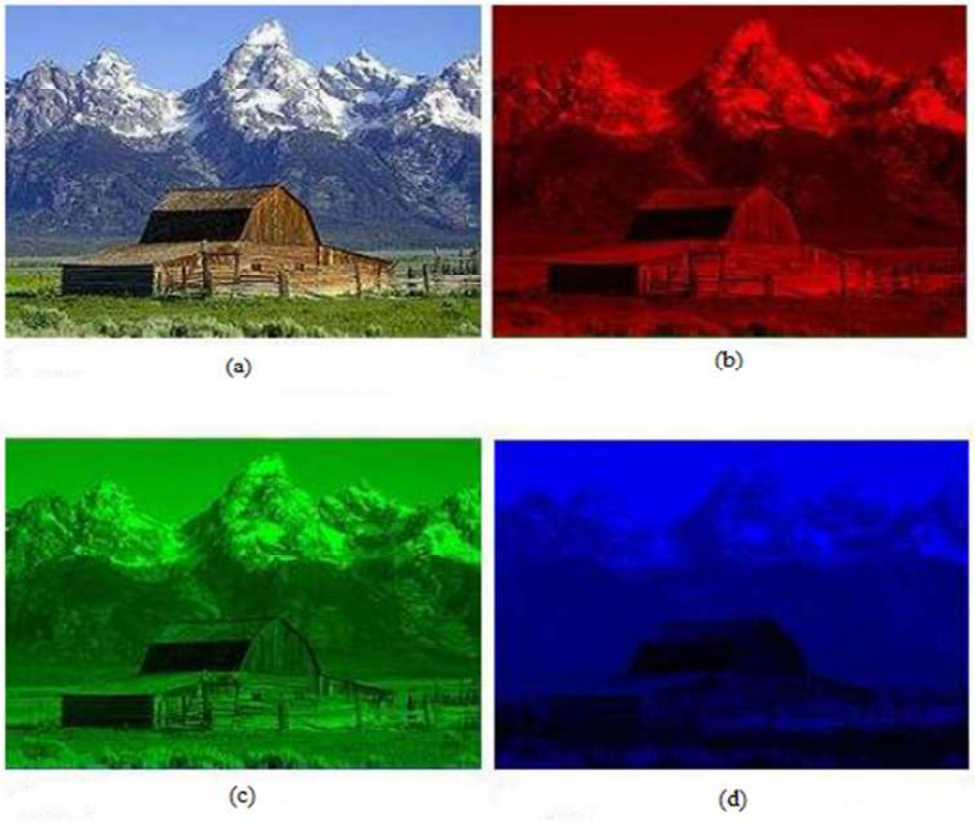
2. 5. 1. Gri Ölçekli Görüntü Renk Uzayı

Gri ölçekli görüntülerde her bir piksel tek bir değerden oluşur ve bu değer görüntüyü oluşturan oluşturan bir pikselin ışık açısından aydınlık (beyaz) yada karanlık (siyah) olduğunu yani yoğunluğunu gösterir.

Gri ölçekli görüntüler tek bir renk kanalı içerirler. En düşük ışık yoğunluğu karanlığı yani siyah rengi, en yüksek ışık yoğunluğu ise aydınlığı yani beyaz rengi gösterirken arada kalan yoğunluklar grinin tonlarını gösterir. Gri ölçekli görüntüler 8 bit derinliğe sabitlerdir ve piksellerin değerleri ikilik tabanda 8 bit ile temsil edilen 0 ile 255 arasında değişir. 0 siyah rengi temsil ederken 255 ise beyaz rengi temsil eder.

2. 5. 2. RGB Renk Uzayı

RGB renk uzayı yoğun bir kullanım alanına sahiptir. İsmini ingilizce kırmızı, yeşil, mavi anlamına gelen **R**ed – **G**reen – **B**lue kelimelerinin baş harflerinden almıştır. Bir biri ile belirli oranlarda karışa bilen üç temel renktir. Diğer renkler ise bu iç temel renkten oluşmuştur.



Şekil 2.3. 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 2.1'de çizelgede karışım oranlarına göre ortaya çıkan renkler gösterilmiştir.

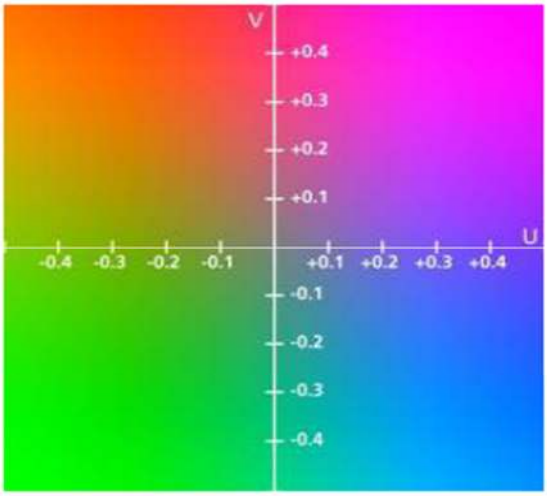
Tablo 2.1. RGB renk uzayı karışım oranlarına göre elde edilen renk tonları

	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 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 ve bazı işlem basamakları daha hızlı yapılabilir. Bu sorunun üstesinden gelmek için parlaklık ve iki farklı renk sinyali kullanan renk uzayları geliştirilmiştir. Bu renk standartları ise YUV, YIQ, YCbCr ve HSV renk uzaylarıdır.

2. 5. 3. YUV Renk Uzayı

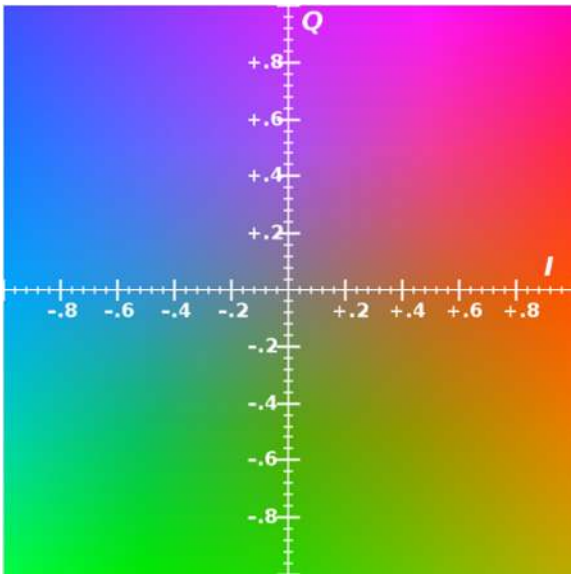
Görüntünün ışık yoğunluğu (parlaklık veya luminance) olan, siyah beyaz resimdeki yoğunluk bilgisini Y bileşeninde tutan U ve V bileşeninde ise renk bilgisini barındıran bir renk uzayıdır.



Şekil 2.4. YUV renk uzayı bileşenleri. U ve V koordinat düzleminin bileşenleri renk bilgisini oluşturur.

2. 5. 4. YIQ Renk Uzayı

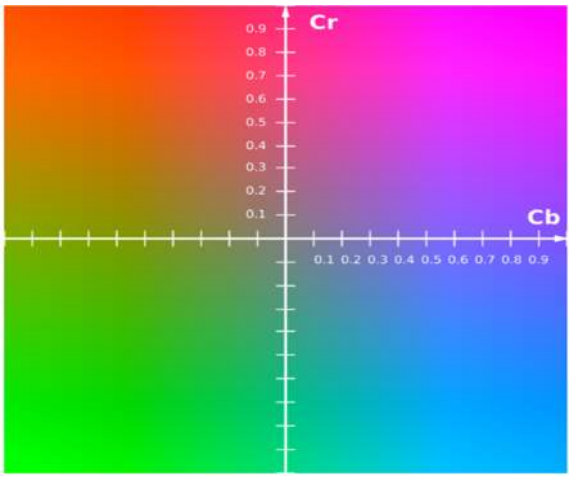
YIQ renk uzayı YUV renk uzayından türetilmiştir. Y ışık yoğunluğu bilgisini tutarken, I ve Q renk bilgisini tutarlar.



Şekil 2.5. YIQ renk uzayı bileşenleri. I ve Q koordinat düzleminin bileşenleri renk bilgisini oluşturur.

2. 5. 5. YCbCr Renk Uzayı

YCbCr Y ile luminance (parlaklık) sinyalini, Cb ve Cr ile ise renk (chrominance) bilgilerini saklayan bir renk uzayıdır. YCbCr renk uzayı, dünya çapında sayısal video standardı oluşturma çabaları sırasında ortaya çıkmıştır. Y, 8 bitliklik 16-235 aralığında tanımlanmaktadır. Cb ve Cr ise de 16-240 arasında tanımlanmaktadır [13].



Şekil 2.6. YCbCr Renk Uzayı, Cr ve Cb koordinat düzleminin bileşenleri renk bilgisini oluşturur.

2. 5. 6. HSV, HSB, HSI ve HLS renk uzayları

HSV Hue-Saturation-Value yani renk tonu – doygunluk - değer, HSB Hue – Saturation - Brightness yani renk tonu – doygunluk – parlaklık, HSI Hue – Saturation - Intensity yani Renk tonu – Doygunluk – Yoğunluk, HLS Hue – Luminance – Saturation yani Renk tonu – Parlaklık - Doygunluk, kelimelerinin baş harfinden oluşmuştur.

Renk tonu, sarı, mavi, yeşil gibi rengin baskın dalga uzunluğunu belirler. Açısal bir değerdir.

Doygunluk, rengin canlılığını belirler. Yüksek doygunluk canlı renklere neden olurken, düşük olasılık rengin gri tonlarına yaklaşmasına neden olur. 0-100 arasında değişir.

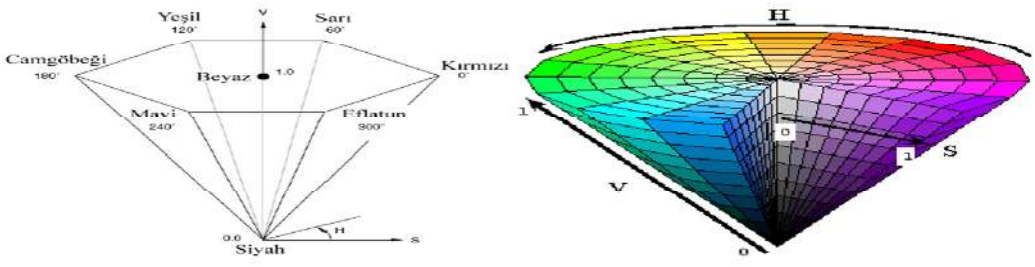
Parlaklık ise rengin aydınlığını yani içindeki beyaz oranını belirler. 0-100 arasından değişir.

Yoğunluk ise ışık miktarı demektir ve parlaklıkla eş anlamlıdır.

HLS ve HIS birbirine çok benzerdirler. Parlaklık bileşeni L yerine yoğunluk bileşeni I kullanılmıştır. HSI ile HSV arasındaki fark ise parlaklık bileşeninin hesaplanma şekli, I yada V renk doyumunun dağılım ve dinamik aralığın değişmesiyle gerçekleşmektedir.

HSV renk uzayı 1978 yılında Alvy Ray Smith tarafından tanımlanmıştır. Amacı RGB uzayına göre insan görü düzeneğine daha yakın bir yapı oluşturmaktır. HSV, RGB renk uzayından doğrusal olmayan bir dönüşüm ile elde edilir. Her ne kadar HSV ve HSB aynı uzayı tanımlasalar da HSL farklı bir renk uzayıdır.

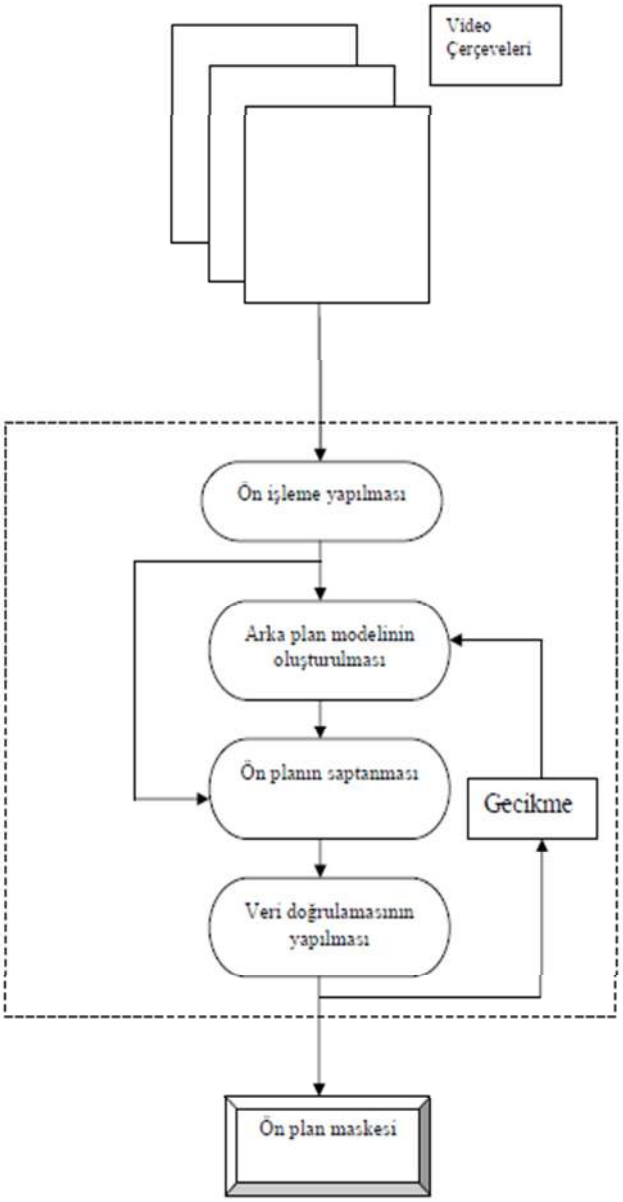
HSI renk uzayı, *R, G, B* değerlerine eşit oranda bağlı olan parlaklık değerleri ile doğrudan alakalı katsayı, eşitleme, histogram gibi geleneksel resim işleme metotları için en iyi renk uzayıdır. HSV renk uzayı ise renk doyumu açısından büyük bir dinamik aralığa sahip olduğu için, renkleri değiştirme ya da renk yoğunluğu ayarlamada kullanılmaktadır.



Şekil 2.7. Altıgen huni şeklinde HSV renk uzayı

3. GERÇEK ZAMANLI GÖRÜNTÜLERDE ÖNPLAN ARKA PLAN AYRIMI

Gerçek zamanlı görüntülerde önemli aşamalardan biri hareketli nesnelerin tespit edilmesidir. Ön plan arka plan ayrımı gerçek zamanlı görüntülerde hareketli nesnelerin tespit edilmesinde yaygın olarak kullanılan bir yöntemdir. Gerçek zamanlı görüntüde hareketli nesne takibi trafik kontrolü, savunma, güvenlik, gibi alanlarda ve araç teknolojilerinden otomatik yönlendirme, sollama, takip, şoförün yorgunluk durumu gibi alanlarda kullanılmaktadır. Kullanım alanının önemli ve yaygın olması dolayısıyla ön plan arka plan ayrımında farklı yöntemler geliştirilmiştir. Fakat genel algoritma aşağıda verilen şekildeki gibidir.[2]



Şekil 3.1. Ön plan arka plan ayrımında kullanılan yöntemlerin temel algoritması

Ön plan, arka plan ayırma algoritmalarındaki dört temel adım; ön işleme (preprocessing), arka plan modelinin oluşturulması (background modelling), ön planın çıkarılması (foreground detection) ve veri doğrulaması (data validation) yapılmasından oluşur [2].

Ön işleme aşamasında gelen gerçek zamanlı videoyu oluşturan çerçeveler bir dizi işlemden geçirilerek sonraki alt adımlarda kullanılabilecek hale getirilmesi işlemleri gerçekleştirilir. Buradaki amaç gürültü gibi olumsuzluklardan kurtulmak ve işlem yükünü hafifletmektir.

Arka plan modellemesi aşamasında ön işlemeden geçmiş çerçeveler kullanılarak arka plan modeli oluşturulur ve güncellenir. Arka plan modellemesi ile arka planların istatistiksel tanımı yapılır.

Ön planın saptanması aşamasında ise video çerçevelerinde arka plan modeliyle tam olarak açıklanamayan pikseller bulunur ve ön plan maskesini oluşturacak bu pikseller ikili (x, y koordinatlarına göre) bileşenler halinde üretilir.

Veri doğrulaması aşamasında ise; önceki aşamalarda oluşturulan ön plan maskesi ele alınır. Bu maskede hareket eden gerçek nesnelerle uyuşmayan pikseller ayrıştırılır ve ön plan ayrımında kullanılacak maske elde edilir.

Kullanım alanının önemli ve yaygın olması dolayısıyla ön plan arka plan ayrımında farklı yöntemler geliştirilmiştir bu bölümde ön plan, arka plan ayrımında kullanılan temel algoritmalara değinilecektir. Bu algoritmaların OpenCV görüntü işleme aracındaki kullanımından bir sonraki bölümde bahsedilecektir.

3. 1. Arka Plan Modellemesı

Hareket eden ön plan nesnelerin tespitinde, var olan kare ile bir önceki karenin karşılaştırılması en kolay yoldur. Bu karşılaştırma sadece iki kare arasındaki farkı bulmayı sağlar. Hareket eden nesnelerin tutarlı şekilde bulunması işlemi ise ancak arka plan modeli oluşturulması ve bu modelin arka planda oluşan değişiklikler, ışık değişimi gibi etmenlere karşı modelin güncellenebilmesiyle mümkündür.

Arka plan modelleme konusunda uygulanabilen en temel varsayım, hareket etmeyen cisimler haricinde kalan görüntünün istatistiksel verileriyle modellenebilecek düzgün bir davranış sergilemesidir. Eğer bunu sağlayan bir model oluşturulursa, modele uymayan parçalar; sahne alanına giren, hareket eden nesneleri belirtir. Bu işlem Arka Plan Modelleme ya da Arka Plan olarak adlandırılmaktadır.

Arka planın etkili bir şekilde modellenmesi, eğitim için 10 – 30 saniye süresince video çerçevelerin istatistiksel değerlenin yorumlanması ile olur [2]. Bu süre boyunca hareketli

nesnelerin mümkün olduğunca az, hatta hiç olmaması gerekmektedir. Fakat gerçek durumlarda bu çoğu zaman mümkün değildir. Kalabalık alışveriş merkezlerindeki hareketli insanlar, sürekli trafik akışı olan yollar, rüzgârın salladığı ağaç yaprakları, akan sular değişen ışık yoğunlukları gibi dinamik arka planlar için arka plan modelleme çalışması yapılması gerekebilir. Bu yüzden modelin bu tarz durumlara karşı çözüm getirmesi gerekmektedir.

Bu bölümde arka plan modelleme yöntemleri özyinelemesiz, özyinelemeli iki grup halinde incelenecektir.

3. 1. 1. Özyinelemesiz Yöntemler

Özyinelemesiz yöntemler N tane video çerçevesini bir arabellekte depolar ve arabellekte bulunan piksel değerlerine bakarak arka plan tahmini yaparlar [14, 15, 16]. Özyinelemesiz yöntemler arabellek dışındaki piksel değerlerine bağımlı olmadığı için kolaylıkla yeni arka plan değerlerine uyarlanabilirler. Fakat bu yöntemlerin bellek tüketimi fazladır. En yaygın özyinelemesiz yöntemler çerçeve farkı, ortanca süzgeç, doğrusal tahmin edici süzgeç, parametrik olmayan model yöntemleridir.

3. 1. 1. 1. Çerçeve Farkı Yöntemi

Çerçeve farkı yöntemi *t-1* zamanındaki çerçeveyi *t* zamanındaki çerçeve için arka plan olarak kabul eder. Arka plan modellemedeki en basit yöntem olarak kabul edilir. Çerçeve farkı yönteminde *t* zamanındaki çerçeve *t-1* zamanındaki çerçeveden eşitlik 3.1 de olduğu gibi çıkarılır ve piksel değerindeki fark eşik değerinden (*T*) büyükse bu piksel arka plana aittir denir [17].

$$|\text{Çerçeve}_{t} - \text{Çerçeve}_{t-1}| > T \qquad (3.1)$$

Çerçeve farkı yöntemi algoritma ve uygulama olarak kolay olması avantajının yanı sıra işlem yükünün az olması ile bilinir. Değişen piksel değerlerinin kolaylıkla uyarlanabilir olması diğer bir avantajıdır. Arka plan sadece bir önceki çerçeveye bağlıdır ve bu çevresel etkilerden kaynaklanan gürültüden ve arka plan değişikliğinde adaptasyon açısından diğer yöntemlere göre daha iyi sonuçlar vermesini sağlar [14].

Çerçeve farkı yöntemi kolay olmasının yanında bir çok sorun içermektedir. Düzgün bir şekilde dağılmış yoğunluk değerine sahip yavaş hareket eden nesnelerin içindeki pikseller arka plan olarak algılanabilir ve bu problem açıklık problemi olarak bilinir. Hareket eden nesnelerin geçtiği güzergahta arkasında bıraktığı pikseller ön plan nesnesi olarak belirlenir ve bu problem

hayalet etkisi olarak bilinir. Bir nesnenin ön plan nesnesi olarak belirlenebilmesi için (1/fps) zaman farkını aşacak hızda sürekli hareket etmesi gerektiğinde bir problemdir. Eşik değeri olan (T) değerinin ortalama parlaklığa göre olan hasas ayarı yine yönteme ait bir problem olarak söylene bilir [17].

3. 1. 1. 2. Ortanca Süzgeç Yöntemi

Ortanca süzgeç yöntemi arabellekteki çerçevelerin her birindeki piksellerin ortancasını bulur ve bu değeri arka plan olarak kabul eder. Arka plan modeli oluşturulurken arka planın arabellekteki çerçevelerin yarısından fazla görüldüğü varsayılır. Yani arka plan görüntüde daha fazla görülür, ön plan nesneleri ise değişken surede ekranda kalır [18, 19]. Ortanca süzgeç yöntemi yaygın kullanılan yöntemlerden biridir.

Ortanca süzgeç yöntemi renkli video çerçeveleri üzerinde çalışırken doğru sonuç üretmez ve sorun oluşturur. Bu sorunu çözmek için medoid yöntemi geliştirilmiştir [19]. Renkli videolarda her rengin görüntü üzerindeki dağılımı farklı olabilmektedir. Bu durumda ortanca değer hesaplanamaz. Bunun yerine medoid hesaplanır. Medoid hesaplanırken çerçevelerdeki renkler ayrı bir küme olarak düşünülür ve kümelerin merkezleri hesaplanır. Arabelleğe yeni katılan her çerçeveden sonra kümelerin merkezleri yeniden hesaplanır. İşlem yükü ortanca süzgece göre fazladır. Ayrıca bu yöntemin bellek gereksinimi fazladır.

3. 1. 1. 3. Doğrusal tahmin edici süzgeç yöntemi

Arabellekteki çerçeveleri oluşturan piksel değerlerinin birlikte ne kadar değiştiklerinin ölçüsü olan kovaryans değeri hesaplanarak süzgeç katsayıları bulunur. Süzgeç katsayıları her bir çerçeve zamanında örneklem değerlerine göre hesaplanır.

Diğer yöntemlere göre bellek gereksinimi daha fazladır. İşlem yükü çok karmaşık olduğu için gerçek zaman uygulamalarında kullanılmaz. Arka plan çıkarımında sık rastlanan bir yöntem değildir [20].

3. 1. 1. 4. Parametrik olmayan model yöntemi

Parametrik olmayan model yönteminde arka plan tahmini yaparken piksel yoğunluk f(u) fonksiyonunu kullanır [21]. Piksel yoğunluk fonksiyonunun parametrik olmayan kestirimini oluşturmak için geçmiş bütün çerçevelere bakılır.

$$f(u) = \frac{1}{L} \sum_{i=1}^{L} K(u - u_i) \tag{3.2}$$

Eşitlik 3.2.'de f(u) fonksiyonu hesaplanırken L geçmiş çerçeve sayısını, K Gauss yoğunluk fonksiyonunu, u ise çerçevedeki x, y koordinatında bulunan pikseli gösterir. Bu yöntem video çerçevelerine uygulandığında iyi sonuç üretir; fakat arka plan modellemesinde ve ön plan belirlenmesinde işlem karmaşıklığı yüksektir. Çünkü her bir piksel için f(u) değeri hesaplanır.

3. 1. 2. Özyinelemeli yöntemler

Özyinelemeli yöntemler arka plan modeli oluştururken arabellek kullanmazlar. Arka plan modelini her girdi çerçevesine göre özyinelemeli olarak güncellerler [22, 23, 24].

Özyinelemesiz yöntemlerle kıyaslandığında özyinelemeli yöntemler daha az bellek gereksinimine sahiptirler. Yaklaşık ortanca süzgeç yöntemi, kalman süzgeci yöntemi, gauss fonksiyonları karışımı yöntemi, özarkaplan yöntemi yaygın kullanılan özyinelemeli yöntemlerdir.

3. 1. 2. 1. Yaklaşık ortanca süzgeç yöntemi

Özyinelemesiz ortanca süzgeç yönteminin etkinliğini gören McFarlane ve Schofield, özyinelemeli yeni bir etkin çalışan yöntem geliştirmişlerdir.

Yaklaşık ortanca süzgeç yönteminin çalışma prensibi; girdi olarak gelen pikselin değeri arka plan olarak kabul edilen piksel değerinden büyükse arka plan piksel değeri 1 artırılır ve aynı şekilde girdi olarak gelen pikselin değeri arka plan piksel değerinden küçükse arka plan değeri 1 azaltılır. Bu şekilde devam edilirse zamanla arka plan, girdi piksellerin yarısının değerinin arka plan değerinden büyük, yarısının da arka plan piksellerinin değerinden küçük olduğu bir tahmine yakınsar, yani arka plan yaklaşık olarak ortanca değer olur. Bu yöntemle elde edilecek sonuçlar, karmaşık yöntemlerle elde edilebilecek sonuçlara yakındır ve bu yöntemin işlemin karmaşıklığı ve bellek gereksinimi çerçeve farkı yönteminden çok farklı değildir [2].

3. 1. 2. 2. Kalman Süzgeci Yöntemi

Kalman süzgeci durum uzayı modeli ile gösterilen bir dinamik sistemde, modelin önceki bilgileriyle birlikte giriş ve çıkış bilgilerinden sistemin durumlarını tahmin edilebilen filtredir. Görüntünün piksel değerlerindeki değişmelerin neden olduğu Gauss gürültüsü altındaki doğrusal dinamik sistemleri incelemede geniş bir kullanım alanına sahiptir.

Arka plan modellemesi için kullanılan çok değişik sürümleri vardır [25, 26, 27]. Bu sürümler izlemede kullanılacak durum değişkenlerinin kapsadığı uzay bakımdan farklılık

gösterirler. En basit sürümü sadece parlaklık yoğunluğu değerini kullanırken Karmann ve von Brandt hem parlaklık yoğunluğunu hem de parlaklık yoğunluğunun zamana göre türevini kullanmışlardır. Koller, Weber ve Malik ise parlaklık yoğunluğunu ve parlaklık yoğunluğunun uzamsal türevini kullanmışlardır. Genel olarak durum değişkenleri uzayı aşağıdaki eşitlik yardımıyla hesaplanır.

$$xn = Axn\text{-}1 + K (un\text{-}1 - HAxn\text{-}1) \quad (3.3)$$

A içsel dinamiklerin oluşturduğu parametre matrisini, K Kalman kazanç matrisini ve H de ölçüm matrisini gösterir. Arka plan yoğunluk değeri ve zamana göre türevi özyinelemeli olarak güncellenir.

Kalman süzgeci yönteminde hareket eden nesnelerin hareket izleri yok edilememektedir. Bu da birçok uygulama için sorun oluşturduğundan çok fazla tercih edilmeyen bir yöntem olmuştur.

3. 1. 2. 3. Gauss fonksiyonları karışımı yöntemi (MoG)

Yüksek karmaşıklığa sahip yöntemler arasında literatürde iki yöntem baskın olarak görülür. Bunlardan biri Kalman süzgeç yöntemi diğeri ise Gauss karışımı (MoG) yöntemidir [14, 23, 28].

Gauss fonksiyonları karışımı yöntemi (MoG) birden fazla çevresel faktörlerin etkisini süzebildiğinden daha sağlam, daha doğru sonuçlar üretir. Örneğin gökyüzü ve sallanan yaprağın olduğu bir görüntüde iki farklı sekil vardır, gökyüzü ve yaprak. Kalman süzgeci yöntemi tek bir Gauss fonksiyonu kullandığı için tek bir sekli ayırabilir. Ya gökyüzünü ayırır ya da yaprağı ayırabilir, ikisini birden ayıramaz. MoG yönteminde ise birden fazla Gauss fonksiyonu kullanıldığı için her iki şekilde başarılı bir şekilde ayrıştırılır.

Gauss fonksiyonları karışımı yöntemi (MoG) arka plan renklerin açıklık, koyuluk değerlerinin oluşturduğu çerçeveler olarak kabul edilmez. Bunun yerine arka plan modeli parametriktir. Her pikselin yeri bir grup Gauss fonksiyonu tarafından belirlenir. Bu fonksiyonlar toplanarak olasılık dağılım fonksiyonu elde edilir.

Her bir Gauss fonksiyonunun ortalamasını, bir sonraki çerçevede gelecek olan piksel değerlerinin eğitim sonucu elde edilmiş tahmini olarak düşünülebilir. Bu işlem uygulanırken piksellerin genelde arka plana ait olduğu kabul edilir. Her bir piksel için genelde 3-5 Gauss fonksiyonu vardır. Fakat bu sayı bellek sınırlamasına göre değişebilir.

Bir pikselin arka plana ya da ön plana ait olduğunu belirleyebilmek için Gauss fonksiyonlarının ilgili pikseli izleyip izlemediğine bakarız. Eğer pikselin değeri arka plan

bileşeninin standart sapmasının ölçekleme faktörüyle çarpımından küçük veya eşitse bu piksel arka plan bileşenidir, diğer durumda ise ön plan bileşenidir.

MoG algoritması aşağıdaki adımlardan oluşur:

1- Girdi piksellerini, girdi piksellerine komşu piksellerin (bileşenlerin) ortalamasıyla karşılaştır. Eğer pikselin değeri bileşenlerin ortalamasına yakınsa bu bileşen uyumlu bileşen olarak kabul edilir. Yani bir bileşenin eşleşen bileşen olması için piksel ile bileşen ortalamasının farkının mutlak değerinin bileşenin standart sapmasının ölçekleme değeriyle çarpımından küçük olması gerekir.

2- Yeni pikselin değerini yansıtmak için bileşen değerlerini güncellenir.

3- Hangi bileşenlerin arka plana ait olduğunu belirlenir.

4- Ön plan piksellerini belirlenir. Arka plan modelinde tanımlanamayan pikseller ön plan pikselleridir.

3. 1. 2. 4. Öz arka plan yöntemi

Öz arka plan yönteminde video çerçevelerindeki pikselleri gruplamak için hareket kullanılır; çünkü arka plan sabit olarak kabul edilir. Hareket eden nesnelerin bulunabilmesi için arka planı modelleyen öz uzay oluşturulur. Öz uzay gün içinde değişen ışık çeşitlerini ve hava koşullarını da tanımlar. Öz uzay basit standart bilgisayar grafik teknikleri kullanılarak oluşturulur. [15, 28, 16].

3. 1. 2. 5. Kod Çizelgesi (CodeBook) Yöntemi

Kod Çizelgesi (CodeBook) yöntemi ön plan arka plan ayrımı yaparken arka plan modelini kullanan yöntemdir. Diğer yöntemlerle kıyaslandığında bellek kullanımı ve hız bakımından daha verimlidir [29].

Kod çizelgesi yöntemi uzun zaman boyunca hareketli ve hareketsiz arka plan nesnelerini sınırlı bellek altında modelleyebilen, kendi kendini güncelleyebilen, dinamik arka planları modelleyebilen bir yöntemdir. Arka plan modelini güncelleyebildiğinden dolayı değişen ışık yoğunlukları ile uyumlu çalışabilmektedir. Arka plan modeli oluştururken görüntüde ön plan nesnesi olmasına izin verir. Kod çizelgesini olusturan temel bileşenlere kod sözcükleri denir.

Kod çizelgesi algoritması uzun video görüntülerinden bir arka plan modeli oluşturmak için bir niceleme / kümeleme tekniğini benimser. Her bir piksel için, bir veya daha fazla kod sözcüklerinden oluşan bir kod çizelgesi oluşturulur. Her pikselin örnekleri parlaklık sınırları ile birlikte metrik bir renk sapmasına dayalı kod sözcükleri kümesi içine kümelenmiştir. Her piksel aynı sayıda kod sözcüğüne sahip değildir. Kod sözcükleri ile temsil edilen kümeler tek

Gaussian veya diğer parametrik dağılımlara karşılık gelmek zorunda değildir. Bir piksel dağılımı tek bir normal olsa bile, o piksel için birkaç codewords söz konusu olabilir. Arka plan piksel-piksel olarak kodlanmıştır.

Ön plan arka plan ayrımı, renk ve parlaklık farklılıkları aracılığı ile arka plan modeli ile mevcut görüntü arasındaki fark karşılaştırılarak yapılır. İşlem yapılan pikselin arka plan olabilmesi için iki şart vardır. Birincisi pikselin renk sapması kod sözcüklerinden bazılarının renk sapması algılama eşiğinden daha az olmasıdır. İkincisi ise pikselin parlaklık değerinin bu kod sözcüğünün parlaklık aralığında yer almasıdır. Bu şartlar sağlandığı taktirde piksel arkaplana aittir. Aksi takdirde ön plan nesnesine ait bir pikseldir.

İlk Kod Sözcüğünün Oluşturulması:

Algoritma renkli görüntüler için açıklanmıştır ama aynı zamanda, küçük değişikliklerle gri ölçekli görüntülerde kullanılabilir. X dizisi tek bir pikselin aldığı N adet RGB değeri için oluşan bir eğitim dizisi olsun $X= \{ x_1; x_2; \ldots ; x_N\}$. $C=\{c1; c2; \ldots ; cL\}$ L kadar kod sözcüğünü kapsayan bir pikselin kod çizelgesi olsun. Her pikselin alabileceği farklı değerlere bağlı olarak kod sözcükleri sayısı dolayısıyla kod çizelgeleri boyutları değişir.

Her kod sözcüğü $ci;\ i = 1 \ldots L;$ bir RGB vektöründen $vi = (R_i ; G_i ; B_i)$ ve 6 adet değişkenden $aux_i=\{ I_{(min)i} , I_{(max)i}, f_i, n_i, p_i, q_i \}$ oluşur. Değişken grubu aux_i geçici parlaklık ve yoğunluk değerleri içerir ve aşağıda açıklanmıştır.

$I(min)_i$,$I(max)_i$; Kod çizelgesine atanmış tüm piksellerin tek tek alınan parlaklık değerlerinin en küçük ve en büyük değerleri göstermektedir. f_i; *k*od sözcüğüne erişim sıklığını gösteren değerdir. n_i kod sözcüğüne erişilmeyen en uzun süreyi gösteren değişkendir. Bu değişken kullanılarak kod çizelgesindeki gereksiz kod sözcükleri atılır. p: Kod sözcüğüne ilk erişim zamanı gosteren değiskendir**.** *q:* Kod sözcüğüne son erisim zamanını gösteren değişkendir.

Eğitim süresinde t zamanında örneklenen her bir RGB değeri (X_t) kod çizelgesinde var olan hangi kod sözcüğü (c_m) ile eşleştiğini belirlemek için kod çizelgesindeki kod sözcükleri ile karşılaştırılır (m değeri kod sözcüğünün eşleşen index değeridir.). Eşlenen kod sözcüğü yaklaşık örnek değer olarak kullanılır. Hangi kod sözcüğünün en iyi eşleşeceğini bulmak için pikselin kod sözcüğüne işlenmiş renk sapma değeri ve parlaklık aralığına bakılır.

Kod Çizelgesi Oluşturma Algoritması:

I. $L \leftarrow 0$, $C \leftarrow \emptyset$ L (kod sözcüğü sayısına (L) 0 atanır, kod çizelgesi (C) ise boş küme olarak ayarlanır .)

II. Döngü başlangıcı t=1 den N'ye tekrarla (t çalışılan görüntü indeksi N çalışılacak görüntü sayısı)

(i). $x_t = (R,G,B),\ I \longleftarrow \sqrt{R^2 + G^2 + B^2}$ (x_t görüntüsündeki bir pikselin RGB değerleri alınır ve Parlaklık bilgisi (I) R, G, B değerlerine göre hesaplanır.)

(ii). Kod çizelgesinde $(C= \{c_i \mid 1 \leq i \leq L\})$ hangi indeksteki kod sözcüğünün (C_m) x_t ile eşleştiğini (a) ve (b) şartlarına göre bul

(a) x_t görüntüsündeki renk değeri ile kod sözcüğündeki renk değeri farkı eşik değerinden (ε_1) küçük veya eşik değerine eşit ise $(\delta (x_t, v_m) \leq \varepsilon_1)$

(b) Parlaklık değeri x_t görüntüsündeki pikselin parlaklık değeri (I) kod çizelgesinin m indeksindeki kod sözcüğünün elemanlarından $I_{(min)m}$, $I_{(max)m}$ değerleri arasında ise (parlaklık $(I, (I_{(min)m}, I_{(max)m})) = true$)

(iii). Eğer kod çizelgesi boş küme ise (C= Ø) veya kod sözcükleri arasında eşleşme yok ise L değerini 1 arttır (L=L+1) ve yeni bir kod sözcüğünü (C_{L+1}) aşağıdaki adımları takip ederek oluştur.

- x_t görüntüsündeki pikselin RGB değerlerini kod sözcüğündeki RGB değerlerine al (V_{L+1}= (R, G, B))
- x_t görüntüsündeki pikselin sırasıyla minimum parlaklık $I_{(min)t}$, maksimum parlaklık $I_{(max)t}$, f_t (erişim sıklığı), n_t (erişilmeyen en uzun süre) , p_t (ilk erişim zamanı), q_t (son erişim zamanı) bilgilerini kod sözcüğüne al (aux $_{L+1}$={ I,I,1,t-1, t, t })

(iv). Kod çizelgesinde x_t ile eşleşen m indeksli kod sözcüğü (c_m) var ise aşağıdaki denklemlerle güncellenir.

m indeksli kod kelimesi V_m= (R_m, G_m, B_m) ve aux_m ={ $I_{(min)m}$, $I_{(max)m}$, f_m, n_m, p_m, q_m } vektörlerinden oluşur. Ve elemanları aşağıdaki gibi güncellenir.

- $V_m \longleftarrow \left(\frac{fm*Rm+R}{fm+1}, \frac{fm*Gm+G}{fm+1}, \frac{fm*Bm+B}{fm+1}\right)$

  m indeksli kod kelimesinin RGB değerleri bu denkleme göre güncellenir.
- $aux_m \longleftarrow (min\ \{I, I_{(min)m}\},\ max\ \{I, I_{(min)m}\},\ f_m + 1,\ max\ \{ n_m , t - q_m \},\ p_m,\ t\ ).$

Döngü Sonu

III. Kod çizelgesindeki her bir kod sözcüğü $(c_i,\ i = 1, \ldots L,)$ erişilmeyen en uzun süre, ilk erişim zamanı ve son erişim zamanı güncellemesi etrafında döner.

$n_i \longleftarrow max\ \{ n_i , (N - q_i, pi - 1) \},$

Denklem 2 ve 3 de detaylı olarak anlatılacak olan Adım II'deki iki koşul (a) ve (b) sağlandığında x_t ve c_m nin saf renkleri yeteri kadar yakındır ve x_t ve c_m'ni kabul edile bilir parlaklık sınırında yer almaktadır. ε_1 eşik değeri örneğidir. Algoritmayı güçlendirmenin bir yolu en son güncellenen kod sözcüğünü kod çizelgesinin en başına koymaktır. Çoğu zaman eşleşen kod sözcüğü bu yer değişmeden dolayı kod çizelgesinin ilk kod sözcüğü olur ve bu da eşleştirme basamağında etkili olur.

Gereksiz Kod Sözcüklerinin Çıkarılması:

Kod çizelgesi yöntemi arka plan modelini oluştururken görüntülerde ön plan nesneleri de bulunabilir. Bunu işlem gereksiz kod sözcüklerinin kod çizelgesinden çıkarılması ile gerçekleşir. Bu aşamada ön plan nesneleri için ya da gürültü için oluşturulan kod sözcükleri kod çizelgesinden çıkarılarak arka plan nesnelerine ait kod sözcüklerini içeren kod çizelgesi elde edilmektedir. Gereksiz kod sözcükleri çıkarılmadan ilk elde edilen kod çizelgesine şişman kod çizelgesi denir. Arka plana ait piksellere erişim sıklığı ön plan piksellerine erişim sıklığından daha fazladır. Şişman kod çizelgesindeki ön plan nesnelerine ait ve gürültüye ait kod sözcüklerini çıkarabilmek için (3,4) numaralı eşitlikten faydalanılır. M gereksiz kod sözcükleri temizlendikten sonra elde edilen kod çizelgesini gösterir.

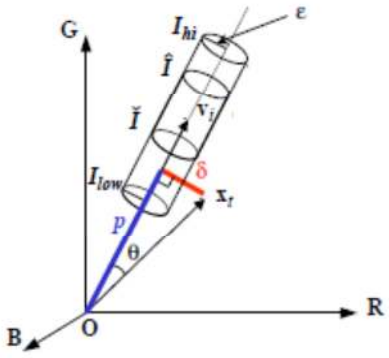
$$M= \{c_m | c_m \varepsilon C \square\ n_m \leq T_M\} \qquad (3.4)$$

(3.4) numaralı eşitlikte, T_M arka plan modeli oluşturulurken kullanılan çerçeve sayısının yarısı olarak kabul edilir. n_m daha önce bahsedildiği gibi kod sözcüğüne erişilmeyen en uzun süreyi gösterir. Bu durumda bir kod sözcüğüne model oluşturma aşamasındaki çerçeve sayısının yarısı kadar bir süre erişilmezse o kod sözcüğü kod çizelgesinden atılır. Yani arka plan kod sözcüklerine erişim sayısı model oluşturma aşamasında kullanılan çerçeve sayısının en az yarısı kadardır.

Renk ve Parlaklık:

Gölgeler ve vurgular gibi küresel ve yerel aydınlatma değişiklikleri ile başa çıkmak için, algoritmalar genellikle normalize renk (renk oranları) kullanır. Bu teknikler genellikle görüntünün karanlık alanlarda kötü çalışır. Karanlık piksel aydınlık piksellere göre yüksek belirsizliğe sahiptir çünkü renk oranı belirsizliği parlaklık ile ilişkilidir. Parlaklık renk oranları karşılaştırarak bir faktör olarak kullanılmalıdır. Bu belirsizlik karanlık bölgelerdeki algılamayı değişken yapar. Yanlış algılamalar karanlık bölgeler etrafında kümelenmiş olma eğilimindedir.

Bu nedenle, aydınlatma değişimi altında piksel değerlerinin zaman içinde nasıl değiştiği gözlenmiştir. Renk ve parlaklık değişimi için ayrı bir değerlendirme mevcuttur. Kod çizelgesi algoritmasının 2. adımındaki (a) renk değişimi ve (b) parlaklık bilgisini hesaplayabilmek için şekil 3.1.'den faydalanır [29]. Şekil 3.1.'e göre oluşturulan renk değişimi ve mantıksal parlaklık fonksiyonları (3,5) ve (3,6) numaralı eşitliklerde verilmiştir.



Şekil 3.2. Kod çizelgesinde renk değişimi ve parlaklık hesaplama modeli [29].

,

,

Renk farkı şu şekilde hesaplanır. (3.5)

——— ,

———————.

Parlaklık (I, ($I_{(min)}$,$I_{(max)}$)) (3.6)

(3.6) numaralı eşitlikte parlaklığı belirli bir aralıkta tutmak için, yöntemin ani ışık değişikliklerinden ve gölgelerden etkilenmemesini sağlamak için I alçak ve I yüksek değerleri kullanılır. Bu değerleri hesaplarken (3.7) ve (3.8) numaralı eşitliklerden faydalanılır. α , β parlaklık bilgisini belirli aralıkta tutabilmek için kullanılacak değişkenlerdir. Bu değişkenlerin alabileceği değer aralığı $0.4 \leq \alpha \leq 0.7$ ve $1.1 \leq \beta \leq 1.5$ seklindedir. Bu değişkenlerin alabileceği değerler deneysel çalımsalar sonucu elde edilmişlerdir [29].

(3.7)

——— (3.8)

Kod çizelgesi yönteminde ön plan arka plan ayrımı:

Ön plan arka plan ayrımı aşaması arka plan modeli oluşturma aşamasına benzerdir. Bu aşamada daha önce elde edilen kod çizelgesi kullanılarak gelen piksellerin ön plan pikseli ya da arka plan pikseli olduğuna karar verilir. Arka plan çıkarmada kullanılan algoritma (BGS) şöyledir:

I. x = (R,G,B), I← $\sqrt{R^2 + G^2 + B^2}$

II. (5.5) numaralı eşitlikle elde edilen M kod çizelgesindeki her bir kod sözcüğü için aşağıdaki şartları x 'e göre sağlayan m c kod sözcüğünü bul

- Renkdeğişimi (x, cm) $\leq \varepsilon_2$
- Parlaklık (I, ($I_{(min)}$,$I_{(max)}$)) = doğru

Kod çizelgesi oluşturma algoritmasının ikinci aşamasındaki (iv) adımındaki gibi,

Eşlesen kod sözcüğünü güncelle

III. BGS (x) $\begin{cases} eşleşme\ yoksa, & \textbf{önplan} \\ eşleşme\ varsa, & \textbf{arkaplan} \end{cases}$

Burada ε_2 ön plan arka plan ayrımında kullanılan eşik değerini, x gelen pikseli gösterir. Gelen piksel için oluşturulan kod çizelgesinde eşleşen herhangi bir kod sözcüğü bulunamazsa gelen piksel ön plan pikseli olarak işaretlenir, eşleşen kod sözcüğü bulunursa gelen piksel arka plan pikseli olarak işaretlenir.

Kod çizelgesi yönteminde arka plan modelinin güncellenmesi

Görüntü alınan yerdeki arka plan değişebileceği için arka plan modelinin değişen şartlara göre güncellenmesi gerekir. Örneğin park alanındaki araçlar park alanından çıkabilirler. Bu durumda kod çizelgesindeki park alanından ayrılan araçlara ait kod sözcüklerinin kod çizelgesinden çıkarılması gerekir. Kod çizelgesi yöntemi arka plan modelini güncelleyebilmek için ara kod çizelgesi H'ı ve T_H, T_{ekleme} ve $T_{çıkarma}$ şeklinde 3 değişken kullanır. T_H değişkenine bakılarak geçici kod çizelgesindeki kod sözcüklerinin kod çizelgesinden çıkarılıp çıkarılmayacağına karar verilir. T_{ekleme} değişkenine bakılarak geçici kod çizelgesinden ana kod çizelgesine taşınacak kod sözcüklerine karar verilir. $T_{çıkarma}$ değişkenine bakılarak ana kod çizelgesindeki silinecek kod sözcüklerine karar verilir. Ara kod çizelgesinde yeterince uzun süre bekleyen kod sözcükleri ana kod çizelgesine taşınır ve ana kod çizelgesinde uzun süre erişilmeyen kod sözcükleri ana kod çizelgesinden çıkarılır [29]. Kod çizelgesi yöntemi arka plan modelini güncellerken aşağıdaki gibi bir algoritma kullanır:

I. Arka plan modeli oluşturma aşamasında M ana kod çizelgesi elde edildikten sonra yeni bir H ara kod çizelgesi oluştur.

II. Gelen pikseli için eşleşebilecek kod sözcüğünü M'de ara. Bulunursa kod sözcüğünü güncelle.

III. Eşleşen kod sözcüğü bulunamazsa eşleşebilecek kod sözcüğünü H'de ara. Bulunursa kod sözcüğünü güncelle. Bulunamazsa yeni bir h kod sözcüğü oluştur ve H'ye ekle.

IV. T_H'ı kullanarak H kod çizelgesindeki gereksiz kod sözcüklerini temizle.

$$H \leftarrow H - \{h_i \mid h_i \in H \wedge \lambda_i \geq T_H\}$$

V. Ara kod çizelgesi H'de yeterince uzun bekleyen kod sözcüklerini ana kod çizelgesi M'ye taşı.

$$M \leftarrow M \cup \{h_i \mid h_i \in H, h_i \text{ bekleme süresi} > T_{ekleme}\}$$

VI. M'de uzun süre erişilmeyen kod sözcüklerini M'den çıkar.

$$M \leftarrow M - \{c_i \mid c_i \in M, c_i \text{' ye erişilmeyen süre} \geq T_{çıkarma}\}$$

Bu algoritma işletilerek arka plan modelinin güncellenmesi sağlanır.

4. OPENCV

Bilgisayarlı görü ve gerçek zamanlı görüntü işleme işlemci gücü isteyen uygulamalardır. Bu durumun öneminin farkında olan işlemci firması Intel gerçek zamanlı uygulamaları hedef alarak "OpenCV" isimli proje ortaya çıkarmıştır.

OpenCV, Intel tarafından geliştirilerek BSD lisansı ile lisanslanmış, "Bilgisayarla Görü/Görme" kütüphanesidir. Özellikle gerçek zamanlı uygulamalar hedef alınarak geliştirilmiş olması, ticari kullanımı dâhil ücretsiz olması ve kütüphaneyi diğer görüntü işleme araçlarından bir adım öne çıkarmaktadır.

OpenCV Massachusetts Teknoloji Enstitüsü (MIT) ve bazı diğer üniversitelerde geliştirilmeye başlanan temel görüntü işleme kütüphaneleri, İntel firmasının da desteğiyle geliştirilen açık kaynaklı C ve C++ dilleri ile yazılmış bir görüntü işleme aracıdır. Phyton Matlab, Ruby Java (JavaVis) gibi diğer diller üzerinde de aktif geliştirmeler mevcuttur.

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 [2]. Akademik ve ticari kullanımı ücretsiz olan bu kütüphane Windows, Linux, MacOS X gibi farklı platformlarda kullanılabilir [3].

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.

4. 1. Opencv'nin Temel Özellikleri

OpenCV (Open Source Computer Vision Library) Windows, Linux, Mac OS X, PSP (PlayStation Portable), IOS, android ve birçok gömülü ve mobil sistem platformları üzerinde çalışabilen, C ve C++ dilleriyle yazılmış, gerçek zamanlı bilgisayarla görme (real time computer vision) ve görüntü işleme (image processing) uygulamaları için kullanılabilen, açık kaynak kodlu bir kütüphanedir.

OpenCV, birçok deneyimli yazılımcının çalıştığı Willow Garage şirketi tarafından desteklenmektedir. OpenCV görüntü işleme aracı bir çok açık kaynak yazılımına ev sahipliği yapan SourceForge sitesinden temin edilebilir. İçerdiği fonksiyonların birçoğu platformdan bağımsız olarak çalışır. 2.0 versiyonundan itibaren, C ara yüzüne ek olarak C++ ara yüzü de eklenmiştir.

Günümüzde, OpenCV içerisindeki bilgisayarla görme ve görüntü işleme algoritmaları kullanılarak;

- İnsan-Bilgisayar Etkileşimi (Human-Computer Interaction – HCI)
- Nesne Kimliklendirme, Bölümleme ve Tanıma (Object Identification, Segmentation and Recognition)
- Yüz Tanıma (FaceRecognition)
- İşaret Dili Tanıma (GestureRecognition)
- Hareket Yakalama, Algılama ve Takibi (Motion Tracking, Ego Motion, Motion Understanding)
- Çiftli ve Çoklu Kamera Kalibrasyonu ve Derinlik Hesaplama (Stereo and Multi-Camera Calibration and Depth Computation)
- Hareketli Robot Teknolojileri (Mobile Robotics)

Uygulamaları geliştirilebilir.

Kütüphane C ve C++ ile yazılmış olup Linux, Windows ve Mac OS X 'de uygulama geliştirmeye elverişlidir. Phyton, Rubby Java, C# gibi diğer diller için ara yüzlerin yazımı aktif olarak devam etmektedir.

OpenCV gerçek zamanlı uygulamalarda yeterli hız ve verimlilikte, çok çekirdekli işlemcilerin özelliklerinden yararlanabilecek şekilde optimize edilmiştir ve kütüphanenin geliştirilmesine devam edilmektedir. Arzu edildiği takdirde İntel tarafından kendi işlemcileri için özel olarak optimize edilmiş IPP (İntel İntegrated Performance Primitives) kütüphaneleri kullanılabilir. IPP kütüphanesi çeşitli algoritmaların daha performanslı

çalışması için düşük seviye optimize edilmiş çeşitli yordamlar içerir. IPP kurulduğu takdirde OpenCV tarafından otomatik olarak tespit edilip çalışma zamanında kullanılacaktır. Eğer proje ticari olarak geliştirilmiyorsa İntel bu kütüphaneyi Linux kullanıcıları için ücretsiz olarak sağlamaktadır.

OpenCV robotik, tıbbi görüntüleme, güvenlik gibi pek çok alanda görüntü işleme için kullanılabilen birçok yüksek seviyeli ve temel fonksiyon içermektedir. Bir alt kütüphanesi olan makine öğrenme kütüphanesinde (*. mll) ise örüntü tanıma ve kümeleme gibi görüntü işlemede sıklıkla kullanılan işlemler için fonksiyon setleri yer almaktadır.

OpenCV lisansında belirtildiği üzere kütüphanesinin bir parçası ya da tamamı, ticari ürünlerde ürünün kodları açılmaksızın kullanılabilir. Sahip olduğu bu lisans sayesinde aralarında IBM, Microsoft, Sony, Google gibi şirketlerden ve Stanford, Massachusetts Teknoloji Enstitüsü (MIT) gibi araştırma merkezlerinden araştırmacıların da bulunduğu geniş bir geliştirici topluluğuna sahiptir.

OpenCV, içerdiği işlevler sayesinde, endüstriyel ürünlerden güvenlik ürünlerine, oyun konsollarından mobese kameralarına, yapay zekâ ürünlerinden medikal ürünlerine, belge işlemeden astronomiye birçok görüntü işleme uygulamasının geliştirilmesine katkıda bulunmaktadır [1] .

4. 2. Opencv İle Neler Yapılabilir

Opencv ile aşağıda belirtilen işlemler yapılabilir;

- ***Görüntü ve Video Giriş/Çıkış işlemleri***

  Bu özellik sayesinde OpenCV ile rahatlıkla bilgisayardaki bir resmi okuyabilir, bir web cam den görüntü alabilir ya da resim ve video dosyaları oluşturabiliriz.

- ***Bilgisayarla Görüş (Computer Vision) ve Görüntü İşleme (İmage Processing) Algoritmaları***

  OpenCV kütüphanesindeki hazır fonksiyonlar ile temel görüntü işleme algoritmaları için yeniden fonksiyonlar oluşturmamıza gerek yoktur. Hazır fonksiyonlar sayesinde hem zamandan tasarruf ederken hem de kodlarımızı daha kısa ve anlaşılabilir bir biçimde yazabiliriz.

•***İleri Düzey Görüntü İşleme Özellikleri***

OpenCV ile aynı zamanda hareket tespiti, yüz algılama ve tanıma, kamera kalibrasyonu gibi ileri düzey görüntü işleme uygulamaları da yapılabilmektedir. OpenCV 'nin ayrıca bu işler için hazır API 'leri de (uygulamaları) bulunmaktadır.

•***Yapay Zekâ ve Otomatik Öğrenme Yöntemleri***

Bilgisayarla görme uygulamalarında çoğu zaman otomatik öğrenme ya da diğer yapay zekâ yöntemleri kullanılmaktadır. Bunlardan bazıları OpenCV 'nin Machine Learning paketinde bulunmaktadır.

•***Binary Görüntülerin Yaratılması ve İncelenmesi***

Şekil farklılıklarını bulma ve parça sayımı gibi sistemlerde kullanılan binary görüntüleri OpenCV ile oluşturabiliriz.

•***3B Görüntülerin Hesaplanması***

Farklı kameralardan alınmış görüntüler ile bir nesnenin yerinin belirlenmesi gibi uygulamalar da kullanılabiliriz.

•***Sık Kullanılan Matematiksel Denklemler***

OpenCV kütüphanesinin görüntü işlemede sık kullanılan lineer cebir, istatistik ve geometrik denklemler için hazır fonksiyonları vardır.

•***Grafik Veriler ile İlgili İşlemler***

OpenCV ile resimlerin üzerine yazı yazabilir ya da şekiller çizebiliriz. Bununla hayal gücümüzü zorlayarak birçok şey yapabileceğimiz gibi kameradan aldığımız görüntüleri işaretleme ya da etiketleme gibi işlemler de yapabiliriz.

•***GUI İşlemleri***

OpenCV ile resimleri göstermek için pencere oluşturabilir, fare ya da klavyeden verilecek komutları yakalayabiliriz.

•***Veri Yapıları ve Algoritmalar***

OpenCV ile dizileri ve resimleri kaydedip, onlar üzerinde hızlı ve verimli bir biçimde aramalar yapabiliriz.

•***Veri Sürekliliği***

Bilgisayara kaydettiğimiz verilere daha sonradan tekrar erişebiliriz.

4. 3. Opencv Bileşenleri

OpenCV kütüphanesi 5 temel bileşenden oluşmaktadır;

- CV bileşeni
- MLL bileşeni
- HighGUI bileşeni
- CXCore bileşeni
- CvAux bileşeni

Şekil 4.1. OpenCV Kütüphanesini oluşturan bileşenler

4. 3. 1. CV Bileşeni

Temel resim işleme fonksiyonları ve Bilgisayarla Görü/Görme için kullanılan yüksek seviyeli algoritmaları bünyesinde barındıran beş temel kütüphaneden biridir.

4. 3. 2. MLL Bileşeni

Makina öğrenmesi dalı için gerekli istatistiksel verilere ulaşmak, mevcut verileri sınıflandırmak için kullanılan fonksiyonları/araçları içerendiğer bir kütüphanedir.

4. 3. 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 [11].

4. 3. 4. CXCore Bileşeni

OpenCV kütüphanesine ait çeşitli veri yapılarını (cvPoint, cvSize, IplImage, cvHistogram, cvMat) bünyesinde barındıran, XML desteği de sağlayan bir kütüphanedir.

4. 3. 5. CvAux Bileşeni,

Pek çok deneysel algoritmaları barındırır.

- Şablon eşleştirme (template-matching),
- Sekil eşleştirme (shape matching),
- Yüz tanıma (face-recognition),
- Ağız hareketleri izleme (mouth-tracking),
- Vücut hareketlerini tanıma (gesture recognition)
- Kamera kalibrasyonu

4. 4. Opencv Kurulumu Ve Kullanımı

Microsoft Visual Studio 2010'da OpenCv kullanmayı bir örnek proje oluşturarak resimlerle anlatacaktır.

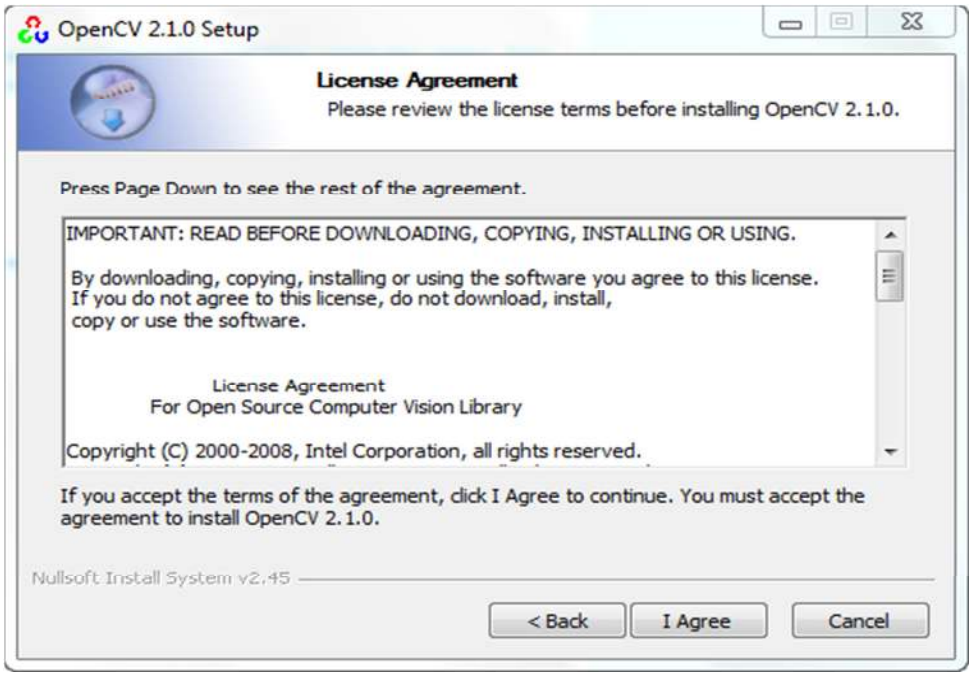
1.Adım : OpenCV kurulumu:

OpenCv 2.1 sürümünü http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.1/OpenCV-2.1.0-win32-vs2008.exe/download linkinden indirebilirsiniz

İndirilen .exe dosyasını çift tıklayınız.Ekrana Şekil 4.2'de ki resim gelecektir.

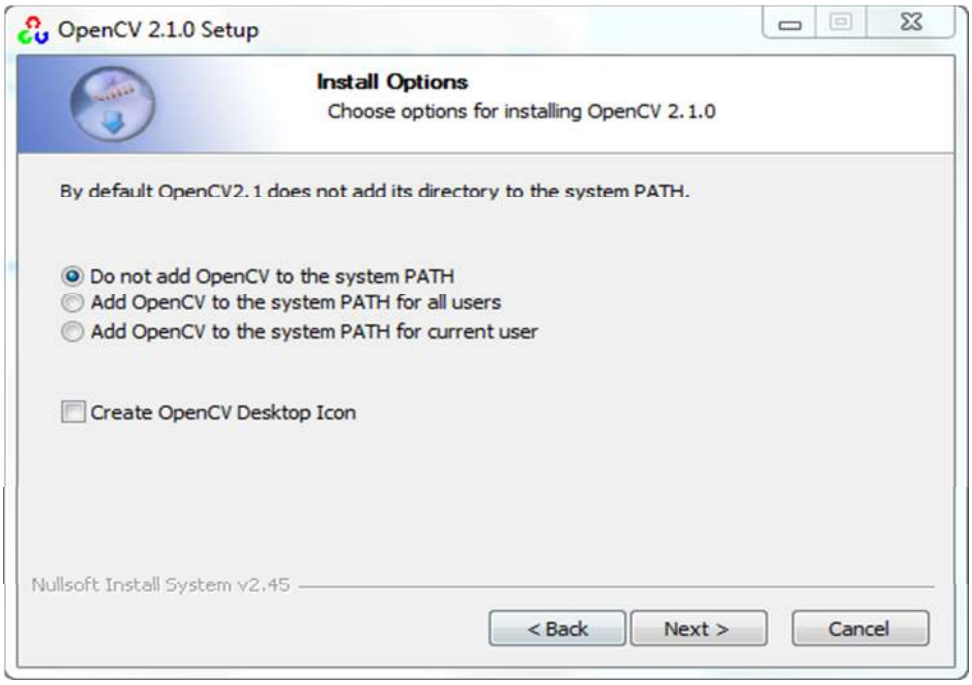
Şekil 4.2. OpenCV kurulum sihirbazı

Next butonuna tıklayınız. Şekil 4.3.'deki gibi bir ekran sizi karşılayacaktır



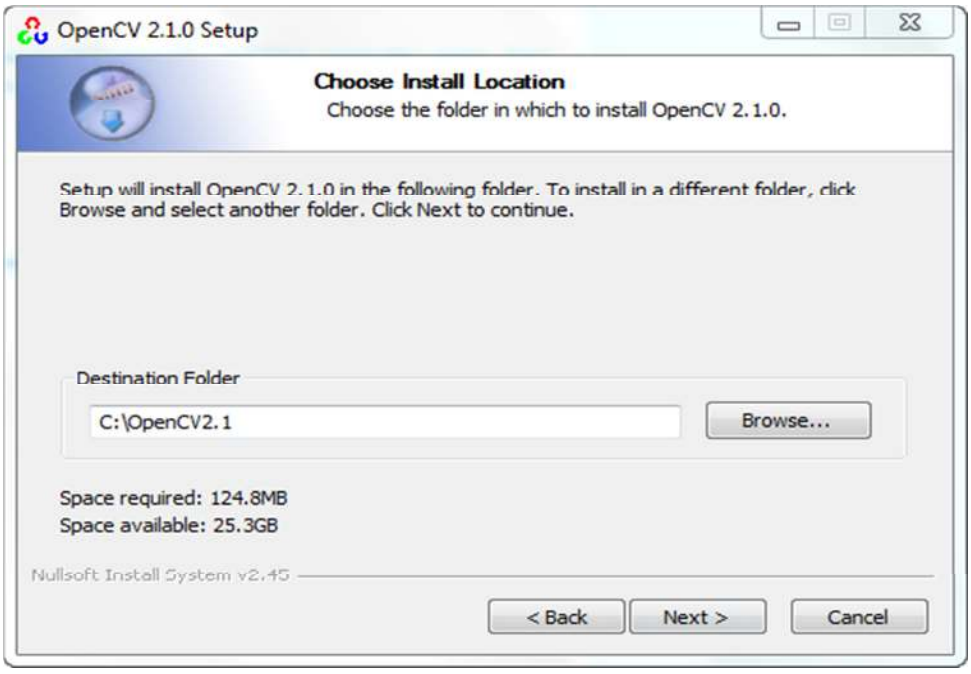
Şekil 4.3. OpenCV kurulum sihirbazı lisans sözleşmesi

I Agree butonuna tıklayınız. Şekil 4.4. deki ekran sizi karşılayacaktır.



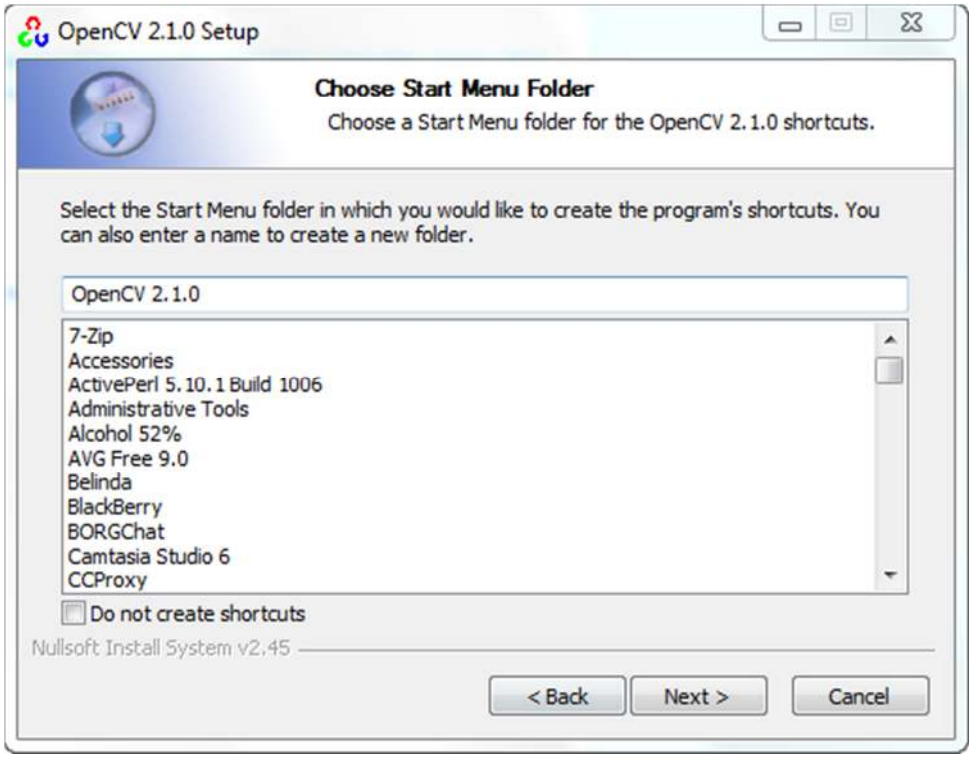
Şekil 4.4. OpenCV kurulum sihirbazı kurulum yolu tanımlama

Do not add OpenCV to the system PATH'i seçiniz ve Next butonuna tıklayınız. Şekil 4.5.'daki gibi bir ekran sizi karşılacaktır.



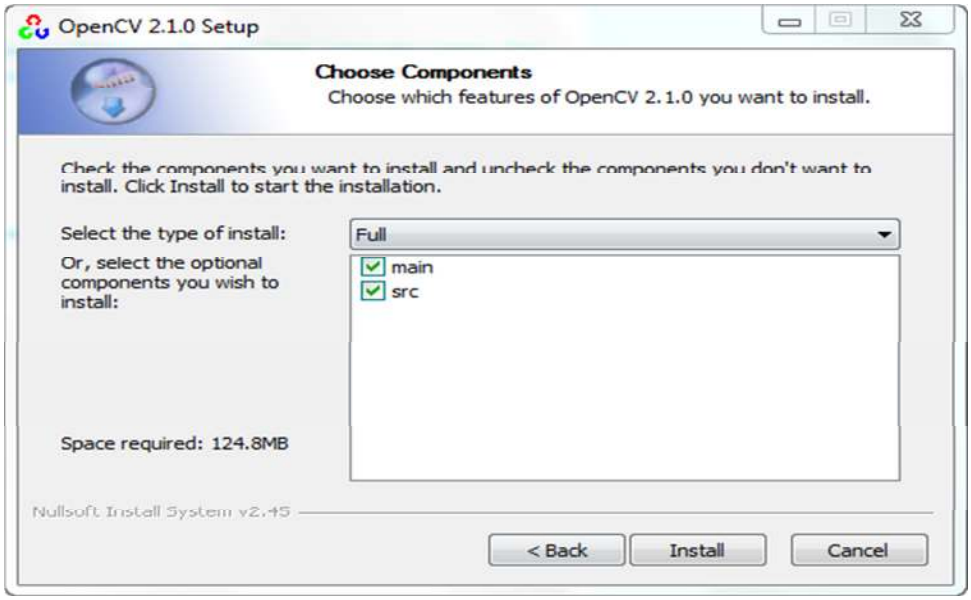
Şekil 4.5. OpenCV kurulum sihirbazı kurulum dosyası belirleme

Destnation Folder ile kurulum yapacağınız dizini değiştirebilirsiniz. Resimde varsayılan yol kabul edilmiştir. **Next** butonuna tıkladığınızda sizi Şekil 4.6. karşılayacaktır.



Şekil 4.6. OpenCV kurulum sihirbazı

Next butonuna tıklayınız. Sizi Şekil 4.7. ekranı karşılayacaktır



Şekil 4.7. OpenCV kurulum sihirbazı

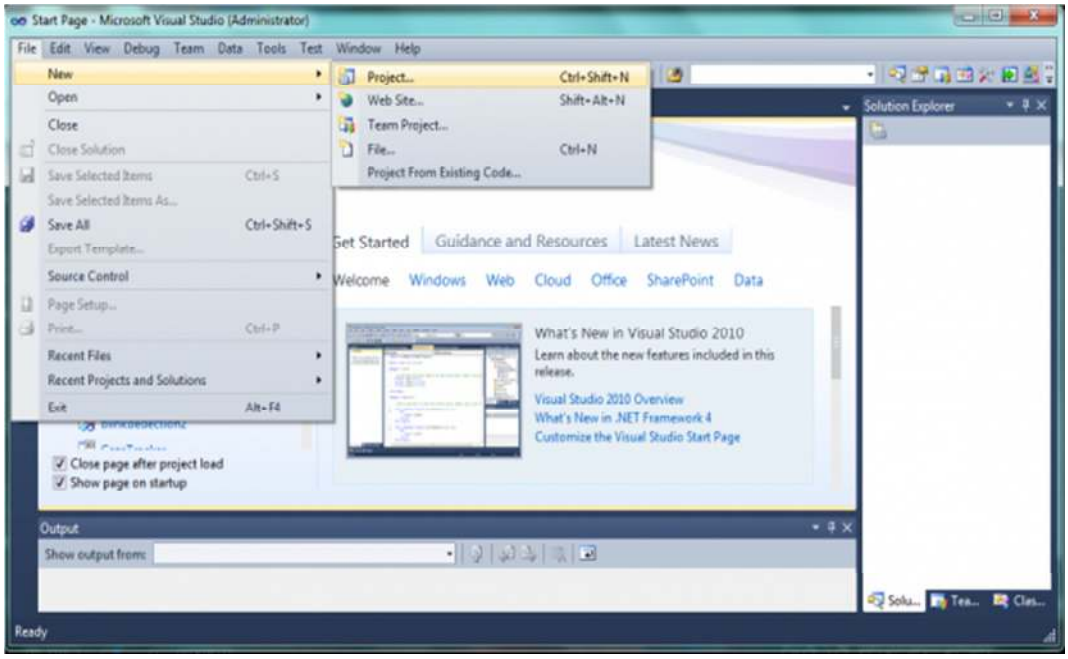
Full kurulumu seçiniz ve Install butona tıklayınız. OpenCv kuruluma başlayacaktır. Kurulum sonrasında sizi Şekil 4.8. deki gibi bir ekran karşılayacaktır.

Şekil 4.8. OpenCV kurulum sihirbazı

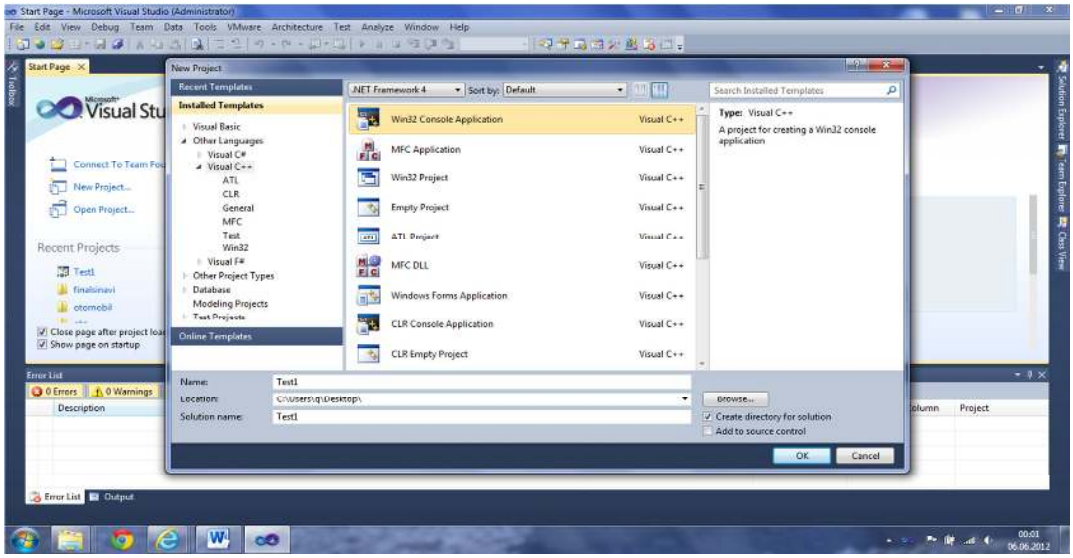
Kurulumu tamamlamak için **Finish** buttona tıklayınız.

2.Adım: Visual Studio 2010 Projesini Ayarlama

Visual Studio 2010'da OpenCV ayarlarını yapabilmek için Şekil 4.9.'da olduğu gibi C/C++ projesi oluşturunuz. Projenize Şekil 4.10 daki gibi isim ve kayıt yeri belirtiniz.

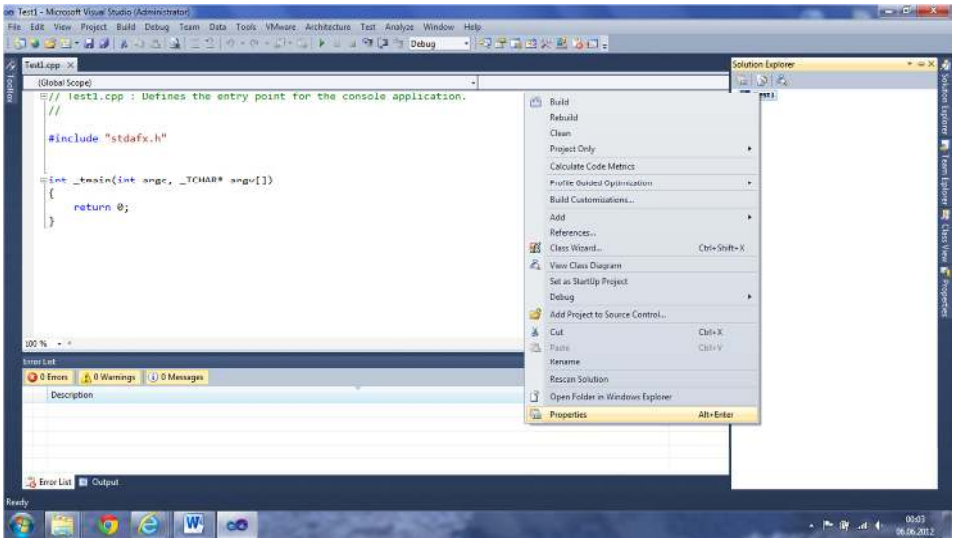


Şekil 4.9. Visual Studio 2010'da C/C++ projesi oluşturma.



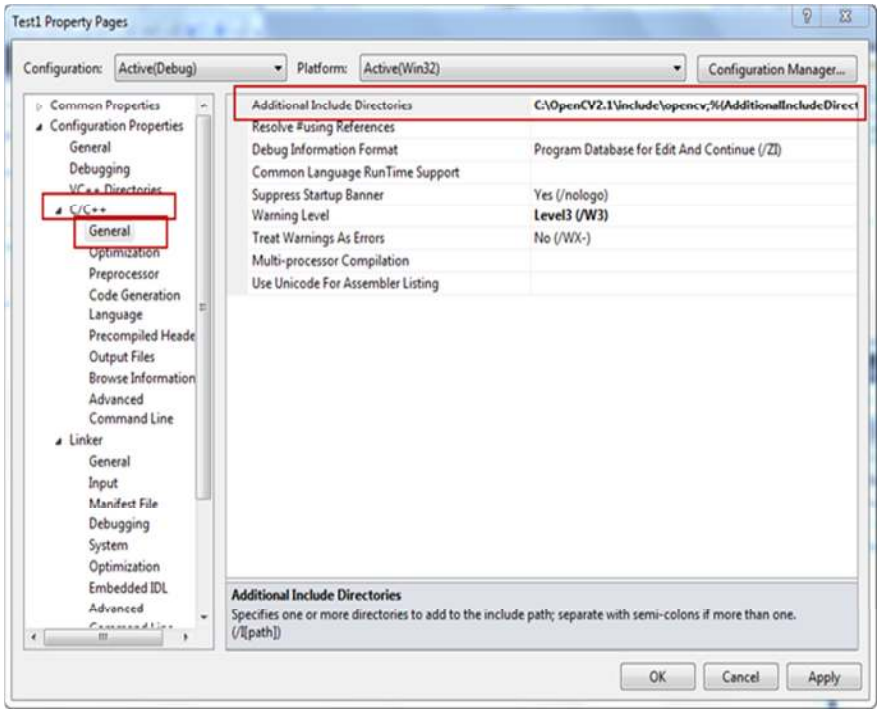
Şekil 4.10. Visual Studio 2010'da C/C++ projesine isim ve kayıt yeri oluşturma.

Projenin properties diyalogunu Şekil 4.11'deki gibi açınız.



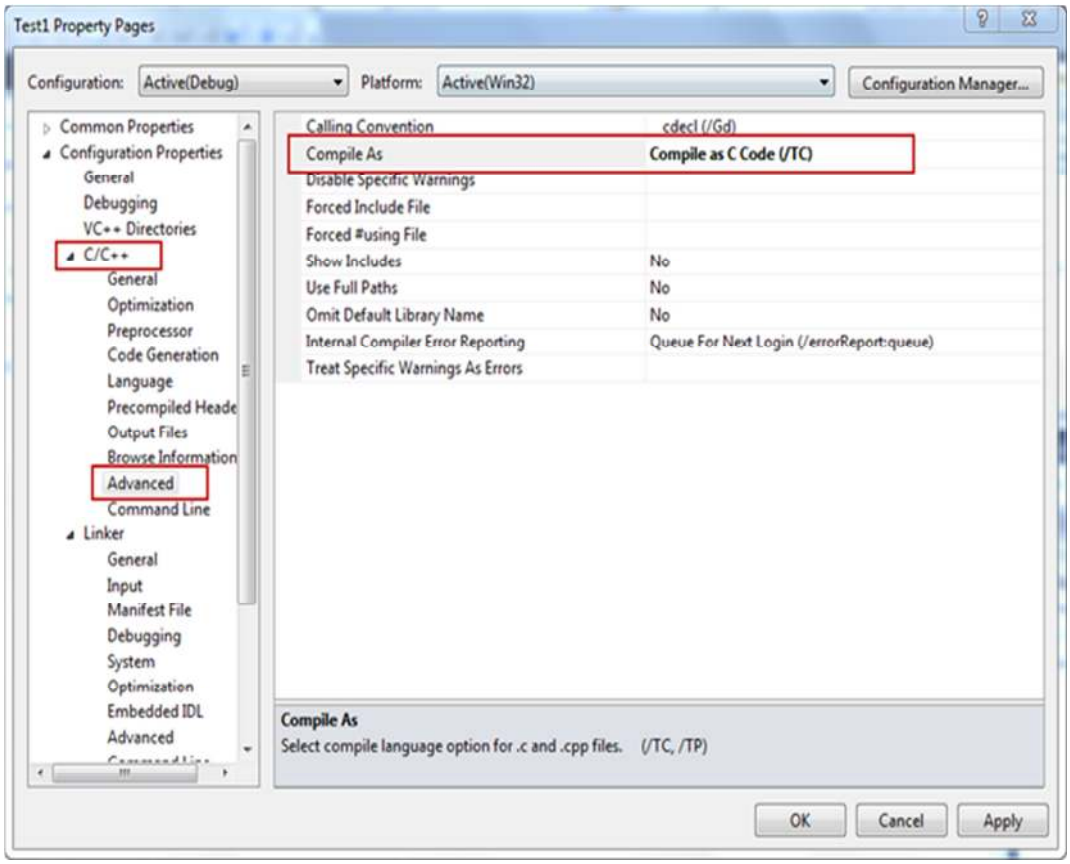
Şekil 4.11. Projenin Properties menüsünü açma

Kurulum dizinini değiştirmediğiniz ve OpenCV 2.1'i C:\OpenCV2.1'e kurduğunuzu farz edersek; Şekil 4.12'de olduğu gibi Properties diyalogunda C/C++ -> Gerenal ve Additional Include Directories alanına C:\OpenCV2.1\include\opencv dizinini yol göstermemiz gerekli.



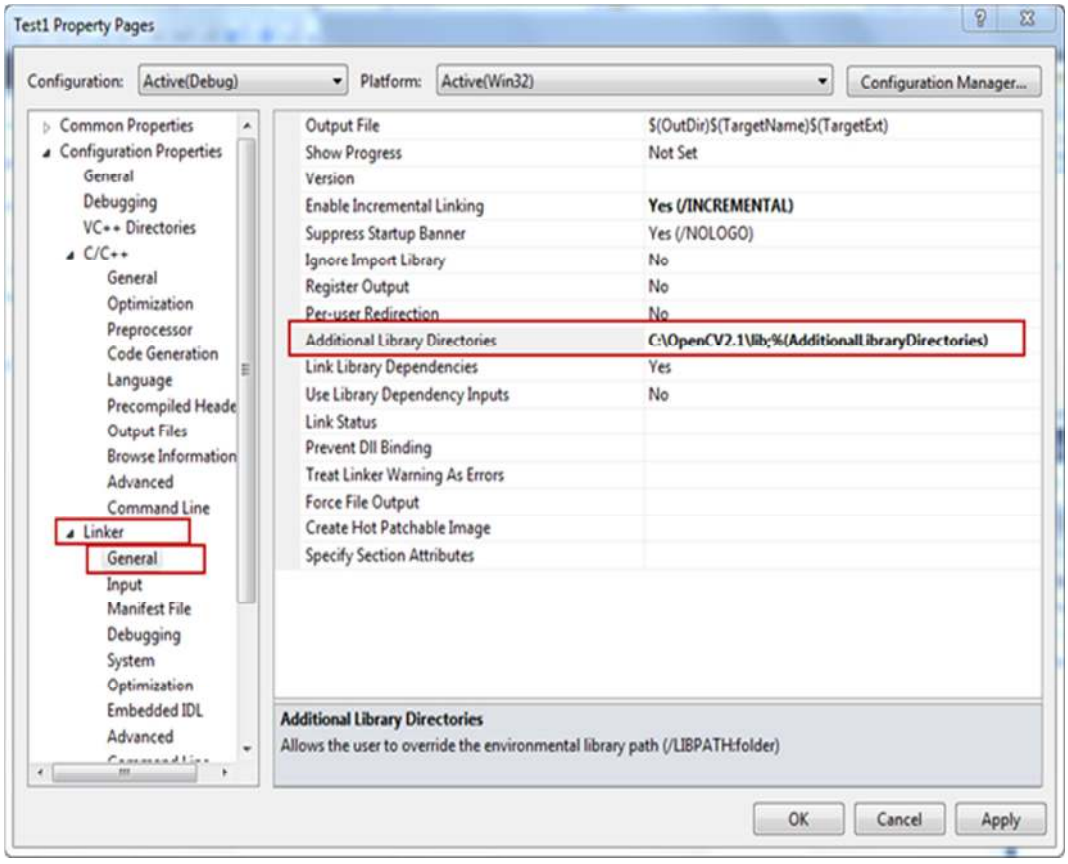
Şekil 4.12. Projenin Properties menüsü

Properties diyalogunda C/C++ –>Advanced'da Compile As'i seçiniz. Şekil 4.13'deki seçimi yapınız.



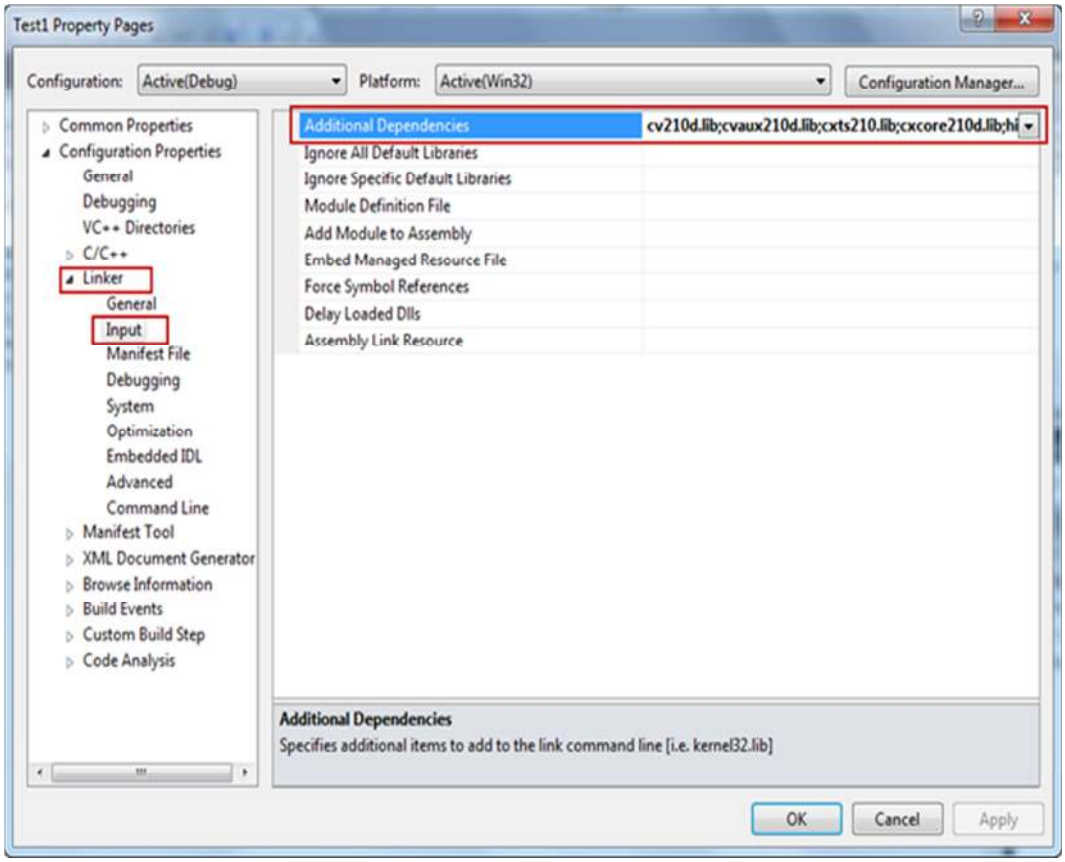
Şekil 4.13. Properties dialog penceresi ayarları

Properties diyalogunda Şekil 4.14 de olduğu gibi Linker –>General'da Additional Library Directories alanına C:\OpenCV2.1\lib giriniz.



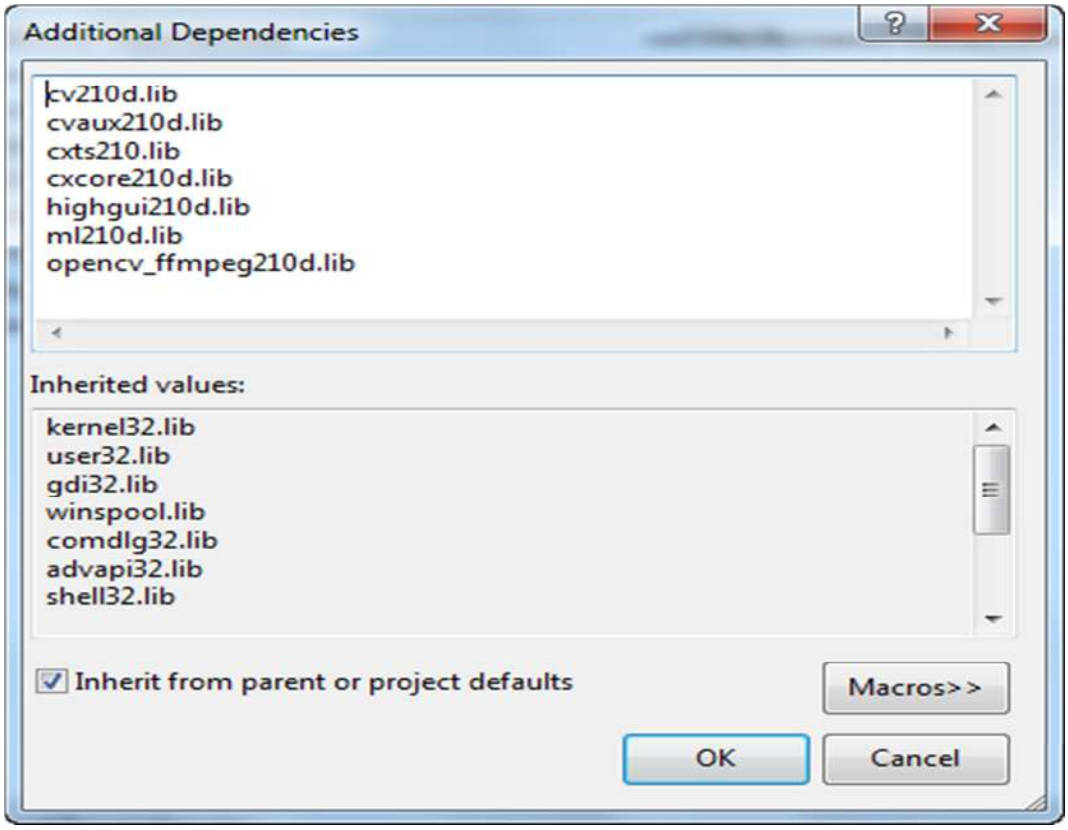
Şekil 4.14. Properties dialog penceresi ayarları

Properties diyalogda Şekil 4.15. de olduğu gibi Linker –>Input'da Additional Dependencies alanına bütün *.lib dosyalarını giriniz. Eğer debug modda çalışıyorsanız, bütün *d.lib dosyalarını ekleyiniz.



Şekil 4.15. Properties dialog penceresi ayarları

Açılan menüden hiç bir şey silmeden <edit> sekmesini tıklayın sizi Şekil 4.16.daki ekran karşılayacaktır. Şekilde gördüğünüz tüm. lib uzantılı dosya isimlerini uzantılarıyla yazıp yazıp OK butonuna tıklayın.

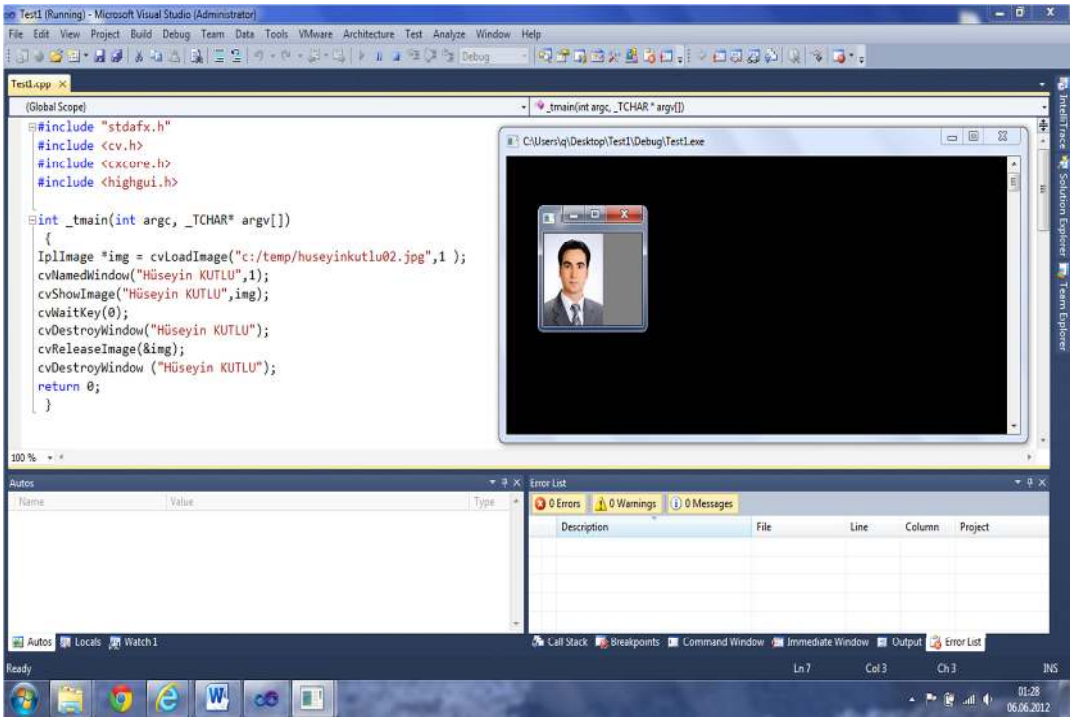


Şekil 4.16. Properties dialog penceresi ayarları

Adım 3: OpenCV'yi Visual Studio 2010'da Test Etme

OpenCV kurduğunuz dizindeki ki biz C:\OpenCV2.1\ dizinine kurduk C:\OpenCV2.1\bin dizinindeki tüm .dll dosyalarını kopyalayarak Visual Studio 2010 projenizin Debug klasörüne atınız.

Test olarak C:/temp/huseyinkutluo2.jpg dosyasını OpenCV ile ekranda göstermeye çalışacağız. Bunun için Şekil 4.17. deki kodları kod bölümüne yazınız. Resim göründüğünde OpenCV Visual Studio 2010'a eklenmiş olacaktır.



Şekil 4.17. Basit bir programla kurulumu test etme

4. 5. Temel OpenCV Komutları

OpenCV optimize edilmiş ve gerçek zamanlı uygulamalar tasarlanması amacı ile ortaya çıkmıştır. OS / donanım / GUI bağımsızlığı ile genel görüntü / video yükleme ve kaydetme gibi birçok işlem yapmak mümkündür. Bunlardan bazı özellikler aşağıda listelenmiştir.

Özellikleri:

- İmge verileri işleme (tanımlama, kaydetme, kopyalama, özellik ayarlama, dönüşüm).
- Resim ve video I / O (video dosyası ve kamera tabanlı girişi, resim)
- Matris ve vektör manipülasyonu ve lineer cebir rutinleri
- Çeşitli dinamik veri yapıları (listeler, kuyruklar, setleri, ağaçlar, grafikler).
- Temel görüntü işleme (filtreleme, kenar algılama, köşe tespit, örnekleme ve interpolasyon, renk dönüşümü, morfolojik işlemler, histogram, görüntü piramitler).
- Kamera kalibrasyonu (bulma ve izleme kalibrasyon desenler, 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, konik, çokgen, metin yazma)

4. 5. 1. Pencere Yönetimi

Opencv'de pencere yönetimi aşağıdaki gibi gerçekleşmektedir.

- ***Bir pencere oluşturmak için;***

```
cvNamedWindow ("pencere1", CV_WINDOW_AUTOSIZE);
cvMoveWindow ("pencere1", 100, 100); / / ekranın UL köşesinden ofset
```

- ***Resim okumak için;***

```
IplImage* imgee=0;
İmgee = cvLoadImage(DosyaAdı);
if(!imgee) //okunmaz ise;
printf("Resim Okunamadı: %s\n", DosyaAdı);
```

- ***Resmi ekranda göstermek için;***

```
cvShowImage ("pencere1" imgee);
```

- ***Pencereyi kapatmak için;***

cvDestroyWindow ("pencere1");

- ***Bir pencereyi yeniden boyutlandırmak için;***

cvResizeWindow ("pencere1", 100, 100); // yeni genişlik piksel cinsinden

4. 5. 2. Girdi Kullanımı

- ***Fare işleyiciyi olayları şu şekilde tanımlanır;***

void mouseHandler (int olay, int x, int y, int bayrak, void * param)

{ (olay) }

Bazı Mouse olayları aşağıda verilmiştir.

- CV_EVENT_LBUTTONDOWN,
- CV_EVENT_RBUTTONDOWN,
- CV_EVENT_MBUTTONDOWN,
- CV_EVENT_LBUTTONUP,
- CV_EVENT_RBUTTONUP,
- CV_EVENT_MBUTTONUP,
- CV_EVENT_LBUTTONDBLCLK,
- CV_EVENT_RBUTTONDBLCLK,
- CV_EVENT_MBUTTONDBLCLK,
- CV_EVENT_MOUSEMOVE:

- ***Klavye olayları:***

Klavye için bir olay işleyici yoktur. Engelleme olmadan klavyeden giriş almak için:

int anahtar;

key = cvWaitKey (10); // giriş için 10ms bekleyin

Klavyeden bir giriş alana kadar bekleme:

int anahtar;

key = cvWaitKey (0); // giriş için belirsiz bir süre bekleme

Klavye ile ilgili ana olaylar için döngü:

```
while(1)
{
süre=cvWaitKey(10);
if(süre==27) break;
```

```
switch(süre)
{
case 'h':
 ...
break;
case 'i':
  ...
break;
}
}
```

4. 5. 3. Temel OpenCV Veri Yapıları

Opencv'de temel veri yapıları aşağıdaki gibidir.

- ***IplImage Görüntü Veri Yapısı***

int n kanal; Renk kanalları için numaraları vardır.(1,2,3,4)
int derinlik; Piksel cinsinden bit derinliği.
(IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U, IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F).
int yükseklik; Piksel cinsinden resim yüksekliği.
İnt genişlik; Piksel cinsinden resim genişliği.

- ***Matrisler ve Vektörler***

CvMat/ 2D dizi: İki boyutlu matris tanımlamak için kullanılır. CvMatND / N-boyutlu bir dizi N boyutlu diziler tanımlamak için kullanılır.
CvArr*: Bir fonksiyon parametresi olarak belirtmek için kullanılır

- ***Points:***

CvPoint p = cvPoint(int x, int y);
CvPoint2D32f p = cvPoint2D32f(float x, float y); //iki boyutlu koordinat sistemi
CvPoint3D32f p = cvPoint3D32f(float x, float y, float z); //üç boyutlu koordinat sistemi

- ***Dikdörtgensel Boyutlar:***

CvSize r = cvSize(int yükseklik, int genişlik); //iki boyutlu koordinat sistemi.
CvSize2D32 r = cvSize2D32f(float yükseklik, float genişlik); //üç boyutlu koordinat sistemi

- ***Ofset ile Dikdörtgensel Boyutlar;***

CvRect r = cvRect(int x, int y, int yükseklik, int genişlik);

- ***Resim tanımlama:***

IplImage* cvCreateImage(CvSize boyut, int derinlik, int kanal);
Size: cvSize(yükseklik, genişlik);
Depth: piksel cinsinden derinlik: IPL_DEPTH_8U, IPL_DEPTH_8S, …
Kanal: kanallar için piksel cinsinden numaralar. Numaralar 1,2,3,4 olabilir.

- ***Bir kanallı bir bayt görüntü tanımlama;***

IplImage* imgee1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);

- ***Üç kanallı bir float görüntü tanımlama;***

IplImage* imgee2=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);

- ***Resmi kapatmak:***

IplImage* imgee=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
cvReleaseImage(&imgee);

- ***Resmi klonlamak:***

IplImage* imgee1 = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
IplImage* imgee2;
imgee2= cvCloneImage(imgee1);

- ***Bilgisayardan bir dosya okumak:***

IplImage* imgee=0;
imgee = cvLoadImage(DosyaAdı);

```
if(!imgee)
printf("Resim dosyası bulunamadı: %s\n", DosyaAdı);
```

OpenCV tarafından desteklenen resim dosyası formatları aşağıda verilmiştir. Bunlar:

BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF, TIF.

Varsayılan olarak yüklenen görüntü 3 kanallı renkli görüntü olarak yüklenir. Bu varsayılan renkli görüntüyü aşağıdaki kod parçacığı kullanarak istediğimiz kanala ayarlayabiliriz:

```
imgee = cvLoadImage(DosyaAdı, bayrak);
```

Bayrak: >0 okunan resim orijinal halini korur.

=0 okunan resim siyah beyaz olarak alınır.

<0 belirtilen kanal sayısına göre renklendirme işlemleri gerçekleştirilir.

- ***Bir görüntü dosyasını yazdırmak için;***

```
if(!cvSaveImage(cikisDosyasıAdi,imgee))
printf("Dosya kaydedilmedi: %s\n", cikisDosyasıAdi);
```

Çıktı dosyası biçimi dosya adı uzantısına göre belirlenir. Bundan dolayı dosya uzantısına dikkat etmeliyiz.

Görüntünün Elemanlarına Erişim Türleri

K kanallı bir görüntünün i. ve j. piksellerine ulaşmayı hedefleyelim. Satir indisinin [0, width-1] aralığında ve sütun indisinin [0, height-1] aralığında olduğu ve kanal aralığının da [0,nchannels-1] olduğunu varsayalım. Buna göre;

Dolaylı erişim, genel, ama verimli değildir. Doğrudan erişim ise etkin bir erişim yöntemi ama hata eğilimlidir.

- ***Daire çizimi:***

Merkezi (100,100) ve yarıçapı 20 olan 1-kanalı koordinatı (100,100)bir daire çizecek olursak;

```
cvCircle(imgee, cvPoint(100,100), 20, cvScalar(0,255,0), 1);
```

- ***Çizgi çizimi:***

Kalınlığı 1 point, başlangıç ve bitiş koordinatları (100,100) ,(200,200) olan bir çizgi çizelim.

```
cvLine(imgee, cvPoint(100,100), cvPoint(200,200),  cvScalar(0,255,0),  1);
```

- ***Kameradan başlatmak:***

```
CvCapture*  yakala = cvCaptureFromCAM(0);
```

Eğer entegre kamera varsa öncelik entegre kameranındır. Aksi halde sırasıyla ilk USB portuna sıfır (0) atanır. Bunun yerine OpenCV2.1 versiyonu ile birlikte gelen device.any(); komutunu yazmamız ile herhangi bir kamara varsa çalıştırsın demek önerilir.

- ***Bir dosyadan fotoğraf çekmeyi başlatmak:***

```
CvCapture*  yakala = cvCaptureFromAVI("dosya. avi");
Bir tane frame yakalamak:
IplImage*  imgee = 0;
if(!cvGrabFrame(yakala)) {              // yakalanan frame
printf("frame yakalanamadı\n");
exit(0);
}
imgee = cvRetrieveFrame(yakala);          // yakalanan çerçeveyi almak
```

Birkaç kamera ile aynı anda görüntü almakta mümkün ancak bunun için senkronizasyona dikkat edilmelidir.

- ***Frame akışını durdurmak için:***

```
cvReleaseCapture(&yakala);
```

- ***Framebilgisi çekmek(set-get):***

Frame akışını sağlayan cihazın özelliklerini tespit etmek istersek aşağıda kodlara başvurmalıyız.

```
cvQueryFrame(yakala);
int frameH = (int) cvGetCaptureProperty(yakala, CV_CAP_PROP_FRAME_HEIGHT);
int frameW =  (int) cvGetCaptureProperty(yakala, CV_CAP_PROP_FRAME_WIDTH);
int fps =  (int) cvGetCaptureProperty(yakala, CV_CAP_PROP_FPS);
int numFrames = (int) cvGetCaptureProperty(yakala,
CV_CAP_PROP_FRAME_COUNT);
```

- ***Frame Bilgisini Almak:***

```
floatposMsec = cvGetCaptureProperty(yakala, CV_CAP_PROP_POS_MSEC);
intposFrames = (int)cvGetCaptureProperty(yakala, CV_CAP_PROP_POS_FRAMES);
floatposRatio  = cvGetCaptureProperty(yakala, CV_CAP_PROP_POS_AVI_RATIO);
```

- ***Bir Video Dosyasını Kaydetmek:***

```
CvVideoWriter *writer = 0;
intisColor = 1;
intfps     = 25;
intframeW  = 640;
intframeH  = 480;
writer = cvCreateVideoWriter("outavi", CV_FOURCC('P','I','M','1'),
fps(cvSize(frameW, frameH), isColor);
```

Diğer olası codec kodları:

```
CV_FOURCC('P','I','M','1') = MPEG-1 codec
CV_FOURCC('M','J','P','G') = motion-jpegcodec (does not workwell)
CV_FOURCC('M', 'P', '4', '2') = MPEG-4,2 codec
CV_FOURCC('D', 'I', 'V', '3') = MPEG-4,3 codec
CV_FOURCC('D', 'I', 'V', 'X')  = MPEG-4 codec
CV_FOURCC('U', '2', '6', '3') = H263 codec
CV_FOURCC('I', '2', '6', '3') = H263I codec
CV_FOURCC('F', 'L', 'V', '1') =FLV1 codec
```

Oynatma işlemini başlatmak için ise aşağıdaki kod parçacığını unutmamalıyız.

```
cvShowImage("GöstermePenceresi",  imgee);
key=cvWaitKey(20);          // bekle 20 ms
```

20 ms'den daha fazla zaman harcanırsa her bir frame için bu düzgün görüntüleme sayılmaz.

- ***Video yazıcıyı kapatmak için:***

```
cvReleaseVideoWriter(&writer);
```

4. 6. OpenCV ile Temel Görüntü İşleme Uygulamaları

OpenCV kütüphanesini etkin bir şekilde kullanabilmek için basit örneklere yer verilmiştir. Tüm örnekler Windows 7 Ultimate 32 bit işletim sistemi üzerinde OpenCV-2.1.0-win32-vs2008 sürümü ve Microsoft Visual Studio 2010 Yazılım Geliştirme Ortamı (IDE) kullanılarak çalıştırılmıştır.

4. 6. 1. OpenCV ile bir Resmin Görüntülenmesi

Ekler bölümünde kodları verilen program dosya yolu verilen bir resmin OpenCV Kütüphanesi ile nasıl açılacağını göstermektedir. IplImage veri yapısı, çeşitli resim dosyalarını hafızaya alabilmek için oluşturulmuş özel bir veri yapısıdır. Program 1'de istenen resim IplImage tipinde bir değişkene cvLoadImage fonksiyonu yardımıyla atanır. Bu fonksiyon, BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF ve TIF uzantılı resim dosyalarını okuyabilir. Okunabilen resim dosyaları cvShowImage Fonksiyonu ile form üzerinde gösterilir. İşlem bittikten sonra yaratılan bütün nesneler hafızadan silinir.

Şekil 4.18. OpenCV'de resmin görüntülenmesi

Şekil 4.18.'de Program 1 çalıştırıldığında elde edilen sonuç görülmektedir.

Belirtilen resim dosyası okunamadığı durumlarda aşağıdaki kontroller ve işlemler yapılmalıdır.

- Programda resim dosyasına ait dosya yolunun (path) doğru şekilde belirtildiğinden emin olunuz.
- Visual Studio 2010 yazılım geliştirme ortamı ile çalışıyorsanız "Visual Studio 2010 Ultimate" yükleyiniz.
- Seçtiğiniz yazılım geliştirme ortamı ile OpenCV kütüphanesini çalıştırabilmek için gerekli ayarları kontrol ediniz.

4. 6. 2. OpenCV 'de Resmi Matris Olarak Alma ve Üzerinde İşlemler Yapma

OpenCV'de resmi bir matris olarak tutup matris kütüphanesindeki fonksyonlardan faydanılabilinir. İmread fonksyonu ile resim okunur. İmread fonksyonununn döndürdüğü değer cv namespace ine ait özel tanımlanmış bir veri yapısı (cv::Mat) bir matrisi temsil eder. Matris üzerinde toplama, çarpma, bölme gibi temel işlemler ve filitreleme işlemleri yapılabilmektedir.

Ekler bölümünde kodları verilen uygulamada öncelikle resmi Mat türünde "im" isminde bir değişkene atanmıştır. Aynı türde "imGray" isminde okunan resimin gri ye çevrilmiş hali için değişken oluşturulmuştur. cvtColor fonksiyonu ile okunan resimin renk dönüşümü gerçekleşmiştir. cvtColor renk dönüşümü işlemini yaparken kaynak, hedef, kod parametreleri ile çalışır. Kaynak renk dönüşümü yapmak istediğiniz resmi bildirir. Hedef renk dönüşümü yapılmış yeni resimi bildirir. Code ise yapılmak istenen renk dönüşümünü bildirir. Bu uygulamada yapılmak istenilen renk dönüşümü CV_RGB2GRAY dir. Resmi göstermek için imshow fonksiyonu kullanılır. İmshow cvShowImage fonksyonundan farklıdır. imshow data olarak direkt Mat türünde alır ve kendi penceresini oluşturur. Mat türündeki resimler üzerinde aritmetik işlemlerle resmin kontrastı düşürülüp arttırılabilir. Griye çevirilmiş bir resimin piksel değerleri 0-255 arasındadır. 0 siyahı 255 de beyazı gösterir. 0-255 arası değerler ise siyah ve beyazın tonlarını gösterir. Bu sebeple resim/3 gibi bir işlemle resmin piksel değerleri düşürerekten resmin kontrastı azaltılabilir, resim * 10 gibi bir işlemle resmin kontrastı arttırılıp daha parlak bir resim elde edilebilir.

Şekil 4.19. Resmi Matris Olarak Alma ve Üzerinde İşlemler Yapma. (a) Orijinal Resim, (b) Gri Resim, (c) Az Kontrastlı Resim, (d) Çok Kontrastlı Resim

4. 6. 3. OpenCV 'de Morfolojik işlemler, Thresholding ve Bağlı Bileşenler Analizi

OpenCV resim üzerinde morfolojik işlemler yapabilmemiz için çeşitli fonksiyonlar sağlamıştır. Morfolojik işlemler çok geniş bir kullanım alanına sahiptir; gürültünün kaldırılması, bazı elementlerin isole edilmesi, kopuk elementlerin birleştirilmesi örnek olarak verilebilir. Morfolojik işlemler genel olarak binary resimlerde kullanılmaktadır.

4. 6. 3. 1. Genişleme (Dilation) ve Aşınma (Erosion)

Sayısal bir görüntüye genişleme süzgecinin uygulanması demek görüntüyü yapısal elemanla kesiştiği bölümler kadar büyütmek demektir. Bunu yapabilmek için yapısal eleman görüntü üzerinde piksel piksel dolaştırılır. Eğer yapısal elemanın merkezi resim üzerinde "0" değerli bir piksel ile karşılasırsa herhangi bir değişiklik meydana gelmez. Eğer değeri "1" olan bir piksel ile karşılaşırsa yapısal elemanla yapısal elemanın altında kalan pikseller mantıksal "or" islemine tabi tutulurlar. Yani herhangi "1" değeriyle sonuç "1" e çevrilir. Genişleme süzgecinin görüntüye uygulanmasıyla görüntü üzerindeki

nesneler büyür. Nesne içinde boşluklar varsa bunlar kapanır ve ayrık nesneler birbirine yaklaşır ya da bağlanır.

Aşınma islemi bir bakıma genişlemenin tersi gibi görülebilir. Burada yine aynı şekilde yapısal eleman görüntü üzerinde piksel piksel dolaştırılır fakat bu defa yapısal elemanın merkez pikseli "1" değeri ile karşılaşırsa yapısal eleman içerisindeki piksellerin durumuna bakılır. Eğer yapısal eleman içerisindeki "1" olan piksellerden herhangi biri altında görüntüye ait "0" değeri varsa yapısal elemanın diğer "1"'lerin altındakilerle beraber bu piksel "0" a dönüştürülür. Aşınma islemi ile görüntü aşındırılmış olur. Yani görüntü içerisindeki nesneler küçülür, bosluk varsa genisler, bağlı nesneler ayrılma eğilimi gösterir.

OpenCV'de dilation ve erosion fonksyonları kullanılırken bir tane kernel, yapısal element, belirlenir. İkisinde de bu element tüm pikseller boyunca dolaştırılır. Dilation da dışarı taşmalar eklenirken, erosion da sadece resmin dâhilinde kalanlar alınır. Yani bir bakıma dilation yeni pikseller eklerken erosion bazı yerleri siler. OpenCV de dilate ve erode diye iki fonksiyon mevcuttur.

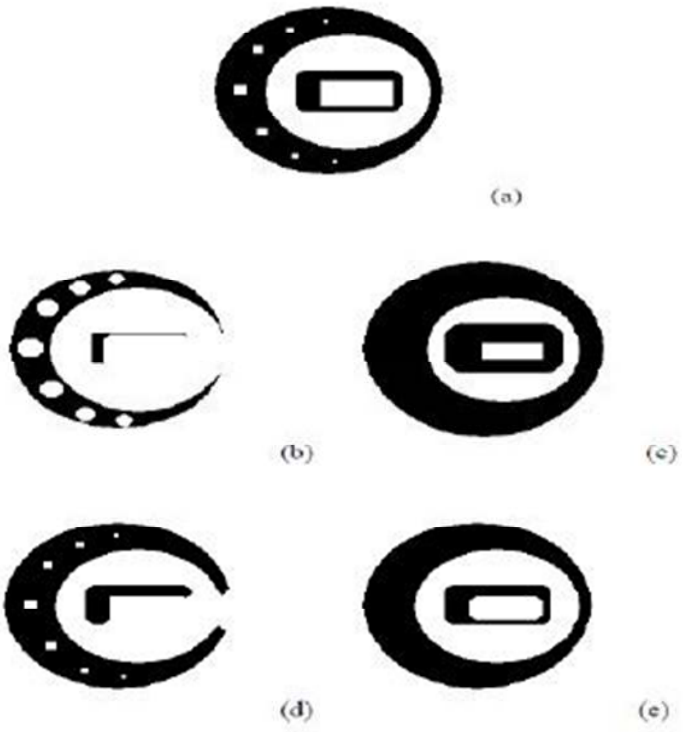
4. 6. 3. 2. Açma (Opening) ve Kapama (Closing)

Açma bir görüntüye önce aşınma daha sonra genişleme uygulanırsa görüntüye açma işlemi uygulanmış olur. Açma işlemi ince çıkıntıları, dışarı doğru olan sivri sınır düzensizliklerini, ince birleşimleri ve izole olmuş küçük nesneleri yok eder. Açma işlemi ile birbirine yakın iki nesne görüntüde fazla değişime sebebiyet vermeden ayrılmış olurlar.

Kapama bir görüntüye önce genişleme daha sonra aşınma uygulanırsa kapama işlemi uygulanmış olur. Kapama islemi ince girintileri, içe doğru olan sivri sınır düzensizliklerini ve küçük boşlukları yok eder. Kapatmada birbirine yakın iki nesne görüntüde fazla değisiklik yapılmadan birbirine bağlanmış olurlar.

Açma ve kapama; genişleme ve aşınma kullanarak elde edilmiş faydalı iki özelliktir. Açma genellikle resimde alanların sayılmasında kullanılır. Kapama çeşitli bağlı elemanlar algoritmalarında gürültü gidermek amaçlı kullanılmaktadır.

OpenCV de opening ve closing kullanmak için morphologyEx fonksiyonu mevcuttur. Bu fonksiyon genişletme ve aşındırma hariç geri kalan morfolojik işlemleri yapmak için kullanılır. Tip olarak bir parametre almaktadır. Parametre olarak MORPH_CLOSE, MORPH_OPEN verilerek kullanılabilir.



Şekil 4.20. OpenCV'de Morfolojik işlemler [2]. (a) Örnek görüntü, (b) Aşınma İşlemi sonucu oluşan görüntü, (c) genişleme işlemi sonucu oluşan görüntü, (d) açma işlemi sonucu oluşan görüntü, (e) kapama işlemi sonucu oluşan görüntü

4. 6. 3. 3. Bağlı Bileşenler Analizi

Bağlı bileşenler analizi yönteminde öncelikle morfolojik işlemlerden aşınma ve genişleme uygulanır (açma ve kapama da uygulanabilir). Morfolojik işlemleri gerçekleştikten sonra görüntüdeki aynı özniteliğe sahip bir bölgeyi kuşatan ve sürekli bir eğri oluşturan noktalar kümesi olan konturlar bulunur. Konturlar bulunurken siyah arka plan pikselleriyle beyaz ön plan piksellerinden faydalanılır. Aynı değere sahip komşu pikseller bir birine bağlıdırlar ve bu birbirine bağlı pikseller birleşerek konturu oluştururlar. Bulunan konturlardan büyüklüğü belirli bir esik değerinin altında olanlar arka planda

meydana gelen gürültü gibi küçük değişiklerden kaynaklanan nesnelerdir ve bu nesneler görüntüden atılır. Konturlara uygulanacak eşik değeri görüntünün boyutlarına bakılarak elde edilir. Böylece eşik değeri görüntüye uygun seçilmiş bir değer olur. Bu yöntem sayesinde görüntüden gürültüler ayrılır.

4. 6. 3. 4. Eşikleme (Thresholding)

Thresholding genellikle resmi binary resme (yani ya siyah ya beyaz) çevirmek için kullanılan bir tekniktir. Gürültüyü önlemek veya nesne belirlemek gibi farklı amaçlarla da kullanılabilir; genel olarak belirlenen bir sınır değeriyle resmin tüm piksellerinin teker teker karşılaştırılması ve belirli bir kurala göre resimi sıfır ve bir'e dönüştürme uygulanmasıdır. Farklı versyonları mevcuttur.

- ***THRESH_BINARY***

Resimdeki piksel sınır değerinden (thresh) büyükse max olarak belirlenen 255 değeri verilir küçükse 0'a değeri verilir. Bu durumda sınırdan büyükler beyaz olurken sınırdan küçükler sıfır olacaktır.

- ***THRESH_BINARY_INV***

THRESH_BINARY nin tam olarak zıttıdır. Piksel sınır değerinden büyükse 0 küçükse 255 değerini alır. Yani piksel sınırdan büyükse siyah küçükse beyaz olacaktır.

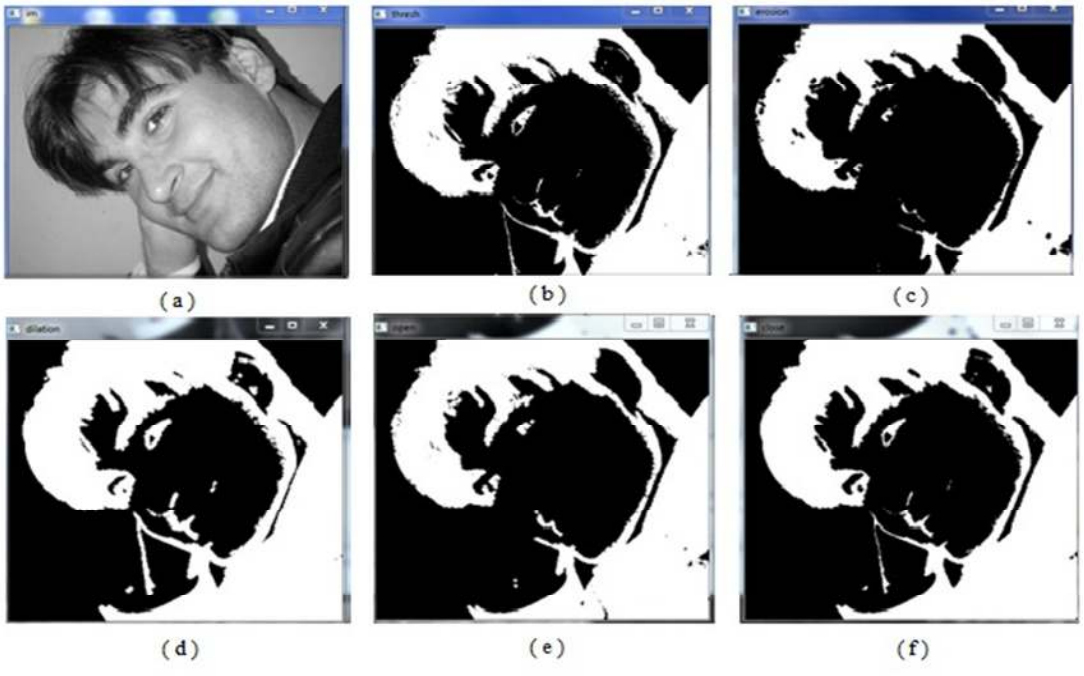
- ***THRESH_TOZERO***

Eğer piksel sınırdan büyükse kendi değerinde kalacak küçükse 0 olacaktır. Yani piksel değerinin sınırdan büyük olduğu yerlerde aynen kalacak, küçük olduğu yerlerde ise tamamen siyah olacaktır.

- ***THRESH_TOZERO_INV***

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

Ekler bölümünde kodları verilen örnek uygulamada “im” isimli matris veri türünde bir değişkene resim atanmıştır. Bu atama yapılırken resimi direkt gri olarak almak için imread fonksyonunda 0 parametresi kullanılmıştır. Yine matris türünde oluşturulacak olan resimleri atamak için thresh, eroded, dilated, open, close isimli değişkenler tanımlanmıştır. cvThreshold(kaynak, hedef, eşik değeri, en üst değer, Code) fonksyonu ile “im” isimli resim ikili resme çevirilmiştir. Code parametresi olarak farklı threshold yöntemleri kullanılabilir. Morfolojik işlemler uygulanırken bir kernel yapısıyla piksellerin üzerinde gezindecek bir yapı tanımlanmıştır ve alternatif yapı tanımlama yöntemleri verilmiştir. Erode fonksyonu ile ikili resim (thresh) hedef resime (eroded) kernel yapısı (element) kullanılarak aşındırma fonksyonu ile kullanılmıştır. Dilate fonksyonu ile ikili resim (thresh) hedef resime (dilated) kernel yapısı (element) kullanılarak aşındırma fonksyonu ile kullanılmıştır. morphologyEx fonksyonu ile resim (thresh) hedef resime (open) MORPH_OPEN parametresi ile kernel yapısı (element) kullanılarak açma işlemi yapılmıştır. morphologyEx fonksyonu ile resim (thresh) hedef resime (close) MORPH_CLOSE parametresi ile kernel yapısı (element) kullanılarak kapama işlemi yapılmıştır. İmshow fonksyonu ile resimler ekranda gösterilmiştir.



Şekil 4.21. OpenCV ‘de Morfolojik işlemler ve Thresholding işlemi. (a) Orijinal Resim. (b) Thresholding uygulanmış resim. (c) Aşındırma işlemi yapılmış resim. (d) Genişleme işlemi yapılmış resim. (e) Açma işlemi yapılmış resim. (f) Kapama işlemi yapılmış resim.

4. 6. 4. OpenCV'de Resimlere Filtre Uygulama

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

Lineer ve lineer olmayan filitre versiyonları mevcuttur. Filitreler genel olarak, bir kernel (çekirdek) matrisin tüm resim boyunca gezdirilip, her piksel için çarpımların toplamının alınması esasına dayanır. Bu sayede her piksel değeri için yeni bir değer hesaplanır. Matrisin büyüklüğü ve içerdiği değerlere göre resmi yumuşatma, keskinleştirme, kenar tespiti gibi çok çeşitli işlemler yapılabilmektedir.

Kernelin ve filtreleme işleminin nasıl yapıldığına örnek olarak:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \quad * \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

İlk matrisi resimdeki piksel değerleri olarak düşünelim ve ikinci matrisi ise kernel olarak düşünelim. Burada e değerine sahip piksel için yeni değeri *(e(2))* hesaplanacak olursa

$$e(2) = a*0 + b*1 + c*0 + d*1\ldots = b+d+f+h-4*e \qquad (4.\ 1.)$$

Bu şekilde kernel tüm resimde gezdirilerek tüm piksellerin yeni değeri hesaplanır. Bu örnekte verilen kernel 3*3 lük laplace filtresidir ve resmin 2. türevini almakta kullanılır. Resmin türevini almak demek resimdeki farklılıkların daha çok vurgulanmasıdır.

Yani kenarların bulunması için kullanılabilir. Aynı zamanda laplace filtresi bir high pass filtredir. Yani yüksek geçirgen, bir bakıma yüksek farklılıkların geçmesine izin verir. Şöyle düşünün, verilen örnekte tüm piksel değerlerini aynı alın; ne olacaktı o zaman e nin yeni değeri 0 olacaktı. Bir bakıma farklılık ne kadar azsa geçirgenliği de bir o kadar az oluyor.

- ***Düşük Geçirgenlikli Filtreler***

Düşük geçirgenlikli (Low pass) filtreler genel olarak resmi blur (smooth) hale getirmek amacıyla kullanılırlar. Bu sayede resimden gürültiler çıkartılabilir. Örnek olarak; Median, Gaussian, Normalize Box gibi filtreler verilebilir. Ya da kendi tanımlayacağınız bir filtreyi de low pass filtre olarak tasarlayabilirsiniz. Örneğin;

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$ Bir Düşük geçirgenlikli filtre kerneli olarak kullanılabilir.

- ***Yüksek Geçirgenlikli Filtreler***

Yüksek geçirgenlikli (High pass) filtreler genel olarak resmi keskinleştirmeye yaramaktadırlar. Laplace filtresi high pass filtrelere en güzel örneklerden biridir. Başka bir kernel örneği olarak;

$$\begin{bmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{bmatrix}$$ Bir yüksek geçirgenlikli filtre kerneli olarak kullanılabilir.

Yüksek geçirgenlikli filtreler genel olarak merkeze daha çok önem verirler.

Resmin köşelerinde kalan piksellere filitre uygulandığında filitre matrisinin dışında kalan elemanlar için bir piksel değeri tayin etmek gerekmektedir. Bu işleme interpolasyon, ya da ekstrapolasyon denilir ve komşu pikselin değeri, komşu pikselin ortalama değeri veya 0 değeri alma gibi farklı yolları mevcuttur. OpenCV'de bu problemleri çözen fonksyonlar aşağıda listelenmiştir.

* BORDER_REPLICATE: aaaaaa|abcdefgh|hhhhhhh

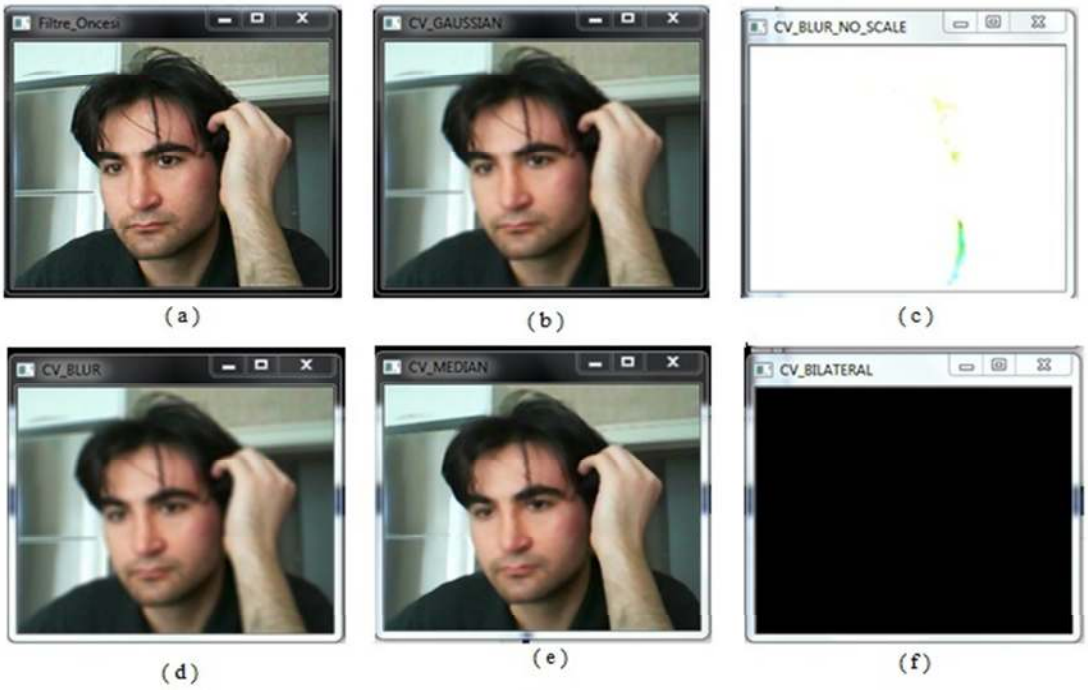
* BORDER_REFLECT: fedcba|abcdefgh|hgfedcb

* BORDER_REFLECT_101: gfedcb|abcdefgh|gfedcba

* BORDER_WRAP: cdefgh|abcdefgh|abcdefg

* BORDER_CONSTANT: iiiiii|abcdefgh|iiiiiii

Ekler bölümünde kodları verilen örnek uygulamada img isimli değişkene resim atanmıştır. Resmin orijinali bir pencere oluşturulup gösterilmiştir. Farklı türlerde blurlaştırma işlemleri yapılacağı için bu türleri alacak resim değişkenleri tanımlanmıştır. Resim değişkenleri için orijinal resmin boyutlarında bir resim yaratılmıştır. Blurlaştırma fonksyonu olan cvSmooth fonksyonuna kaynak resim değişkeni, hedef resim değişkeni, blurlaştırma tipi parametre olarak verilip sonuçlar ekranda gösterilmiştir.

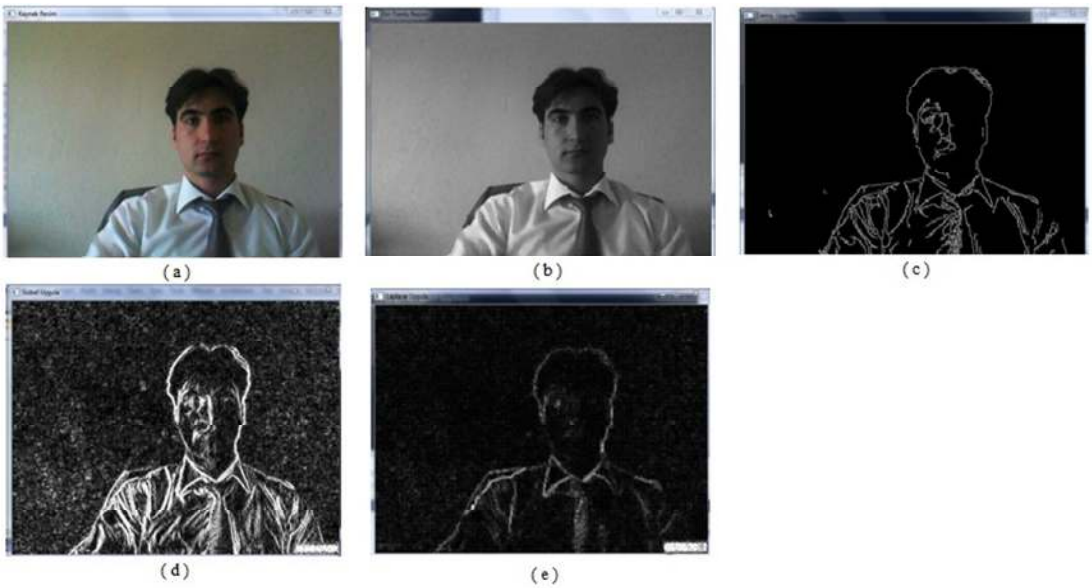


Şekil 4.22. OpenCV 'de resme filitre uygulama. (a) Orijinal resim. (b) Gaussian filitresi uygulanmış resim. (c) cvblurnoscale parametresi ile filitrelenmiş resim. (d) Blurlaştırılmış resim. (e) Median filitresi uygulanmış resim. (f) Bilateral filitresi uygulanmış resim.

4. 6. 5. OpenCV'de Kenar Bulma

Bir resimde kenarlar sınırları karakterize etmektedir. Bir görüntüdeki kenarlar güçlü yoğunluk zıtlığı olan bölgelerdedir 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.

Ekler bölümünde kodları verilen uygulamada görüntü işlemede sık kullanılan Canny, Sobel, Laplace, kenar bulma yöntemlerinin OpenCV fonksyonları ile uygulaması yapılmıştır. Şekil 4.3'deki standart test imajı (hk02.jpg) ile çalıştırıldığında, Şekil 4.4'teki Gri tonlu resim ile program akışındaki seçime bağlı olarak bu üç kenar bulma yöntemlerinden biri uygulanmış resim elde edilir. Şekil 4.5'te Canny kenar bulma yöntemi ile kenarları tespit edilen resim, Şekil 4.6'da Sobel kenar bulma yöntemi ile kenarları tespit edilen resim, Şekil 4.7'de ise Laplace kenar bulma yöntemi ile kenarları tespit edilen resim görülmektedir.



Şekil 4.23. OpenCv kenar bulma fonksyonlarının kullanımı. (a) Kaynak Resim..(b)Gri Tonlu Resim. (c) cvCanny Fonksiyonu Çıktısı. (d) cvSobel Fonksiyonu Çıktısı (e). cvLaplace Fonksiyonu Çıktısı

4. 6. 6. Kamera ile Gerçek Zamanlı Görüntü Yakalama

OpenCV kütüphanesi ile USB ya da dâhili web kamerasından görüntü yakalayarak gerçek-zamanlı uygulamalar geliştirilmektedir. Sistemde birden fazla kamera kullanılması durumunda gerekli kamera ID'si belirlenip ilgili metoda parametre olarak girilmelidir.

Ekler bölümünde kodları verilen örnek uygulamada herhangi bir kameradan görüntü yakalayabilmek için gereken temel kodlara yer verilmiştir. "cvCaptureFromCAM" metodu

kullanılan kameranın ID'sini parametre olarak alır. Sistemde tek bir USB kamera kullanılıyorsa parametre olarak 0 (CV_CAP_ANY), birden fazla kamera kullanılıyorsa 100 (CV_CAP_MIL), 200 (CV_CAP_VFW) ya da 300 (CV_CAP_FIREWIRE, CV_CAP_IEEE1394, ..vs) değerlerinden biri parametre olarak kullanılır. Sistemde kullanılan kameranın çeşidine ve sayısına göre, deneme yanılma yöntemiyle gereken parametre seçilir. cvCaptureFromCAM" metodu ile "CvCapture" tipinden bir değişkene görüntü gelmesi sağlandıktan sonra sonsuz bir döngü yardımıyla yakalanan görüntünün içerisindeki çerçeveler/resimler (frame) "cvQueryFrame" metodu ile tek tek sorgulanıp okutularak ekranda gösterilir. ESC'a basıldığında görüntü yakalama işlemi sona erer.

4. 7. OpenCV' de Arka Plan Ayırma

Nesne veya nesne parçalarını görüntünün geri kalan parçalarından ayırmaktaki amaç görüntünün tümüyle uğraşmak yerine gerekli olan parça ile ilgilenip zaman ve hız açısından kazanç sağlamaktır. Örneğin video güvenlik sistemlerinde kamera genel olarak aynı arka plana bakar. Aynı arka planla ilgilenmek sürekli bir emek istemekte ve vakit ve performans kaybına sebep olmaktadır. Arka planla ilgilenmek yerine insanların ya da araçların görüntüye girdiği ya da görüntüye daha önce orada olmayan bir şey bırakıldığı parçalarla ilgilenmek zaman ve performans kazancı sağlayacaktır.

Görüntü işleme uygulamalarında görüntünün geri kalan kısmından ön plandaki nesnelerin ayrılmasının ötesinde görüntüdeki nesnelerin parçalarını örneğin kişinin görüntüsünden sadece yüzünü ya da ellerini izole etmek gibi pek çok durumla karşılmaktadır. Ayrıca bir görüntüyü önişleme tabii tutarak kol, bacak, saç, yüz, gövde, yaprak, göl, yol, çimen vb. nesneleri barındıran bir görüntüden anlamlı olan pikselleri ayırabiliriz. Resmin tümünden ayrılmış anlamlı piksellerin kullanımı sayesinde hesaplamadan tasarruf edilmektedir.

Bu bölümde görüntü segmentasyon algoritmalarından görüntü morfolojisi, flood fill, eşik(threshold) ve piramit segmentasyonu incelenmiştir. Bu bölümde, bir görüntüdeki nesne ve nesne parçalarının bulunması, doldurulması ve izole edilmesi ile ilgili diğer algoritmalar incelenmiştir. Bu arkaplan modelleme fonksiyonları gömülü OpenCV fonksiyonları olmaktan ziyade daha karmaşık algoritmaları tamamlamak için OpenCV fonksiyonlarının nasıl kullanılabileceğini göstermektedir.

4. 7. 1. Arka Plan Çıkarımı

Arka plan çıkarımı basit olması ve görüntü işleme uygulamalarında kamera yerlerinin sabit olması nedeniyle *arkaplan çıkarımı* video güvenlik uygulamaları için en temel görüntü işleme operasyonudur. Toyama, Krumm, Brumitt, ve Meyers pek çok tekniğe ilişkin faydalı bir genel açıklama ve mukayese ortaya koymuşlardır [20]. Arkaplan çıkarımını uygulamak için öncelikle bir arkaplan modeli öğrenilmelidir. Arkaplan modeli öğrenildiğinde mevcut görüntü ile karşılaştırılır ve daha sonra ise bilinen arka plan parçaları çıkarılır. Çıkarma işleminden sonra kalan nesneler ön plan nesneleridir.

Arkaplan çıkarma işleminde ilk adım arka planı belirlemektir. Arka plan belirleme işlemi uygulamalara göre değişen bir kavramdır. Örneğin bir anayolu izlendiğini farz edelim. İzlenen görüntüde ortalama trafik akışı peryodik bir hareket olduğundan arkaplan olarak farzedilebilir. Normalde arkaplan, kameradaki görüntünün ilgili dönem boyunca statik veya periyodik nitelikte olup statik ya da periyodik biçimde hareket eden parçaları olarak düşünülmektedir. Sabah ve akşam rüzgârda sallanıp öğlen vaktinde hareketsiz duran ağaçların olduğu bir görüntüde ağaçlar gibi zamanla değişen bileşenler olabilir. Karşılaşılması muhtemel bu ortamda arka plan çıkarımı yapabilen uygulamalara ihtiyaç duyulmaktadır. Bu bölümde arkaplan çıkarma yöntemleri incelenirken ilk olarak tipik arkaplan modellerinin zayıf yönleri daha sonra yüksek seviye görüntü modellerinde arkaplan ayrımı ele alınacaktır. Ardından görüntünün arka planında rüzgârda sallanan ağaçlar gibi periyodik hareketlerin olmasından ve aydınlatmanın yavaş veya periyodik olarak değişmesinden etkilenmeyen, ön plan nesneleri hareket etseler bile arkaplanın öğrenilmesine karşı toleranslı olan kod çizelgesi metodu (CodeBook) metodu incelenecektir. Son olarak ise hızlı arkaplan yöntemini (quick background method) tez çalışması kapsamında olan bilgisayar insan etkileşimi uygulamamızda kullandığımız kod çizelgesi arkaplan yöntemiyle (codebook background method) karşılaştıracağız.

4. 7. 2. Arkaplan Çıkarım İşleminin Zayıf Yönleri

Arkaplan modelleme yöntemleri ışığın sabit, arka planın sabit olduğu bir görüntüde gayet etkili olsa da tüm piksellerin bağımsız olduğu değişken bir arka planda aynı etkiye sahip olmamaktadırlar. Sabit bir arka planda yapılan arka plan modellemesi bir pikselin komşu pikselleri hesaba katmadan geçirdiği değişimler için olan modeli açıklar. Etrafındaki pikselleri de hesaba katmak için çok parçalı bir model öğrenebilir. Örnek

olarak temel komşularından bağımsız piksellerden oluşan arkaplan modelimize komşu piksellerin parlaklığının ve ana özelliğinin de dâhil edilmesi için genişletilmesi gösterilebilir. Bu durumda komşu piksellerin değerlerinin ne zaman önemli ölçüde parlak ya da loş olduğunu anlamak için onların parlaklığından faydalanılmaktadır. Pikselin komşu piksellerinin parlaklık ve loşluk değerlerine göre iki tane tekli piksel modeli oluşmakta ve böylelikle, elimizde bağlılığı da hesaba katan bir model olmaktadır. Bu durumda komşu piksellerin parlak veya loş olduğu zamanlar için farklı iki değerlere ihtiyacımız olduğuna göre bunu yapabilmek için iki kat daha fazla hafıza kullanımı ve daha fazla hesaplama işlemi ve ayrıca bu iki durumlu modeli tamamlamak için iki kat veri gerekmektedir. Bu ekstra maliyetlerden dolayı, daha karmaşık modellerden genellikle uzak durulmaktadır.

Genel olarak kaynaklarımızı bağımsız piksel varsayımı bozulduğu zaman yani hareketli bir arka plan olduğu zaman hatalı sonuç veren piksellerin temizlenmesiyle daha etkili bir şekilde kullanılabilir. Bu temizleme işlemi, hatalı sonuç veren piksel parçalarını çoğunlukla cvErode(), cvDilate(), ve cvFloodFill() gibi fonksyonlarla yok etmektedir.

4. 7. 3. Arka Plan Modelleme

Görüntüdeki arka ve önplanı nesnelerinin nasıl tanımlanacağı bir örnekle anlatılabilinir. Bir otoparkı izleniyorsa ve otoparka park etmek için bir araba gelirse, o zaman araba yeni bir önplan nesnesi olur. Ancak bu hep böyle kalmaz. Hareket ettirilen bir çöp kutusu ekrana girdiğinde çöp kutusu iki yerde önplan olarak görünür: hareket ettirildikten sonra koyulduğu yer ve başlangıçta olduğu yer birbirinden farklıdır. Bu durumda çöp kutusuda ön plan nesnesi olarak tanımlanacaktır. Başka bir örnek şu şekilde verilebilir örneğin karanlık bir odayı modellenirken aniden bir ışık yanarsa, tüm oda önplan aydınlandığından ön plan nesnesi olacaktır. Örneklerde belirtilen olumsuzluklar için ön ve arkaplan durumları arasındaki çoklu seviyeleri ve zamana dayalı hareketsiz önplan parçalarını arkaplan parçalarına yavaşça indirgeme yöntemini tanımladığımız daha yüksek seviyeli bir "görüntü/alan" modeli gerekmektedir. Ayrıca görüntüde global bir değişim olduğunda yeni bir model bulunmalıdır.

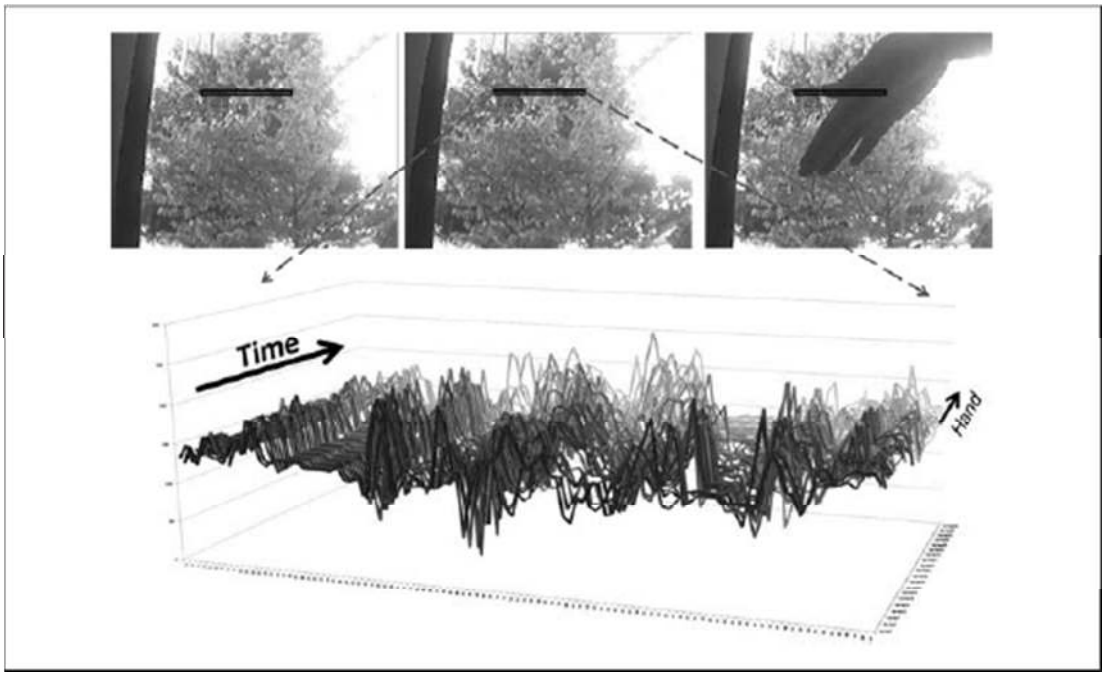
Genel olarak bir görüntü modeli yeni önplan ve eski önplan olmak üzere çoklu katmanlar içermektedir. Hareket algılama bu şekilde gerçekleşmektedir. Bir nesnenin eski ve yeni yeri belirlendiğinde nesnenin hareketi tanımlanabilir.

Herhangi bir önplan nesnesi olmadığı alanlarda ise arkaplan modelimizi geliştirmeye

devam edilebilir. Eğer önplan nesnesi belli bir süre boyunca hareket etmezse, piksel istatiklerinin öğrenilmiş modeli öğrenilmiş arkaplan modeliyle birleşene kadar geçici bir süreliğine öğrenildiği "eski önplan" durumuna indirgenir.

Bir odadaki ışığı açmak gibi genel değişiklik sezimi için genel çerçeve farkından faydalanılabilir. Örneğin, çok sayıda piksel aniden değişirse genelden çok yerel bir değişim olarak sınıflandırdıktan sonra yeni durum için aydınlık arkaplanı model olarak kullanabiliriz.

Piksel değişikliklerini modellemeye geçmeden önce bir görüntüdeki piksellerin zamanla neye benzediklerine dair örnekler vermek gerekir. Örneğin bir kamera rüzgârda sallanan bir ağacı çektiğinde Şekil 4.24.'da, görüntünün belirli bir çizgi parçasında piksellerin 60 karede neye benzediklerini göstermektedir. Bu tür dalgalanmaları arkaplan olarak algılamak istenmektedir.



Şekil 4.24: 60 karede rüzgârda hareket eden bir ağaç görüntüsünde bulunan piksel doğrusundaki dalgalanmalar: hareket eden dallar (üst merkez) büyük ölçüde değişiklik gösterebilirken koyu alanların bazıları (sol üst) oldukça sabittir.

OpenCV'de rastgele bir piksel çizgisini kolayca örnekleyen fonksiyonlar mevcuttur. Doğru örnekleme fonksiyonları cvInitLineIterator() ve CV_NEXT_LINE_POINT() fonksyonudur. CvInitLineIterator() için fonksiyonunun kullanımı aşağıda verilmiştir:

int cvInitLineIterator(

```
    const CvArr* image,
    CvPoint pt1,
    CvPoint pt2,
    CvLineIterator* line_iterator,
    int connectivity = 8,
    int left_to_right = 0
);
```

Girdi görüntüsü, herhangi bir türde ya da sayıda kanaldan oluşabilir. pt1 ve pt2 noktaları çizginin uçlarıdır. Yineleyici line_iterator araya girip uçlar arasındaki çizgi boyunca pikselleri işaret eder. Çok kanallı görüntülerde ise CV_NEXT_LINE_POINT()'a yapılan her çağrı line_iterator'u bir sonraki piksele doğru hareket ettirir. Tüm kanallar line_iterator.ptr[0], line_iterator.ptr[1] vb. olarak hemen kullanıma hazır olur. Bağlantı 6 (çizgi sağa, sola, yukarı veya aşağı gidebilir) veya 8 (çizgi ayrıca köşegenler boyunca hareket edebilir) olabilir. Son olarak left_to_right 0'a (yanlış) ayarlanırsa o zaman line_iterator pt1'den pt2'ye doğru tarar; aksi halde en soldan en sağ noktaya gider. CvInitLineIterator() fonksiyonu, bahsi geçen çizgi için yinelenecek olan nokta sayısını geri çevirir. CV_NEXT_LINE_POINT(line_iterator) yineleyiciyi bir pikselden diğerine götürür.

Bir dosyadan veri çıkarmak için bu yöntemin nasıl kullanılabileceğine tekrardan bakalım. Daha sonra o görüntü dosyasından veriler açısından Şekil 4.24.'u yeniden inceleyebiliriz.

Aşağıdaki kodlar yardımı ile video satırındaki tüm piksellerin RGB değerleri okunur ve bu değerler, üç ayrı dosyada toplanır.

```
// STORE TO DISK A LINE SEGMENT OF BGR PIXELS FROM pt1 to pt2.
 //
 CvCapture* capture = cvCreateFileCapture( argv[1] );
 int max_buffer;
 IplImage* rawImage;
 int r[10000],g[10000],b[10000];
 CvLineIterator iterator;
 FILE *fptrb = fopen("blines.csv","w"); // Store the data here
```

```
FILE *fptrg = fopen(“glines.csv”,“w”); // for each color channel
FILE *fptrr = fopen(“rlines.csv”,“w”);

// MAIN PROCESSING LOOP:
//
for(;;){
if( !cvGrabFrame( capture ))
break;
rawImage = cvRetrieveFrame( capture );
max_buffer = cvInitLineIterator(rawImage,pt1,pt2,&iterator,8,0);
for(int j=0; j<max_buffer; j++){
fprintf(fptrb,“%d,”, iterator.ptr[0]); //Write blue value
fprintf(fptrg,“%d,”, iterator.ptr[1]); //green
fprintf(fptrr,“%d,”, iterator.ptr[2]); //red
iterator.ptr[2] = 255; //Mark this sample in red
CV_NEXT_LINE_POINT(iterator); //Step to the next pixel
}
// OUTPUT THE DATA IN ROWS:
//
fprintf(fptrb,“/n”);fprintf(fptrg,“/n”);fprintf(fptrr,“/n”);
}
// CLEAN UP:
//
fclose(fptrb); fclose(fptrg); fclose(fptrr);
cvReleaseCapture( &capture );
```

Çizgi örneklenmesini aşağıdaki gibi daha kolay yapabilirdik:

```
int cvSampleLine(
const CvArr* image,
CvPoint pt1,
CvPoint pt2,
void* buffer,
int connectivity = 8
```

);

Kısacası bu fonksiyon cvInitLineIterator() fonksiyonunu makro CV_NEXT_ LINE_POINT (line_iterator)'e bağlar. pt1'den pt2'ye örnekleme yapar; sonrasında doğru tür ve uzunlukta bir *N*channels × max(|pt2*x* – pt2*x*| + 1,|pt2*y* – pt2*y*| + 1) arabellek pointer'ına devrolur. Çizgi yineleyicisi(line iterator) gibi cvSampleLine() bir sonraki piksele geçmeden önce çok kanallı bir görüntü içerisindeki her pikselin her kanalına girer.

Basitten oldukça kompleks modellere geçerken gerçek zamanlı ve makul hafıza sınırları içerisinde çalışan bu modeller üzerine yoğunlaşacağız.

4. 7. 3. 1. Çerçeve Farkı Yöntemi

En basit arkaplan çıkarım yöntemi bir çerçeveyi diğerinden çıkarıp daha sonra "yeteri kadar büyük" olan her farkı tanımlamaktır. Bu süreçte hareket eden nesnelerin kenarları yakalanmaktadır. frameTime1, frameTime2 ve frameForeground olmak üzere üç adet tek kanallı görüntüden FrameTime1 görüntüsü önceki griölçek görüntüsüyle(grayscale image), frameTime2 ise şu anki griölçek görüntüsüyle doldurulur. Sonrasında aşağıdaki kodu frameForeground'daki önplan farklılıkların mutlak değer büyüklüğünü belirlemek için kullanabiliriz:

```
cvAbsDiff(
frameTime1,
frameTime2,
frameForeground
);
```

Piksel değerleri her seferinde gürültü ve dalgalanmaları gösterdiğinden dolayı 0'a ayarlı küçük çaplı farklılıkları görmezden gelip geri kalanını 255'e ayarlı büyük çaplı olarak işaretlemeliyiz.

```
cvThreshold(
frameForeground,
frameForeground,
15,
255,
```

CV_THRESH_BINARY

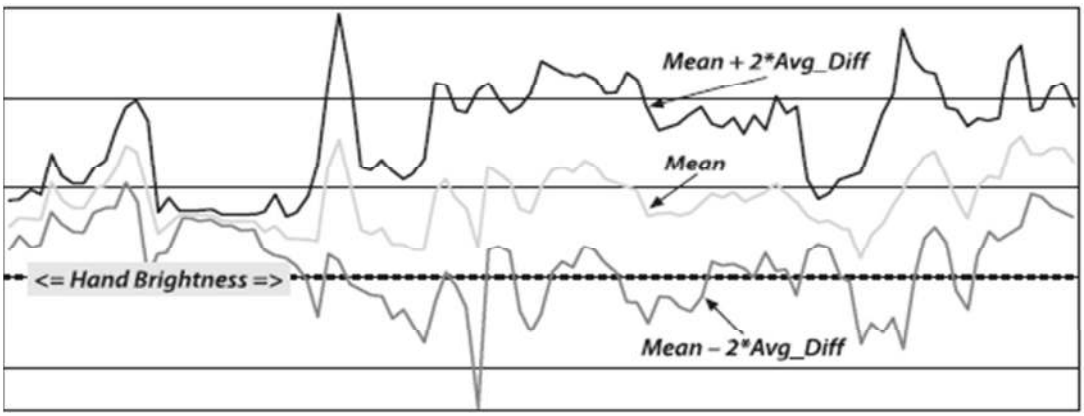
);

frameForeground görüntüsü aday önplan nesnelerini 255 ve arkaplan piksellerini ise 0 olarak işaretler. Daha önce anlattığımız gibi küçük gürültü alanlarını temizlemeliyiz; bunu cvErode() ile ya da bağlantılı bileşenleri kullanarak yapabiliriz. Renkli görüntülere gelince her renkli kanal için aynı kodu kullanıp daha sonra kanalları cvOr() ile birleştirebiliriz. Bu yöntem, hareketli bölgeleri öylece göstermekten ziyade uygulamaların çoğu için çok fazla basittir. Daha etkili bir arkaplan modeli için görüntüde bulunan piksellerin ortalamalarına(mean) ve ortalama farklılıklara ilişkin istatistiksel bilgiler tutmamız gerekir.

4. 7. 3. 2. Arkaplan Ortalama Yöntemi(Averaging Background Method)

Bu yöntem temelde her pikselin arkaplan modeli olarak ortalama ve standart farklılığı ile ilgilidir.

Şekil 4.24.'da bulunan piksel çizgisi göz önüne alındığında her çerçeve için bir değer zinciri çizmek yerine tüm video boyunca ortalama farklılıklar yönünden (Şekil 4.25.) her piksel değişimi açıklanabilir. Rüzgârdan sallanan ağaçların bulunduğu videoda bir önplan nesnesi olan el kameranın önüne geçtiğinde bu önplan nesnesi, arkaplandaki gökyüzü ve ağaç kadar parlak değildir. Elin parlaklığı da resimde gösterilmiştir.



Şekil 4.25. Yüksek/Düşük Eşikli Ortalama Yöntemi**.** Ortalama farklılıkları yönünden Şekil 4.24.daki veriler: kameranın önüne geçen bir el biraz daha koyu olur ve o nesnenin parlaklığı grafikte gösterilmiştir.

Ortalama yöntemi, dört OpenCV şablonunu kullanmaktadır: cvAcc(), zamanla görüntü biriktirmek için; cvAbsDiff(), zamanla kareden kareye(frame-to-frame) görüntü farkları biriktirmek için; cvInRange(), görüntüyü(arkaplan modeli öğrenildiği zaman) ön ve arkaplan bölgelerine ayırmak için; ve cvOr(), farklı renkli kanallardan tekli maske görüntüsüne(single mask image) segmentasyon toplamak için kullanılmaktadır. Bu oldukça uzun bir kod örneği olduğundan dolayı parçalara ayırıp sırasıyla her parçayı ele alınacaktır.

İlk olarak çeşitli istatistiksel bilgiler içeren görüntüler için pointer oluşturulmuştur. Bu pointerların daha sonra içerecekleri görüntü türüne göre sınıflandırılmasında yardımcı olacaktır.

```
//Global storage
//
//Float, 3-channel images
//
IplImage *IavgF,*IdiffF, *IprevF, *IhiF, *IlowF;
IplImage *Iscratch,*Iscratch2;
//Float, 1-channel images
//
IplImage *Igray1,*Igray2, *Igray3;
IplImage *Ilow1, *Ilow2, *Ilow3;
IplImage *Ihi1, *Ihi2, *Ihi3;
// Byte, 1-channel image
//
IplImage *Imaskt;
//Counts number of images learned for averaging later.
//
float Icount;
```

Daha sonra ise gerekli olan tüm ara görüntüleri toplamak için tekli bir çağrı oluşturulmuştur. Ara görüntülerin boyutlandırılması için kaynak olarak kullanılabilecek (videodan) tekli bir görüntüye geçilmiştir.

```
// I is just a sample image for allocation purposes
// (passed in for sizing)
//
void AllocateImages( IplImage* I ){
CvSize sz = cvGetSize( I );
IavgF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
IdiffF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
IprevF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
IhiF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
IlowF = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
Ilow1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Ilow2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
```

```
Ilow3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Ihi1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Ihi2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Ihi3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
cvZero( IavgF );
cvZero( IdiffF );
cvZero( IprevF );
cvZero( IhiF );
cvZero( IlowF );
Icount = 0.00001; //Protect against divide by zero
Iscratch = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
Iscratch2 = cvCreateImage( sz, IPL_DEPTH_32F, 3 );
Igray1 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Igray2 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Igray3 = cvCreateImage( sz, IPL_DEPTH_32F, 1 );
Imaskt = cvCreateImage( sz, IPL_DEPTH_8U, 1 );
cvZero( Iscratch );
cvZero( Iscratch2 );
}
```

Bir sonraki kod parçasında toplanan arkaplan görüntüsünü ve kareden kareye olan farkların toplam mutlak değerini öğreniriz. Bunun için genelde 30-1000 kare gerekir. Bazen her saniyeden sadece birkaç kare alınırken bazen de mevcut tüm kareler alınır. Program, derinliği 8 bit olan üç renkli bir kanal görüntüsüyle adlandırılır.

```
// Learn the background statistics for one more frame
// I is a color sample of the background, 3-channel, 8u
//
void accumulateBackground( IplImage *I ){
static int first = 1; // nb. Not thread safe
cvCvtScale( I, Iscratch, 1, 0 ); // convert to float
if( !first ){
cvAcc( Iscratch, IavgF );
cvAbsDiff( Iscratch, IprevF, Iscratch2 );
cvAcc( Iscratch2, IdiffF );
Icount += 1.0;
}
first = 0;
cvCopy( Iscratch, IprevF );
}
```

Ham arkaplan kanal başına 8 bit ve üç kanallı görüntüyü kayan noktalı ve üç kanallı bir görüntüye dönüştürmek için önce cvCvtScale()'i kullanır ve ham kayan noktalı görüntüleri IavgF'de toplanmaktadır. Ardından cvAbsDiff() kullanarak kareden kareye mutlak farkı hesaplanmakta ve IdiffF görüntüsünde toplanmaktadır. Bu görüntüler her toplandığında daha sonra ortalamak için görüntü sayısı Icount arttırılmaktadır.

Yeteri kadar kare toplandığı zaman arkaplanın istatistik modeline dönüştürülmektedir. Yani, her pikselin ortalama mutlak farklılıkları hesaplanmaktadır.

```
void createModelsfromStats() {
cvConvertScale( IavgF, IavgF,( double)(1.0/Icount) );
cvConvertScale( IdiffF, IdiffF,(double)(1.0/Icount) );
//Make sure diff is always something
//
cvAddS( IdiffF, cvScalar( 1.0, 1.0, 1.0), IdiffF );
setHighThreshold( 7.0 );
setLowThreshold( 6.0 );
}
```

Bu kodda cvConvertScale(), ortalama ham ve mutlak fark görüntülerini toplanan girdi görüntülerinin sayısına bölerek hesaplamaktadır. Önlem amaçlı olarak ortalama fark görüntüsünün en az 1'de olmasını sağlanmaktadır. Önplan-arkaplan eşiğini hesaplarken bu faktörü ölçülmesi gerekmektedir ve bu iki eşiğin eşit olabileceği durumdan kaçınmak gerekmektedir.

Hem SetHighThreshold() hem de setLowThreshold() kareden kareye ortalama mutlak farklar üzerine bir eşik kuran fonksiyonlardır. SetHighThreshold(7.0) eşiği öyle bir sabitler ki o piksel için ortalama değerin üzerinde ortalama kareden kareye mutlak farkın 7 katı olan her değer önplan olarak görülür; aynı şekilde, setLowThreshold(6.0) yine o piksel için ortalama değerin altında ortalama kareden kareye mutlak farkın 6 katı olan bir eşik sınırı çizmektedir. Pikselin ortalama değer aralığında nesneler arkaplan olarak görülmektedir. Bu eşik fonksiyonları aşağıda verilmiştir:

```
void setHighThreshold( float scale )
{
cvConvertScale( IdiffF, Iscratch, scale );
cvAdd( Iscratch, IavgF, IhiF );
cvSplit( IhiF, Ihi1, Ihi2, Ihi3, 0 );
}
void setLowThreshold( float scale )
{
cvConvertScale( IdiffF, Iscratch, scale );
cvSub( IavgF, Iscratch, IlowF );
cvSplit( IlowF, Ilow1, Ilow2, Ilow3, 0 );
}
```

setLowThreshold() ve setHighThreshold()'de IavgF'e ilişkin aralıkları eklemeden ya da çıkarmadan önce değerleri katlamak için cvConvertScale()'i kullanılmaktadır. Böylece cvSplit() aracılığıyla görüntüdeki her kanal için IhiF ve IlowF aralığını oluşturulmaktadır.

Arkaplan modelimiz yüksek ve düşük eşiklerle tamamlandığında onu görüntüyü önplan ve arkaplana ayırmak için kullanırız

Segmentasyon işlemi aşağıdaki gibi yapılır:

```
// Create a binary: 0,255 mask where 255 means foreground pixel
// I Input image, 3-channel, 8u
// Imask Mask image to be created, 1-channel 8u
//
void backgroundDiff(
IplImage *I,
IplImage *Imask
) {
cvCvtScale(I,Iscratch,1,0); // To float;
cvSplit( Iscratch, Igray1,Igray2,Igray3, 0 );
//Channel 1
//
cvInRange(Igray1,Ilow1,Ihi1,Imask);
//Channel 2
//
cvInRange(Igray2,Ilow2,Ihi2,Imaskt);
cvOr(Imask,Imaskt,Imask);
//Channel 3
//
cvInRange(Igray3,Ilow3,Ihi3,Imaskt);
cvOr(Imask,Imaskt,Imask)
//Finally, invert the results
//
cvSubRS( Imask, 255, Imask);
}
```

İlk olarak bu fonksiyon, cvCvtScale() ile girdi görüntüsü I'i kayan noktalı bir görüntüye dönüştürülmektedir. Sonrasında üç kanallı görüntüyü, cvSplit()'i kullanarak ayrı tek kanallı görüntü düzlemlerine dönüştürülmektedir. Bu renkli kanal düzlemler, ortalama arkaplan pikselinin, aralıkta olduğunda griölçekli 8-bit derinliğindeki Imaskt görüntüsünü maksimuma (255) ve aksi durumda 0'a alan cvInRange() fonksiyonu üzerinden yüksek ve düşük aralığında olup olmadıklarını anlamak için kontrol edilmektedir. Her renkli kanaldaki belirgin farklar bir önplan pikselinin kanıtı olarak görüldüğünden dolayı her renkli kanal için mantık çerçevesinde segmentasyon sonuçlarını OR ile bir maske görüntü olan Imask'e dönüştürülmektedir. Son olarak ise önplanın, değer aralığındaki değil de bunun dışındaki değerlerde olması gerektiğinden dolayı cvSubRS() kullanarak Imask'i tersine çevirириz. Maske görüntü, çıktı sonucudur.

Tamamlayabilmek için arkaplan modelini kullandıktan sonra görüntü belleğini

serbest bırakmamız gerekir:

```
void DeallocateImages()
{
cvReleaseImage( &IavgF);
cvReleaseImage( &IdiffF );
cvReleaseImage( &IprevF );
cvReleaseImage( &IhiF );
cvReleaseImage( &IlowF );
cvReleaseImage( &Ilow1 );
cvReleaseImage( &Ilow2 );
cvReleaseImage( &Ilow3 );
cvReleaseImage( &Ihi1 );
cvReleaseImage( &Ihi2 );
cvReleaseImage( &Ihi3 );
cvReleaseImage( &Iscratch );
cvReleaseImage( &Iscratch2 );
cvReleaseImage( &Igray1 );
cvReleaseImage( &Igray2 );
cvReleaseImage( &Igray3 );
cvReleaseImage( &Imaskt);
}
```

Arkaplan Ortalama Yöntemi arkaplan görüntülerini öğrenmenin ve önplan nesnelerini ayırmanın kolay bir yöntemidir. Dalgalanan bir perde ya da sallanan ağaçlar gibi dinamik bir arkaplanı olmayan görüntülerde etkili bir yöntemdir. Ayrıca bu yöntemde, aydınlatmanın oldukça sabit olduğu farz edilmektedir.

4. 7. 3. 3. Ortalama(mean), varyans ve kovaryansların toplanması

Daha önce açıkladığımız ortalama arkaplan yöntemi, bir toplama fonksiyonu olan cvAcc()'dan faydalanmaktadır. Bu fonksiyon, görüntü, kareli, çarpılmış veya ortalama görüntülerin toplanması için yardımcı fonksiyonlar grubundan biridir. Bu görüntülerden, bir görüntünün tamamı ya da bir kısmı için ortalama, varyans ve kovaryanslar gibi temel istatiksel bilgileri hesaplanabilir. Bu bölümde ise bu gruptaki diğer fonksiyonlara değinilecektir.

Herhangi bir fonksiyondaki görüntülerin hepsi aynı genişlik ve yükseklikte olmalıdır. Her fonksiyonda image, image1, ya da image2 adlı girdi görüntüleri bir ya da üç kanallı bit (8 bit) ya da kayan noktalı (32F) görüntü dizilimleri(floating-point image arrays) olabilir. sum, sqsum, or acc adlı çıktı toplama görüntüleri ya tek duyarlıklı (single-precision) (32F) ya da çifte duyarlıklı(double-precision) (64F) dizilimler olabilir. Toplama fonksiyonlarında

maske görüntüsü varsa işlemi, sadece maske piksellerin sıfır dışında olduğu yerlerle sınırlandırır.

Ortalamanın bulunması

Büyük bir görüntü dizisinden her piksel için ortalama bir değer hesaplanması için en kolay yöntem, cvAcc() kullanılarak hepsinin toplanıp ortalamayı elde etmek amacıyla toplam görüntü sayısına bölünmesidir.

```
void cvAcc(
const Cvrr* image,
CvArr* sum,
const CvArr* mask = NULL
);
```

Çoğu zaman yararlı olan bir alternatif yol ise *çalışma ortalamasını* kullanmaktır.

```
void cvRunningAvg(
const CvArr* image,
CvArr* acc,
double alpha,
const CvArr* mask = NULL
);
```

Çalışma ortalaması aşağıdaki formülle bulunur:

$$acc(x, y) = (1-\alpha)\cdot acc(x, y)+\alpha \cdot image(x, y) \text{ if } mask(x, y) \neq 0$$

α 'nın sabit bir değeri için çalışma ortalamaları cvAcc() ile toplama işleminin sonucuna denktir. Bunu anlamak için üç sayıyı(2, 3, 4) 0,5'e ayarlı αile toplamak yeterlidir. cvAcc() ile toplansaydı toplam 9, ortalama ise 3 olurdu. CvRunningAverage() ile toplasansaydı ilk toplam $0.5 \times 2 + 0.5 \times 3 = 2.5$, ve sonra üçüncü sayıyı da ekleyince $0.5 \times 2.5 + 0.5 \times 4 = 3.25$ sonucunu verirdi. İkinci sayının büyük olmasının nedeni son yapılan katkıların, öncekilerden daha önemli olmasıdır. Böyle bir çalışma ortalamasına *'tracker'* denir. α parametresi temelde bir önceki karenin gitmesi için gerekli zamanı ayarlar.

Varyansın bulunması

Kareli görüntüleri de toplayabiliriz. Bu bize bireysel piksellerin varyansını hızlı bir şekilde hesaplama olanağı sağlar.

```
void cvSquareAcc(
const CvArr* image,
CvArr* sqsum,
const CvArr* mask = NULL
```

);

Sonlu bir evrenin varyansı aşağıdaki formülle belirlenir.

$$\sigma^2 = \left(\frac{1}{N}\sum_{i=0}^{N-1} x_i^2\right) - \left(\frac{1}{N}\sum_{i=0}^{N-1} x_i\right)^2 \quad (4.\ 2.)$$

Eşitlik (4.2.) 'de x, tüm *N* örnekleri için x'in ortalamasıdır. Bu formülle tek bir geçişte hem piksel değerlerini hem de karelerini toplayabiliriz.

Kovaryansın bulunması.

Özel bir gecikme miktarı seçip mevcut görüntüyü bahsi geçen gecikmeye denk gelen geçmişten bir görüntüyle çarparak görüntülerin zamanla nasıl değiştikleri görülebilir. CvMultiplyAcc() fonksiyonu, iki görüntüyü piksel piksel çarpıp daha sonra ise sonucu acc'daki "çalışan toplama(running total)" ekler:

```
void cvMultiplyAcc(
const CvArr* image1,
const CvArr* image2,
CvArr* acc,
const CvArr* mask = NULL
);
```

Kovaryans için de varyansınkine benzer bir formül bulunmaktadır. Eşitlik (4.3.) 'de bu formül gösterilmiştir. Ayrıca bu formül, tek geçişli(single-pass) olup görüntü listesinden iki kere geçilmemesini sağlamak için standart formdan cebirsel olarak manipüle edilmiştir.

$$Cov(x,y) = \left(\frac{1}{N}\sum_{i=0}^{N-1}(x_i\, y_i)\right) - \left(\frac{1}{N}\sum_{i=0}^{N-1} x_i\right)\left(\frac{1}{N}\sum_{j=0}^{N-1} y_j\right) \quad (4.\ 3.)$$

Bizim konumuzda ise x, t zamanındaki görüntü, y ise t – d (d, gecikmedir) zamanındaki görüntüdür.

Burada bahsi geçen toplama fonksiyonlarını istatistiklere dayalı çeşitli arkaplan modelleri oluşturmak için kullanılabilir.

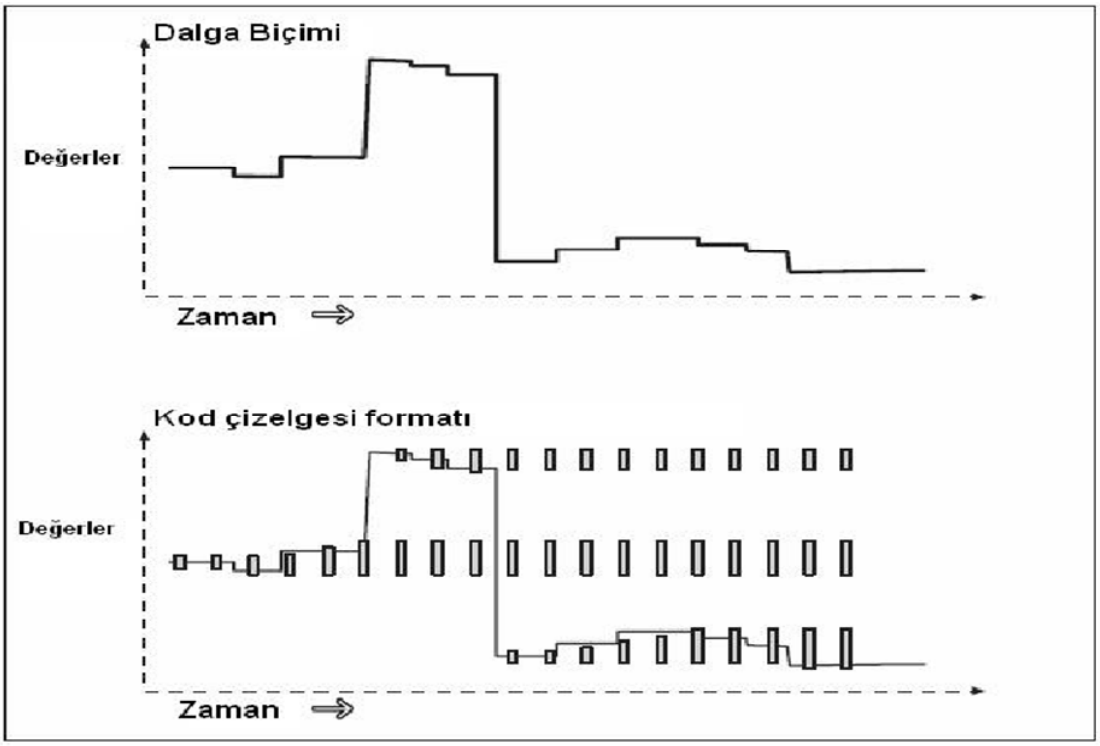
4.7.3.4. Kod Çizelgesi (CodeBook) Yöntemi

Çoğu arkaplan görüntülerinde rüzgârda sallanan ağaçlar, dönen vantilatörler, dalgalanan perdeler vb. gibi hareket eden karmaşık nesneler bulunur. Genellikle bu gibi görüntülerde geçip giden bulutlar ya da farklı ışıkların girmesine olanak sağlayan kapı ve pencereler gibi değişken olan aydınlanma durumuda vardır.

Yukarıda belirtilen dinamik olan bir görüntüde arka plan çıkarmanın yöntemi her piksele ya da her piksel grubuna bir zaman serisi modelini uyarlamaktır. Bu model türü zamansal dalgalanmaları iyi bir şekilde ele alır ancak dezavantajı ise çok miktarda bellek gerekmesidir. Önceki girdinin 2 saniyesini 30 Hz'de kullanırsak her bir piksel için 60 örnek gerekir. Her bir piksel için ortaya çıkan model ise 60 farklı adapte edilmiş *ağırlıklar* şeklinde öğrettiklerini kodlar. Genellikle 2 saniyeden çok daha uzun bir süre arkaplan istatistik bilgileri toplamamız gerekir. Bu da demektir ki benzeri yöntemler genel olarak günümüz donanım sistemlerinde kullanışsızdır.

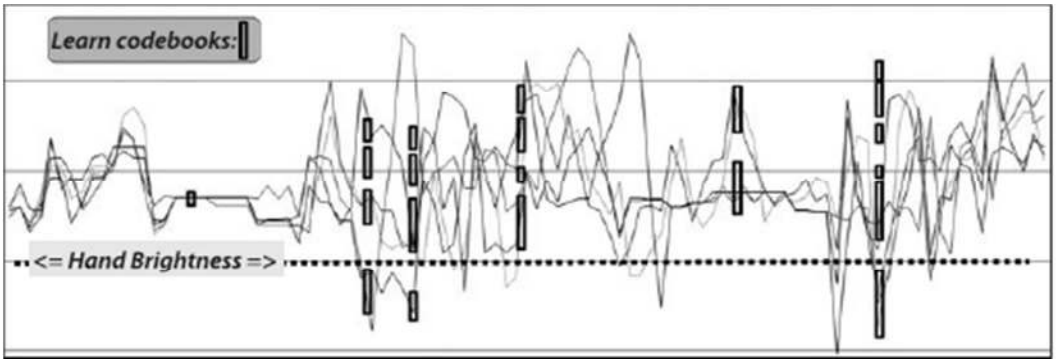
Uyarlanabilir filitrelemenin (adaptive filtering) performansına yaklaşmak için video sıkıştırma tekniklerinden ilham alıp arkaplandaki değişimleri yansıtmak için bir *kod çizelgesi* (codebook) oluşturulmaktadır. Kod çizelgesi oluşturmanın en kolay yolu ise daha önceki gözlemlenen değerlerle bir piksel için gözlemlenen yeni bir değerin kıyaslanmasıyla olmaktadır. Bu değer bir öncekine yakınsa o zaman o renk üzerinde bir sarsım(perturbation) olarak örneklendirilmektedir. Eğer bahsi geçen pikselle yakın değilse bağlanacak yeni renk grubuna katılabilir. Sonuç olarak her biri muhtemel arkaplan olarak görülen ayrı hacimi temsil eden, RGB boşluğunda kayan birkaç damla olarak düşünülebilir. Damlaların ortalama sayısı ve kovaryansla Gaus kümeleri olarak modellenebilir. “Damlaların”, renk alanımızın üç ekseninin her birinde boyutları bilinen kutulardan ibaret olduğunu bilmek gerekli bellek ve yeni gözlemlenmiş bir pikselin öğrenilen kutuların herhangi birinin içerisinde olup olmadığını belirleme işleminin hesapsal maliyeti yönünden kolaylaştırmaktadır.

Bir örnekle (Şekil 4.26.) bir kod çizelgesinin ne olduğunu açıklanmıştır. Kod çizelgesi, zamanla görülen ortak değerleri kapsamak için büyüyen kutulardan oluşur. Şekil 4.26.'nın üst panelinde zamanla bir dalga şekli görülmektedir. Alt panelde ise yeni bir değeri kapsamak için için kutular oluşur ve sonra da çevredeki değerleri kapsamak için yavaş bir şekilde büyür. Bir değer çok uzakta ise o zaman onu içini alması için yeni bir kutu oluşur ve aynı şekilde yeni değerlere doğru yavaş yavaş büyür.



Şekil 4-26. Kod çizelgeleri yoğunluk değerlerinin sınırlarını belirleyen "kutulardan" ibarettir: yeni bir değeri yeni bir kutu oluşur ve çevredeki değerleri kapsayabilmek için yavaş bir şekilde büyür; değerler çok uzakta ise o zaman yeni bir kutu oluşur.

Şekil 4.27.'de, Şekil 4.26.'daki verilerden öğrenilen altı farklı piksel için kod çizelgeleri ve kod çizelgelerinin yoğunluk ölçüleri görselleştirilmiştir. Bu kod çizelgesi yöntemi, pek çok yaprak ya da ağacın arkasındaki gökyüzünün birleşmesinden oluşmuş rüzgârda sallanan bir ağaçtaki pikseller gibi seviyeleri büyük ölçüde değişen piksellerle çalışmaktadırlar. Daha hassas olan bu modelleme yöntemiyle piksel değerleri arasında değerleri olan bir önplan nesnesini belirleyebiliriz.



Şekil 4.27. Seçilmiş altı piksel (dikey kutular olarak gösterilmiştir) ağaç yaprağındaki dalgalanmalar için öğrenilen kod çizelgesi girdilerinin yoğunluk paylarıdır. Kod çizelgesi kutuları çoklu ayrık değerleri alır. Böylece kesikli dağıtımları daha iyi bir şekilde modelleyebilen pikselleri içerir; böylece ortalama değeri arkaplan piksellerinin alabileceği değerler arasında olan bir önplan elini (noktalı çizgideki değeri) algılayabilirler. Bu durumda kod çizelgeleri, tek boyutlu olup sadece yoğunlukta meydana gelen değişiklikleri temsil eder.

Bir arkaplan modelinin kod çizelgesi yöntemiyle öğrenilmesinde her kutu üç renk ekseninin her biri üzerindeki iki maksimum ve minimum iki eşik değeri ile tanımlanır. Bahsi geçen bu kutu sınır eşikleri, yeni arkaplan örnekleri sırasıyla maksimum'un üzerinde ya da minimumun altında bir öğrenme eşiği (learnHigh ve learnLow) içerisinde olursa maksimum değer büyüyecek, minimum değer küçülecek böylece bir genişleme olacaktır. Yeni arkaplan örnekleri kutunun ve onun öğrenme eşikleri dışına çıkarsa, o zaman yeni bir kutu başlatılır. *Arkaplan farkı* modunda kabul eşikleri maxMod ve minMod vardır; bu eşik değerlerini kullanarak, bir piksel bir maksimum ya da minimum kutu sınırına yakınsa o zaman onu, kutunun içindeymiş gibi sayarız. İkinci bir runtime eşiğiyle model belirli durumlara ayarlanabilir.

Önce YUV alanında birkaç kutuyu gösterecek kod çizelgesi yapısı gerekmektedir.

```
typedef struct code_book {
code_element **cb;
int numEntries;
int t; //count every access
} codeBook;
```

numEntries değişkenine kaç tane kod çizelgesi girdisi olduğunu takip ederiz. t değişkeni başından ya da son silme işleminden beri topladığımız noktaları sayar. Aşağıda gerçek kod çizelgesi elementleri verilmiştir:

```
#define CHANNELS 3

typedef struct ce {
uchar learnHigh[CHANNELS]; //High side threshold for learning
uchar learnLow[CHANNELS]; //Low side threshold for learning
uchar max[CHANNELS]; //High side of box boundary
uchar min[CHANNELS]; //Low side of box boundary
int t_last_update; //Allow us to kill stale entries
int stale; //max negative run (longest period of inactivity)
} code_element;
```

Her kod çizelgesi girdisi kanal başına dört bit, ayrıca iki integer, ya da CHANNELS 4 +4+4 bit yer. CHANNELS'ı bir görüntüde bulunan renk kanalarının sayısına denk ya da ondan daha az olan herhangi bir pozitif sayıya ayarlayabiliriz ancak genel olarak ya 1'e (“Y”, ya da sadece parlaklık) ya da 3'e (YUV, HSV) ayarlanır. Bu yapıda her kanal için, max ve min kod çizelgesi kutusunun sınırları niteliğindedir. learnHigh[] ve learnLow[] parametreleri yeni bir kod elementinin oluşumunu tetikleyen eşiklerdir. Özellikle de değerleri her bir kanalda min – learnLow ve max + learnHigh arasında olmayan yeni bir pikselle karşılaşıldığında yeni bir kod elementi oluşturulur. Son ve eski güncelleme zamanı (t_last_update) öğrenme sırasında oluşturulan nadiren kullanılan kod çizelgesi girdilerinin silinmesini sağlamak için kullanılır. Artık dinamik arkaplanları öğrenmek için bu yapıyı kullanan fonksiyonları inceleyebiliriz.

Her piksel için bir codeBook of code_elements kullanacağız. Uzunluğu öğreneceğimiz görüntülerde bulunan piksel sayısına eşit olan kod çizelgeleri dizini gerekmektedir. Her piksel için, arkaplanda meydana gelen ilgili değişiklikleri yakalamaya yetecek kadar çok görüntü için update_codebook() fonksyonu kullanılır. Öğrenme işlemi periyodik bir biçimde güncellenebilir ve clear_stale_entries() fonksyonu ise az sayıda hareket eden önplan nesneleri varken arkaplanın öğrenilmesi için kullanılabilir. Hareket eden bir önplan kaynaklı az kullanılan “eski” girdiler silineceğinden dolayı bu mümkündür. update_codebook() arayüzü aşağıda verilmiştir:

```
////////////////////////////////////////////////////////////
// int update_codebook(uchar *p, codeBook &c, unsigned cbBounds)
// Updates the codebook entry with a new data point

//
// p Pointer to a YUV pixel
// c Codebook for this pixel
```

```
// cbBounds Learning bounds for codebook (Rule of thumb: 10)
// numChannels Number of color channels we're learning
//
// NOTES:
// cvBounds must be of length equal to numChannels
//
// RETURN
// codebook index
//
int update_codebook(
uchar* p,
codeBook& c,
unsigned* cbBounds,
int numChannels
){
unsigned int high[3],low[3];
for(n=0; n<numChannels; n++)
{
high[n] = *(p+n)+*(cbBounds+n);
if(high[n] > 255) high[n] = 255;
low[n] = *(p+n)-*(cbBounds+n);
if(low[n] < 0) low[n] = 0;
}
int matchChannel;
```

// SEE IF THIS FITS AN EXISTING CODEWORD

```
//
for(int i=0; i<c.numEntries; i++){
matchChannel = 0;
for(n=0; n<numChannels; n++){
if((c.cb[i]->learnLow[n] <= *(p+n)) &&
//Found an entry for this channel
(*(p+n) <= c.cb[i]->learnHigh[n]))
{
matchChannel++;
}
}
if(matchChannel == numChannels) //If an entry was found
{
c.cb[i]->t_last_update = c.t;
//adjust this codeword for the first channel
for(n=0; n<numChannels; n++){
if(c.cb[i]->max[n] < *(p+n))
{
c.cb[i]->max[n] = *(p+n);
}
else if(c.cb[i]->min[n] > *(p+n))
{
```

```
c.cb[i]->min[n] = *(p+n);
}
}
break;

}
}
// OVERHEAD TO TRACK POTENTIAL STALE ENTRIES
//
for(int s=0; s<c.numEntries; s++){
// Track which codebook entries are going stale:
//
int negRun = c.t - c.cb[s]->t_last_update;
if(c.cb[s]->stale < negRun) c.cb[s]->stale = negRun;
}
```

P pikseli mevcut kod çizelgesi kutularının dışına çıkınca bu fonksiyon bir kod çizelgesi girdisi ekler yada geliştirir. Piksel mevcut bir kutunun cbBounds'ında ise kutular büyür. Bir piksel kutudan cbBounds mesafesi dışındaysa eğer neyi bir kod çizelgesi kutusu oluşturulur. İlk olarak program daha sonra kullanılacak yüksek ve düşük seviyeleri ayarlar. Sonra piksel değeri p'nin kod çizelgesi "kutusunun" öğrenme sınırları içerisinde olup olmadığını kontrol etmek için her kod çizelgesi girdisine girer. Piksel tüm kanalların öğrenme sınırları içerisindeyse o zaman bu pikseli kapsaması için uygun max ya da min seviye ayarlanır ve son günceleme zamanı, şu anki zaman sayımı c.t.'ye ayarlanır. Daha sonra ise update_codebook() fonksyonu her kod çizelgesi girdisinin ne kadar sıklıkla denk gelindiğine dair istatistikleri tutar.

Eski değişkende en fazla *negatif runtime* (çalışma zamanı - O koda verilerle erişilemediği en uzun süre) mevcuttur. Eski girdileri takip ederek gürültü ya da hareket eden önplan nesnelerinden oluşmuş dolayısıyla zamanla eskiyen kod çizelgeleri silinebilir. Arkaplanın öğrenilmesine ilişkin bir sonraki safhada ise update_codebook() fonksyonu gerekirse yeni bir kod çizelgesi ekler:

```
. . . yukarıda verilen kodlamanın devamı
// ENTER A NEW CODEWORD IF NEEDED
//
if(i == c.numEntries) //if no existing codeword found, make one
{
code_element **foo = new code_element* [c.numEntries+1];
for(int ii=0; ii<c.numEntries; ii++) {
foo[ii] = c.cb[ii];
```

```
}
foo[c.numEntries] = new code_element;
if(c.numEntries) delete [] c.cb;
c.cb = foo;
for(n=0; n<numChannels; n++) {
c.cb[c.numEntries]->learnHigh[n] = high[n];
c.cb[c.numEntries]->learnLow[n] = low[n];
c.cb[c.numEntries]->max[n] = *(p+n);
c.cb[c.numEntries]->min[n] = *(p+n);
}
c.cb[c.numEntries]->t_last_update = c.t;
c.cb[c.numEntries]->stale = 0;
c.numEntries += 1;
}
```
. . . devamı aşağıda verilmiştir

Son olarak, pikseller kutu eşiklerinin dışında olmasına rağmen high(yüksek) ve low(düşük) sınırların içerisinde bulunursa update_codebook() (1 ekleyerek) yavaşça learnHigh ve learnLow öğrenme sınırlarını ayarlar.

```
. . . yukarıdaki kodlamanın devamı
// SLOWLY ADJUST LEARNING BOUNDS
//
for(n=0; n<numChannels; n++)
{
if(c.cb[i]->learnHigh[n] < high[n]) c.cb[i]->learnHigh[n] += 1;
if(c.cb[i]->learnLow[n] > low[n]) c.cb[i]->learnLow[n] -= 1;
}
return(i);
}
```

Değiştirilmiş kod çizelgesinin endeksini geri getirerek işlem sonlanır. Artık kod çizelgelerinin nasıl öğrenildiğini biliyoruz. Hareket eden önplan nesneleri varken öğrenmek ve istenmeyen gürültüler için olan kodları öğrenmekten kurtulmak için öğrenme aşamasında sadece nadiren erişilen girdileri silmek için bir yol gerekmektedir.

clear_stale_entries() fonksyonu hareket eden önplan nesneleri olsa bile arkaplanı öğrenmemizi sağlar.

```
//////////////////////////////////////////////////////////////////
//int clear_stale_entries(codeBook &c)
// During learning, after you've learned for some period of time,
// periodically call this to clear out stale codebook entries
//
// c Codebook to clean up
//
```

```
// Return
// number of entries cleared
//
int clear_stale_entries(codeBook &c){
int staleThresh = c.t>>1;
int *keep = new int [c.numEntries];
int keepCnt = 0;
// SEE WHICH CODEBOOK ENTRIES ARE TOO STALE
//
for(int i=0; i<c.numEntries; i++){
if(c.cb[i]->stale > staleThresh)
keep[i] = 0; //Mark for destruction
else
{
keep[i] = 1; //Mark to keep
keepCnt += 1;
}
}
// KEEP ONLY THE GOOD
//
c.t = 0; //Full reset on stale tracking
code_element **foo = new code_element* [keepCnt];
int k=0;
for(int ii=0; ii<c.numEntries; ii++){
if(keep[ii])
{
foo[k] = c.cb[ii];
//We have to refresh these entries for next clearStale
foo[k]->t_last_update = 0;
k++;
}
}
// CLEAN UP
//
delete [] keep;
delete [] c.cb;
c.cb = foo;
int numCleared = c.numEntries - keepCnt;
c.numEntries = keepCnt;
return(numCleared);
}
```

Program, toplam çalışma zamanı sayımı c.t.'nin yarısına denk gelmesi için sabit kodlu staleThresh parametresini tanımlayarak başlar. Bunun anlamı arkaplanı öğrenirke kod çizelgesi girdisi i'ye toplam öğrenme zamanının yarısı kadar bir süre boyunca erişilmezse o zaman i silme işlemi (keep[i] = 0) için işaretlenir. Keep[] vektörü her kod

çizelgesi girdisini işaretleyebileceğimiz şekilde ayrılır; böylece c.numEntries uzunluğunda olur. KeepCnt değişkeni kaç tane girdi tutacağımızı sayar. Hangi kod çizelgesi girdilerinin tutulacağını kaydettikten sonra keepCnt uzunluğunda olan code_element pointerlarının bir vektörüne yeni bir pointer, foo, oluşturulur ve ardından eski olmayan girdiler içerisine kopyalanır. Son olarak ise eski kod çizelgesi vektörü pointerını silip yerine yeni ve eskimeyen vektör yerleştirilir.

background_diff() fonksyonu ön plan piksellerini önceden öğrenilmiş arkaplandan ayırmak için kullanılır:

```
/////////////////////////////////////////////////////////
// uchar background_diff( uchar *p, codeBook &c,
// int minMod, int maxMod)
// Given a pixel and a codebook, determine if the pixel is
// covered by the codebook
//
// p Pixel pointer (YUV interleaved)
// c Codebook reference
// numChannels Number of channels we are testing
// maxMod Add this (possibly negative) number onto
// max level when determining if new pixel is foreground
// minMod Subract this (possibly negative) number from
// min level when determining if new pixel is foreground
//
// NOTES:
// minMod and maxMod must have length numChannels,
// e.g. 3 channels => minMod[3], maxMod[3]. There is one min and
// one max threshold per channel.
//
// Return
// 0 => background, 255 => foreground
//
uchar background_diff(
uchar* p,
codeBook& c,
int numChannels,
int* minMod,
int* maxMod
) {
int matchChannel;
// SEE IF THIS FITS AN EXISTING CODEWORD
//
 for(int i=0; i<c.numEntries; i++) {
```

```
matchChannel = 0;
for(int n=0; n<numChannels; n++) {
if((c.cb[i]->min[n] - minMod[n] <= *(p+n)) &&
(*(p+n) <= c.cb[i]->max[n] + maxMod[n])) {
matchChannel++; //Found an entry for this channel
} else {
break;
}
}
if(matchChannel == numChannels) {
break; //Found an entry that matched all channels
}
}
if(i >= c.numEntries) return(255);
return(0);
}
```

Eğer piksel kutunun içerisindeyse ve de her bir kanal için maxMod yüksek taraf ya da minMod alçak taraftaysa o zaman matchChannel sayımı arttırılacaktır. MatchChannel kanal sayısına eşit olduğunda her boyutu arar ve bir eşini bulunduğu anlaşılır. Piksel öğrenilmiş bir kutunun içerisindeyse önplanın pozitif olarak algılanması yani 255 döner; aksi halde arkaplan yani 0 döner.

update_codebook(), clear_stale_entries(), and background_diff() olmak üzere bu üç fonksiyon önplanı, öğrenilmiş arkaplandan ayırmak için kod çizelgesi yöntemini uygular.

4. 7. 3. 4. 1. Kod Çizelgesi Arkaplan Modelinin Kullanımı

CodeBook (Kod çizelgesi) arkaplan ayırma tekniğini kullanmak için genel olarak aşağıda verilen adımlar takip edilir:

1.update_codebook() fonksyonu kullanarak birkaç saniyede ya da dakikada arkaplanın temel bir modelini öğrenilir

2.clear_stale_entries() fonksyonu ile eski girdileri temizlenir.

3.Bilinen önplanı en iyi şekilde ayırmak için minMod ve maxMod eşiklerini ayarlanır.

4.Yüksek seviye bir görüntü modeli kullanılmalıdır.

5.background_diff() fonksyonu ile önplanı arkaplandan ayırmak için kullanılır.

6.Öğrenilmiş arkaplan piksellerini periyodik olarak güncellenmesi yapılmalıdır.

7.clear_stale_entries() fonksyonu ile eski kod çizelgesi girdilerini periyodik olarak ve çok daha yavaş bir frekansta temizlenir.

Genel olarak kod çizelgesi yöntemi pek çok durumda işe yarar ve uygulaması da oldukça hızlıdır. Sabah, öğlen ve akşam vaktindeki güneş ışığı gibi değişen ışık şekillerinde ya da kapalı alanda açılıp kapanan ışıklarda sınırlılıkları mevcuttur. Bu tür global değişkenlik, her biri bir durum için olmak üzere birkaç farklı kod çizelgesi modeli kullanılarak ve sonra da durumun hangi modelin aktif olduğunu kontrol etmesi sağlanarak hesaplanabilir.

4. 7. 3. 4. 2. Önplan Temizliği için Bağlantılı Bileşenler

Ortalama yöntemini kod çizelgesi yöntemiyle karşılaştırmadan önce bağlantılı bileşenler analiziyle ayrılmış görüntüleri temizlemenin yolları üzerinde durulmalıdır. Bu tür analiz gürültülü bir maske girdisini inceler; daha sonra kurtulan bileşenlerin açma işlemi sırasında kaybedilen alanlarını yeniden yapmak için *close(kapat)* morfolojik işlemi tarafından takip edilen 0'a indirmek için *open(aç)* morfolojik işlemi kullanılmaktadır. Ardından kurtulan segmentlerin "yeterince büyük" olan konturlerini bulabilir ve isteğe bağlı olarak benzeri segmentlerin tamamının istatistiğini tutulmaya başlanır. O zaman eşik üzerindeki ya en büyük ölçek çevriti ya da hepsi geri alınabilinir. Ardından bağlantılı bileşenlerde isteyebileceğiniz fonksiyonların çoğunu çalıştırılır:

- Poligon ya da dışbükey örtülerle kurtulan bileşen konturleri yaklaşıklamak veya yaklaşıklamamak
- Bir bileşen konturünün silinmemesi için büyüklüğünü ayarlamak
- Maksimum sayıda bileşen konturünü geri dönmeye ayarlamak
- Kurtulan bileşen konturlerin sınırlayıcı kutularını opsiyonel olarak geri çevirmek
- Kurtulan bileşen konturlerin merkezlerini opsiyonel olarak geri çevirmek

Bu işlemleri uygulayan bağlantılı bileşenler başlığı aşağıdaki gibidir.

```
//////////////////////////////////////////////////////////////////////
// void find_connected_components(IplImage *mask, int poly1_hull0,
// float perimScale, int *num,
// CvRect *bbs, CvPoint *centers)
// This cleans up the foreground segmentation mask derived from calls
// to backgroundDiff
//
// mask Is a grayscale (8-bit depth) "raw" mask image that
// will be cleaned up
```

```
//
// OPTIONAL PARAMETERS:
// poly1_hull0 If set, approximate connected component by
// (DEFAULT) polygon, or else convex hull (0)
// perimScale Len = image (width+height)/perimScale. If contour
// len < this, delete that contour (DEFAULT: 4)
// num Maximum number of rectangles and/or centers to
// return; on return, will contain number filled
// (DEFAULT: NULL)
// bbs Pointer to bounding box rectangle vector of
// length num. (DEFAULT SETTING: NULL)
// centers Pointer to contour centers vector of length
// num (DEFAULT: NULL)
//
void find_connected_components(
IplImage* mask,
int poly1_hull0 = 1,
float perimScale = 4,
int* num = NULL,
CvRect* bbs = NULL,
CvPoint* centers = NULL
);
```

Fonksiyon organı aşağıda listelenmiştir. Öncelikle bağlantılı kontur için bellek depolama işlemini başlatılır. Sonrasında düşük piksel gürültüsünü temizlemek için morfolojik açma ve kapama işlemini yapılır. Ardından açma işleminden kaynaklı aşındırmadan kurtulan aşınmış alanları yeniden oluşturulur. İşlem için #define ile sabit kodlu iki ek parametre gerekir. Tanımlanan değerler gayet iyi bir şekilde çalıştığından dolayı onları hiç değiştirmek istemeyeceksiniz. Bu ek parametreler bir önplan bölgesinin ne kadar basit ve morfolojik operatörlerin kaç tane iterasyon yapması gerektiğini kontrol eder; iterasyon sayısı ne kadar çoksa kapatma işleminde meydana gelen genişlemeden önce açma işleminde daha fazla aşınma olur. Fazla aşınma olursa büyük bölgelerin sınırlarını aşındırma pahasına daha büyük kabarık gürültü bölgeleri yok olur.

```
// For connected components:
// Approx.threshold - the bigger it is, the simpler is the boundary
//
#define CVCONTOUR_APPROX_LEVEL 2
// How many iterations of erosion and/or dilation there should be
//
#define CVCLOSE_ITR 1
```

Şimdiyse bağlantılı bileşen algoritmasını incelenecektir. Programın ilk kısmında

morfolojik açma ve kapama işlemleri yapılmıştır:

```
void find_connected_components(
IplImage *mask,
int poly1_hull0,
float perimScale,
int *num,
CvRect *bbs,
CvPoint *centers
) {
static CvMemStorage* mem_storage = NULL;
static CvSeq* contours = NULL;
//CLEAN UP RAW MASK
//
cvMorphologyEx( mask, mask, 0, 0, CV_MOP_OPEN, CVCLOSE_ITR );
cvMorphologyEx( mask, mask, 0, 0, CV_MOP_CLOSE, CVCLOSE_ITR );
```

Gürültü maskeden çıkarılmış ve tüm konturler bulunmuştur:

```
//FIND CONTOURS AROUND ONLY BIGGER REGIONS
//
if( mem_storage==NULL ) {
mem_storage = cvCreateMemStorage(0);
} else {
cvClearMemStorage(mem_storage);
}

CvContourScanner scanner = cvStartFindContours(
mask,
mem_storage,
sizeof(CvContour),
CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE
);
```

Çok küçük olan konturleri çıkartıp (karmaşıklığı CVCONTOUR_APPROX_LEVEL tarafından çoktan ayarlanmış olan) poligon ya da dışbükey örtülerle gerisini yaklaşıklanacaktır.

```
CvSeq* c;
int numCont = 0;
while( (c = cvFindNextContour( scanner )) != NULL ) {
double len = cvContourPerimeter( c );
// calculate perimeter len threshold:
//
double q = (mask->height + mask->width)/perimScale;
//Get rid of blob if its perimeter is too small:
```

```
//
if( len < q ) {
cvSubstituteContour( scanner, NULL );
} else {
// Smooth its edges if its large enough
//
CvSeq* c_new;
if( poly1_hull0 ) {
// Polygonal approximation
//
c_new = cvApproxPoly(
c,
sizeof(CvContour),
mem_storage,
CV_POLY_APPROX_DP,
CVCONTOUR_APPROX_LEVEL,
0
);
} else {
// Convex Hull of the segmentation
//
c_new = cvConvexHull2(
c,
mem_storage,
CV_CLOCKWISE,
1
);
}
cvSubstituteContour( scanner, c_new );
numCont++;
}
}
contours = cvEndFindContours( &scanner );
```

CV_POLY_APPROX_DP, fonksyonu Douglas-Peucker yaklaşım algoritmasının kullanılmasına neden olur ve CV_CLOCKWISE fonksyonu dışbükey örtü çevritinin varsayılan yönüdür. Tüm bu işlemler sonucunda bir kontur listesi ortaya çıkar. Konturleri tekrar maskeye çekmeden önce bazı basit renkleri tanımlamamız gerekir:

```
// Just some convenience variables
const CvScalar CVX_WHITE = CV_RGB(0xff,0xff,0xff)
const CvScalar CVX_BLACK = CV_RGB(0x00,0x00,0x00)
```

Bu tanımlamaları aşağıdaki kodda kullanırız. Bu kodda öncelikle maskeyi sıfırlayıp temiz konturleri maskeye geri çekeriz. Ayrıca kullanıcının konturlere ilişkin istatistiksel bilgi toplamak isteyip istemediğini de kontrol ederiz:

```
// PAINT THE FOUND REGIONS BACK INTO THE IMAGE
//
cvZero( mask );
IplImage *maskTemp;
// CALC CENTER OF MASS AND/OR BOUNDING RECTANGLES
//
if(num != NULL) {
//User wants to collect statistics
//
int N = *num, numFilled = 0, i=0;
CvMoments moments;
double M00, M01, M10;
maskTemp = cvCloneImage(mask);
for(i=0, c=contours; c != NULL; c = c->h_next,i++ ) {
if(i < N) {
// Only process up to *num of them
//
cvDrawContours(
maskTemp,
c,
CVX_WHITE,
CVX_WHITE,
-1,
CV_FILLED,
8
);
// Find the center of each contour
//
if(centers != NULL) {

cvMoments(maskTemp,&moments,1);
M00 = cvGetSpatialMoment(&moments,0,0);
M10 = cvGetSpatialMoment(&moments,1,0);
M01 = cvGetSpatialMoment(&moments,0,1);
centers[i].x = (int)(M10/M00);
centers[i].y = (int)(M01/M00);
}
//Bounding rectangles around blobs
//
if(bbs != NULL) {
bbs[i] = cvBoundingRect(c);
}
cvZero(maskTemp);
numFilled++;
}
// Draw filled contours into mask
//
```

```
cvDrawContours(
mask,
c,
CVX_WHITE,
CVX_WHITE,
-1,
CV_FILLED,

8
);
} //end looping over contours
*num = numFilled;
cvReleaseImage( &maskTemp);
}
```

Eğer kullanıcının maske içerisindeki bölgelerin sınırlayıcı kutularına ve merkezlerine ihtiyacı yoksa arkaplanın yeteri kadar büyük bağlantılı bileşenlerini temsil eden temizlenmiş konturler maskeye geri çekilir.

```
// ELSE JUST DRAW PROCESSED CONTOURS INTO THE MASK
//
else {
// The user doesn't want statistics, just draw the contours
//
for( c=contours; c != NULL; c = c->h_next ) {
cvDrawContours(
mask,
c,
CVX_WHITE,
CVX_BLACK,
-1,
CV_FILLED,
8
);
}
}
```

Böylelikle gürültülü ham maskelerden temiz maskelerin yapımında faydalı bir program tamamlanmış olur.

4. 8. OpenCV Arka Plan Çıkarma Yöntemlerinin Karşılaştırılması

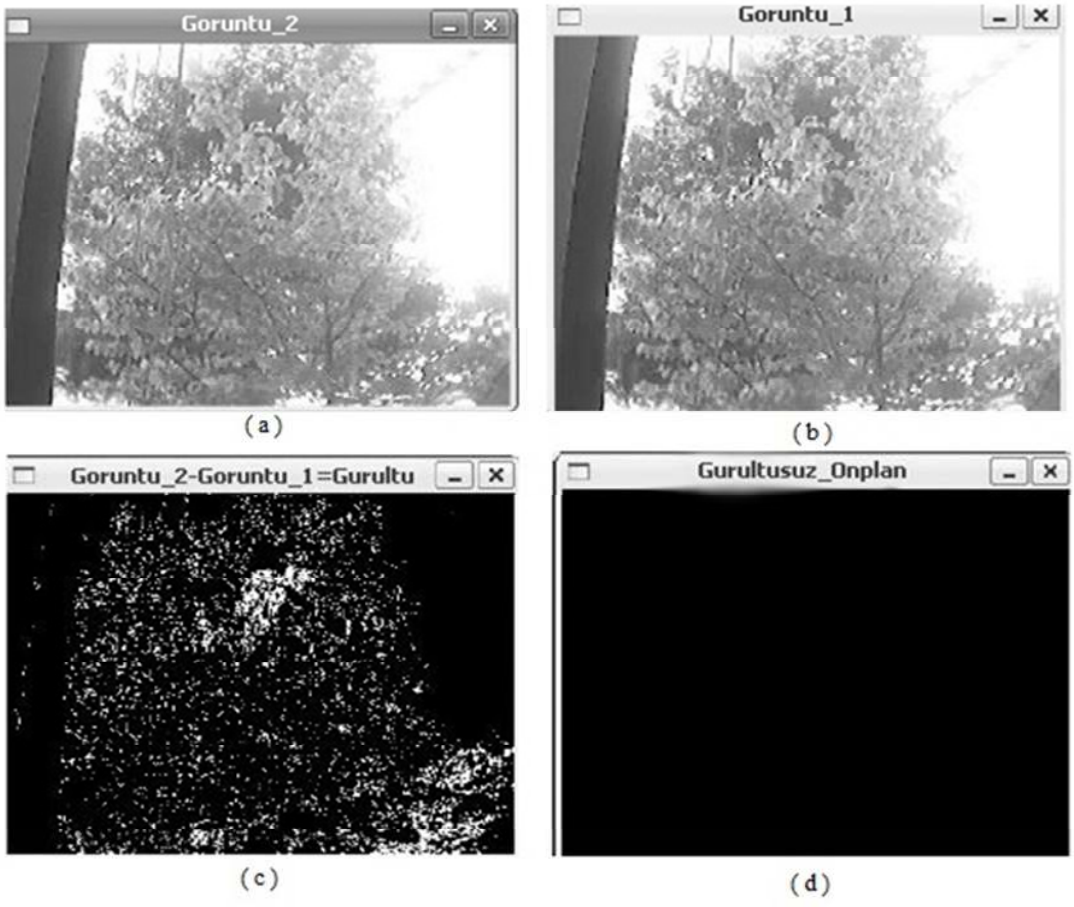
Arkaplan yöntemleri bir örnek üzerinde karşılaştırılacaktır. Pencerenin dışındaki bir ağaçı çeken bir kameranın önünden el geçtiği bir video görüntüsünün arkaplanını çerçeve farkı gibi bir teknikle elde edelim. Çerçeve farkı tekniği ile video'nun önünden geçen eli çok kolay bir şekilde bulabileceğimiz düşünülebilir. Çerçeve farkının arkasındaki temel fikir "gecikmeli" bir çerçeveden şu anki çerçeveyi çıkartıp farkı eşiklemektir.

Bir videoda bulunan ardışık çerçeveler birbirlerine oldukça benzerdir. Dolayısıyla orijinal çerçevenin ve gecikmeli çerçevenin basit bir farkını alırsak görüntüde hareket eden bir önplan nesnesi yoksa iki çerçeve arasındaki fark "gürültü" anlamına gelir.

Tüm bu gürültü olayını daha iyi anlamak için öncelikle hiçbir önplan nesnesinin olmadığı videodan bir çift çerçevearasındaki farka yani gürültüye bakılacaktır. Şekil 4.35.'te videodan tipik bir çerçeve(Goruntu_2) ve bir önceki çerçeve (Goruntu_1) görülmektedir. Resimde ayrıca 15 eşik değeriyle çerçeve farkı (Goruntu_2-Goruntu_2=Gurultu) sonuçlarıda görülmektedir. Ağacın hareket eden yapraklarında önemli ölçüde gürültü olduğunu görebilirsiniz. Yine de bağlantılı bileşenler yöntemi etrafa yayılmış gürültüyü gayet iyi bir şekilde temizleyebilir(Gurultusuz_onplan). Bu gürültüde çok fazla uzaysal korelasyon olmasını beklemek için bir sebep olmadığından dolayı bu durum hiç de şaşırtıcı sayılmaz. Dolayısıyla sinyali çok sayısıda oldukça küçük bölgeler tarafından karakterize edilir.

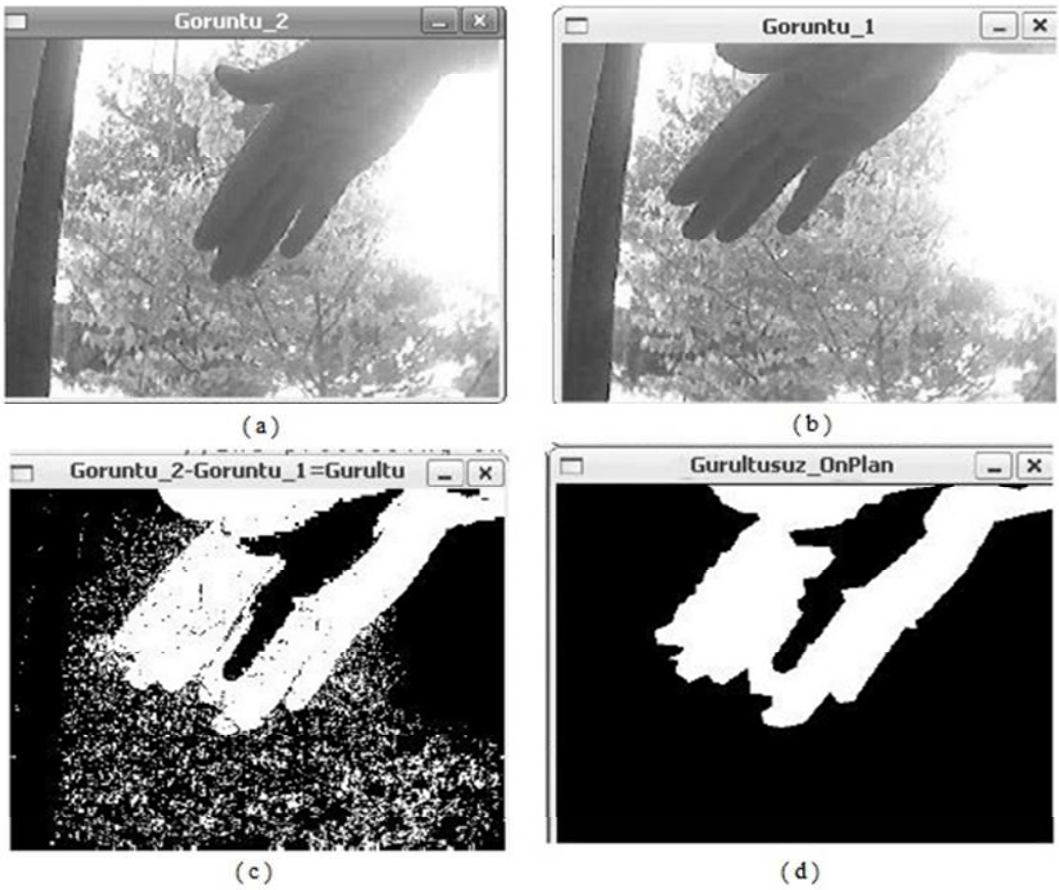
Çerçeve farkı bağlamında bir nesne genel olarak iki çerçeve arasındaki zaman farkı hızına göre "önplan" olarak tanımlanır. Genel olarak statik ya da kameraya arkaplan nesnelerinden çok daha yakın olması beklenen görüntülerde oldukça akla uygundur

Bağlantılı bileşenler için büyüklük eşiği bu boş çerçevelerde sıfır tepki vermesi için ayarlanmıştır.



Şekil 4.28. *Çerçeve farkı: şu anki (b) ve bir önceki (a) resim çerçevilerindeki arkaplanda bir ağaç sallanıyor; fark görüntüsü (c) bağlantılı bileşen yöntemiyle tamamen temizlenir(d).*

Şimdi de bir önplan nesnesi olan elin görüş alanının içinden geçtiği durumu incelenecektir. Artık elin soldan sağa doğru hareket etmesi dışında Şekil 4.29. da, Şekil 4.28'tekilere benzeyen iki çerçeve görülmektedir. Daha önce olduğu gibi çerçeve farkına verilen tepkime (c) ve bağlantılı bileşen temizleme işleminden alınan gayet iyi sonuçlar (d) ile beraber şu anki ve bir önceki kare gösterilmiştir.



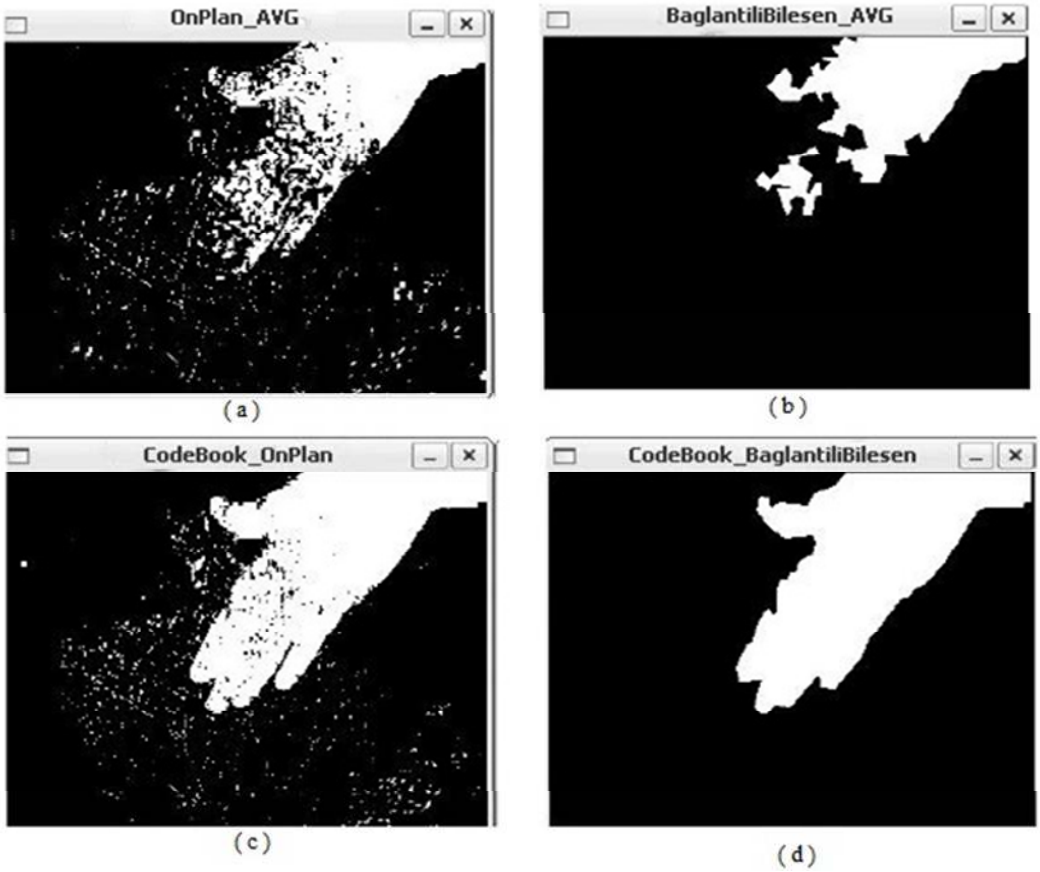
Şekil 4.29. Çerçeve farkı yöntemiyle önplan nesnesi algılama. (a) şimdiki görüntü. (b) bir önceki görüntü. Daha önce olduğu gibi çerçeve farkına verilen tepkime (c) ve bağlantılı bileşen temizleme işleminden geçmiş resim (d) ile beraber şu anki ve bir önceki kare gösterilmiştir.

Çerçeve farkının kusurlarından birini gayet net bir şekilde görebiliriz: nesnenin hareket ettiği yer ve nesnenin şu anda durduğu yeri ayırt edememektedir. Ayrıca, bindirme bölgesinde genel olarak bir boşluk oluşur. Bunun nedeni "flesh minus flesh" (en azından eşiğin altında) 0 olduğu içindir.

Dolayısıyla temizlik işlemi için bağlantılı bileşen kullanımının, arkaplan çıkarımındaki gürültüyü atmak için etkili bir teknik olduğunu görüyoruz.

Ortalama uzaklık yöntemi ve kod çizelgesi yöntemini karşılaştırmada da aynı örnek üzerinden gidilecektir. Hareket eden ağaca ilaveten bu gerçek zamanlı görüntüde bir binadan sağa doğru ve soldaki iç duvarın bazı kısımlarından gelen çok fazla parıltı olduğu söylenmelidir.

Şekil 4.30.'de üstteki ortalama fark yöntemini alttaki kod çizelgesi yöntemiyle karşılaştırılmıştır; solda ham önplan görüntüleri, sağda ise temizlenmiş bağlantılı bileşenler bulunmaktadır. Gördüğünüz üzere ortalama fark yöntemiyle geride daha kaba saba bir maske kalıp, el iki bileşene ayrılmıştır. Kod çizelgelerinin yaprak ve dalların dalgalanmalarını daha doğru bir şekillde modelleyebildiği ve böylelikle önplan el piksellerini(noktalı çizgi)arkaplan piksellerinden daha net bir şekilde tanımlayabildiği görülmüştür. Şekil 4.30., hem arkaplan modelinin daha az miktarda gürültü yarattığını hem de bağlantılı bileşenlerin oldukça doğru bir nesne konturu oluşturabileceğini doğrulamaktadır.



Şekil 4.30. Ortalama yöntemi ile kod çizelgesi yöntemi karşılaştırması. Ortalama yöntemiyle(a, b) bağlantılı bileşenlerin temizlenme işlemi parmakları (sağ üstte) devre dışı bırakır; kod çizelgesi yöntemi (c, d) ise segmentasyonda daha iyi olup net bir bağlantılı bileşen maskesi oluşturmaktadır.(a) Avg ön plan resmi. (b) Avg ön plan resminin bağlantılı bileşenlerle işlenmiş hali. (c) Kod çizelgesi ön plan resmi. (d) Bağlantılı bileşenlerle kod çizelgesi ön plan resminin işlenmiş hali

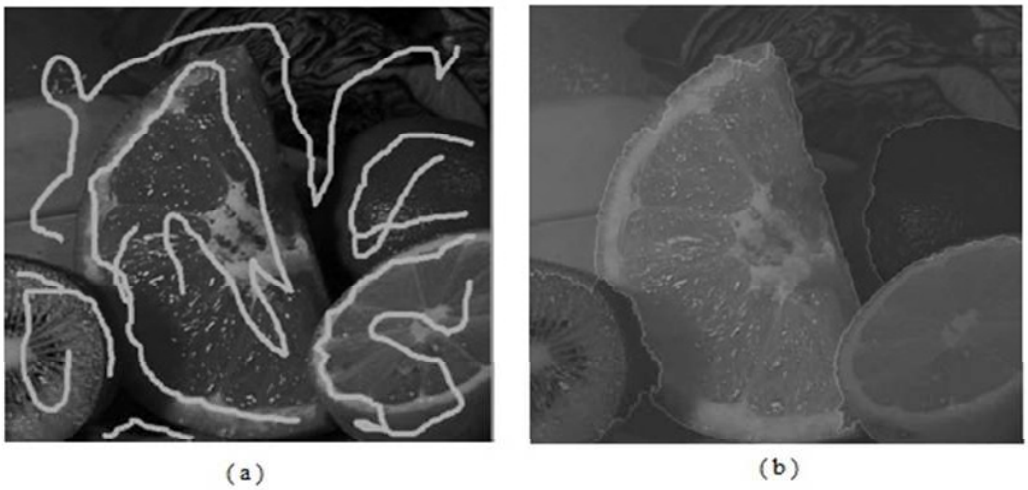
4. 9. Watershed Algoritması

Bu algoritma, bir görüntüdeki çizgileri "dağlara", benzer bölgeleri ise nesnelerin ayrılmasına yardımcı olabilecek "vadilere" dönüştürür. Watershed algoritması öncelikle yoğunluk görüntüsünün gradyanınıyla ilgilenir; böylece, hiçbir doldurma desenin olmadığı vadiler ya da *havzalar* ve görüntüde baskın çizgilerin olduğu, yüksek dağ sırtlarının kenar görevi gördüğü dağ ya da *dağ silsileleri* oluşur. Daha sonra ise kullanıcı a da algoritma tanımlı (y) noktalardan başlayarak bu bölgeler birleşene kadar havzaları ard arda su basar. Görüntü "yukarı doldururken" oluşan işaretlerden birleşen bölgeler birbirlerine bağlı olarak ayrılırlar. Sonrasında ise görüntü, ilgili işaretlenmiş bölgelere ayrılır.

Daha belirgin bir biçimde anlatmak gerekirse watershed algoritması kullanıcının bir nesnenin ya da arkaplanın bağlı olduğunun bilindiği parçaları işaretlemesini sağlar. Kullanıcı ya da algoritma, watershed algoritmasına "bu gibi noktaları bir araya toplamasını" söyler. Watershed algoritması da, işaretlenmiş bölgelerin segmentlerle bağlantılı gradyan görüntüsünde bulunan kenar tanımlı vadileri "almasını" sağlayarak görüntüyü ayırır. Şekil 4.31. de bu süreç net bir şekilde görünmektedir.

Watershed segmentasyon algoritmasının fonksiyon spesifikasyonu aşağıdaki gibidir:

```
void cvWatershed(
const CvArr* image,
CvArr* markers
);
```

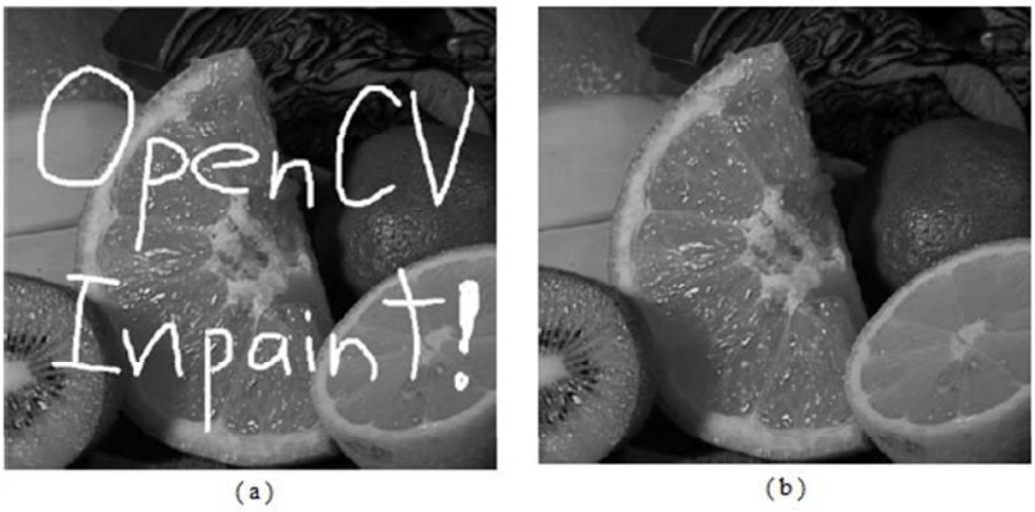


Şekil 4.31. Watershed algoritması: bir bütünü oluşturan nesneleri(a) kullanıcı işaretledikten sonra algoritma işaretlenmiş alanı segmentlerle (b) Birleştirir. (a) Kullanıcının işledipi bir bütünü oluşturan nesneler. (b) Watershed algoritması ile birleştirilmiş resim.

Buradaki görüntü bir 8 bitlik renkli görüntü olup işaretçilerin ise aynı boyutlardaki tek kanallı integer görüntüsüdür; kullanıcının (ya da algoritmanın) olumlu sayılar kullanarak bazı bölgelerin birbirlerine bağlı olduklarını gösterdiği yer haricinde işaretçilerin değeri 0'dır. Örneğin, Şekil 4.38.'in sol panelinde portakal "1", limon "2", misket limonu "3", üst arkaplan "4" vb. ile işaretlenmiş olabilir. Böylece sağdaki aynı resimde gördüğünüz segmentasyon ortaya çıkar.

4. 10. İç boyamayla Görüntü Onarımı

Çoğu zaman gürültü, görüntülere zarar verir. Video veya fotoğraftaki lenslerde toz ya da su damlaları, önceki görüntülerde ya da bir görüntünün tahrip edilen kısımlarında çizikler olabilir. *İçboyama*, zarar gören alanın kenarından renk ve doku alarak bu gibi zararları ortadan kaldırıp zarar görmüş alan içerisine yaymak için kullanılan bir yöntemdir. Bir resimden yazıların kaldırılmış halini görmek için Şekil 4.32'a bakınız.

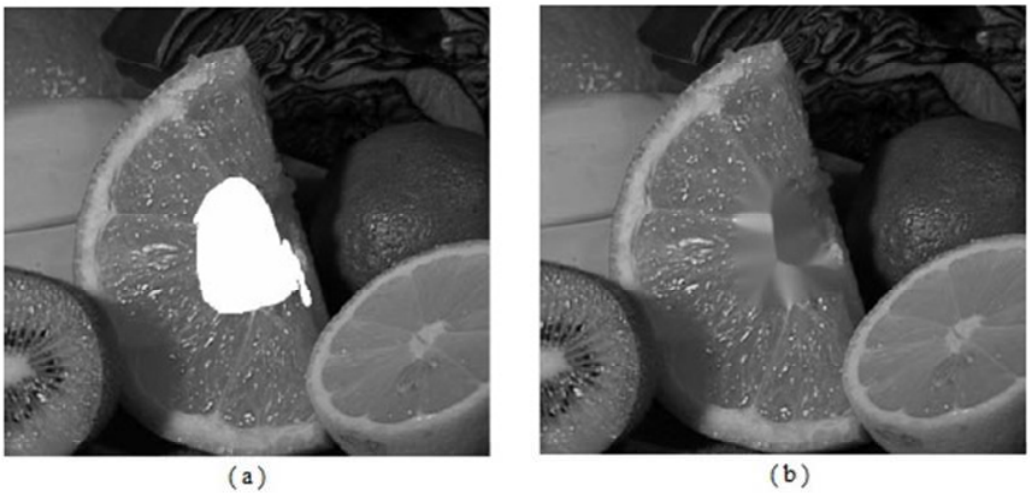


Şekil 4.32. İçboyama: (a) üstüne yazılmış metinle zarar görmüş olan görüntü. (b) Zarar görmüş görüntünün içboyamayla düzeltilmiş hali

Zarar görmüş alan çok kalın değilse ve bu alanın etrafında yeterli miktarda orijinal doku ve renk kalmışsa içboyama işlemi işe yarar. Şekil 4.32.'da zarar görmüş alan çok büyük olunca neler olduğu gösterilmiştir.

CvInpaint() prototibi aşağıdaki gibidir:

```
void cvInpaint(
const CvArr* src,
const CvArr* mask,
CvArr* dst,
double inpaintRadius,
int flags
);
```



Şekil 4.33. İçboyama işlemiyle, tamamen silinmiş dokular: (a) portakalın ortası tamamen silinmiş halde; (b)içboyamayla düzeltiliş resim.

Burada src, onarılması gereken 8 bitlik tek kanallı griölçek görüntüsü ya da üç kanallı renkli bir görüntüdür ve mask ise zarar görmüş alanların sıfırdan farklı piksellerle işaretlenmiş src ile aynı büyüklükteki 8 bitlik tek kanallı bir görüntüdür; diğer piksellerin hepsi mask'ta 0'a ayarlanmıştır. Çıktı görüntüsü hedefe (dst'ye) yazılır. Bu da src ile aynı büyüklük ve sayıda olmalıdır. İnpaintRadius ise o pikselin ortaya çıkan rengiyle çarpanlarına ayrılacak içi boyanmış her pikselin çevresindeki alandır. Şekil 4.33.'de olduğu gibi yeterince kalın içboyama yapılmış bir bölgedeki iç pikseller renklerini tamamen sınırlara yakın olan içboyama yapılmış diğer piksellerden alabilirler. Hemen hemen her zaman radyan çok büyük olursa gözle görülür bir bulanıklık olacağından dolayı 3 gibi küçük bir radyan kullanılır. Son olarak ise flags parametresiyle iki farklı içboyama yöntemiyle çalışılabilir: CV_INPAINT_NS (Navier-Stokes yöntemi) ve CV_INPAINT_TELEA (A. Telea yöntemi).

4. 11. Mean-Shift (Ortalama-Kayma) Bölütlemesi

Piramit segmentasyonunda, görüntüleri ayırmak için renk birleştirme işlemi (renklerin birbirlerine olan benzerliğine dayanan ölçek üzerinde) yapılır. Bu yaklaşım, görüntüdeki *toplam enerjinin* minimize edilmesine dayanır; burada enerji daha çok *renk benzerliği* ile tanımlanan *bağlantı kuvveti (link strength)* ile tanımlanır. Bu bölümde renk üzerinde kümelenen mean-shift'e dayalı benzer bir algoritma olan cvPyrMeanShiftFiltering()'ı ele alacağız[Comaniciu99]. cvMeanShift() fonksyonu takip etme ve hareket konusununda etkili olduğu gibi renk ya da başka bir özellik dağılımının en yüksek noktasını bulduğunu bilmemiz yeterlidir. Bu noktada mean-shift segmentasyonu alandaki renk dağılımlarının en yüksek noktalarını bulur. Anafikir, hem hareket takibinin hem de renk segmentasyon algoritmalarının, bir dağılımın modlarını (en yüksek noktaları/piklerini) bulmak için mean-shift'e bağlı olmasıdır.

Boyutları (x,y, blue(mavi), green(yeşil), red(kırmızı)) olan bir dizi çok boyutlu veri noktaları sayesinde mean-shift, alan üzerindeki bir *pencereyi* tarayarak bu alandaki yüksek yoğunluk "kümelerini" bulabilir. Ancak dikkat edin ki uzamsal değişkenlerde(x,y) renk parlaklığı alanlarındaki (blue, green, red) çok farklı alanlar olabilir. Dolayısıyla, mean shift'in farklı boyutlarda farklı pencere radyanları da hesaba katması gerekir. Bu durumda uzamsal değişkenler(spatialRadius) ve renk parlaklıkları(colorRadius) için birer radyanımızın olması gerekir. Mean shift pencereleri hareket ederken verilerdeki yüksek bir noktada birleşen pencerelerden geçmiş tüm noktalar, o en yüksek nokta y tarafından bağlanır ya da alınır. Bu durum, en yoğun noktalardan ışık yayarak, görüntünün segmentasyonunu oluşturur. Aslında segmentasyon işlemi, bir ölçek piramidinde,(cvPyrUp(), cvPyrDown()), yapılmaktadır. Böylece piramitte bulunan yüksek düzeydeki küçültülmüş görüntü halindeki renk kümelerinin(k), piramitteki düşük piramit seviyelerinde geliştirilmiş sınırları olur. CvPyrMeanShiftFiltering() için yapılan fonksiyon çağrısı aşağıdaki gibidir:

```
void cvPyrMeanShiftFiltering(
const CvArr* src,
CvArr* dst,
double spatialRadius,
double colorRadius,
int max_level = 1,
CvTermCriteria termcrit = cvTermCriteria(
CV_TERMCRIT_ITER | CV_TERMCRIT_EPS,
5,1));
```

cvPyrMeanShiftFiltering()'de bir girdi görüntüsü, src, ve bir çıktı görüntüsü, dst bulunur. İkisi de genişliği ve yüksekliği aynı olan 8 bitlik üç kanallı renk görüntüleri olmalıdır. spatialRadius ve colorRadius, mean-shift algoritmasının bir segmentasyon yapmak için rengi ve alanı nasıl ortaladığını belirler. 640x480 boyutlarındaki bir renkli görüntü için spatialRadius'ın 2'ye ve colorRadius'ın ise 40'a kurulması gayet işe yarar. Bu algoritmanın bir sonraki parametresi ise segmentasyon için hangi seviyelerde ölçek piramidi kullanılmasını istediğinizi belirleyen max_level'dir. 2 ya da 3'lük bir max_level, 640x480 boyutlarındaki bir renkli görüntü için kullanışlı olmaktadır.

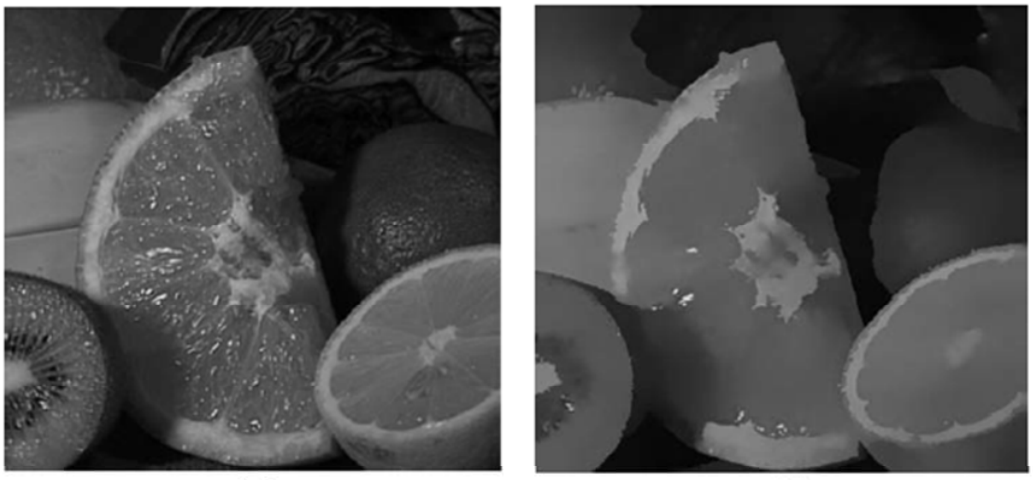
CvTermCriteria OpenCV'deki yineleyici algoritmaların tamamı için kullanılmaktadır. Eğer bu parametreyi boş bırakmak istiyorsanız mean-shift segmentasyonu fonksiyonunun sağlam varsayılan değerleri olur. Aksi halde, cvTermCriteria'da aşağıdaki yapıcı kod bulunur:

```
cvTermCriteria(
int type; // CV_TERMCRIT_ITER, CV_TERMCRIT_EPS,
int max_iter,
double epsilon
);
```

Genel olarak ihtiyacımız olan CvTermCriteria yapısını oluşturmak için cvTermCriteria() fonksiyonunu kullanırız. İlk bağımsız değişken, algoritmaya (sırasıyla) ya sabit yineleme sayısından sonra ya da birleşme ölçeği küçük bir değere ulaştığında sonlandırmak istediğimizi bildiren CV_TERMCRIT_ITER ya da CV_TERMCRIT_EPS'dir. Bir sonraki iki bağımsız değişken ise değerleri ayarlar. Bu durumda, bu ölçütlerden biri, diğeri ya da ikisi algoritmayı sonlandırabilir. Bizde iki seçeneğin de olmasının nedeni, iki sınırdan birine ulaşıldığı zaman durması için tipi CV_TERMCRIT_ITER | CV_TERMCRIT_EPS'e ayarlayabilmemizdir. max_iter parametresi, CV_TERMCRIT_ITER ayarlanırsa yineleme sayısını sınırlar. Öte yandan CV_TERMCRIT_EPS ayarlanırsa epsilon hata sınırını belirler. Elbette ki epsilon'un asıl amacı algoritmaya bağlıdır.

Şekil 4.34.'de aşağıdaki değerler kullanılarak yapılan mean-shift segmentasyonu örneği görülmektedir:

```
cvPyrMeanShiftFiltering( src, dst, 20, 40, 2);
```

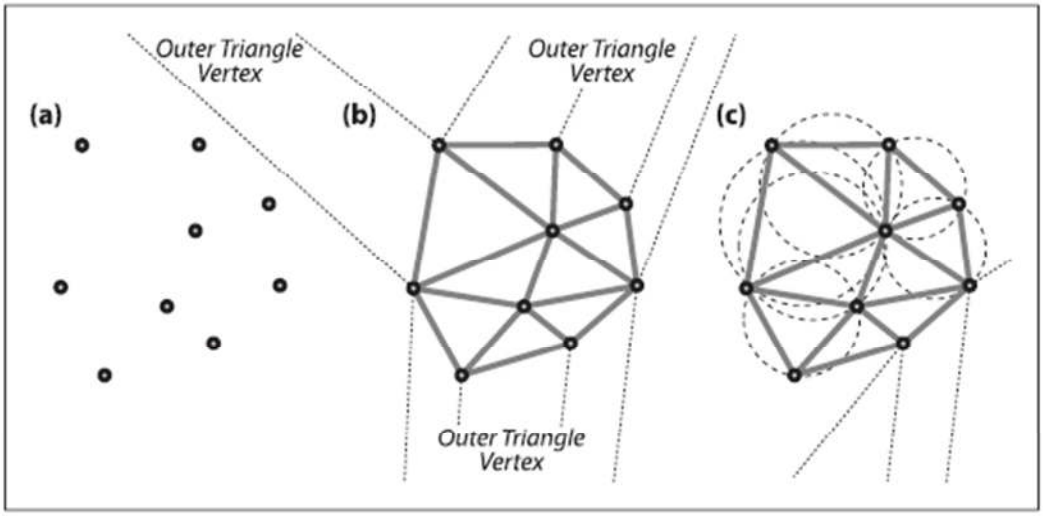


Şekil 4.34. Mean-Shift (Ortalama-Kayma) Bölütlemesi: max_level=2, spatialRadius=20, ve colorRadius=40 parametreleriyle cvPyrMeanShift Filtering() kullanılarak ölçekle yapılan mean-shift bölütlemesi; şimdi benzer alanlarda benzer değerler mevcut olup böylelikle süper piksel olarak işlenebilir. Bu da ardından yapılan işlemi hızlandırabilir.(a) ham resim. (b) işlenmiş resim

4. 12. Delaunay Üçgenlemesi, Voronoi Tesselasyonu

Delaunay üçgenlemesi 1934'te bulunan bir alandaki noktaları üçgenlemedeki tüm açıların minimum açısısının bir maksimum değerde olması için üçgen gruplarla birleştirmek için kullanılan bir tekniktir. Bu, şu anlama gelir: Delaunay üçgenlemesi, noktaları üçgenlerken uzun ve çok zayıf olan üçgenlerden uzak durmaya çalışır. Konunun özünü anlamak için Şekil 4.43.'e bakınız. Resimde üçgenleme işlemi, her üçgenin tepe noktalarında bulunan noktalara yerleştirilen her çemberde başka hiçbir tepe noktası bulunmayacak şekilde uygulanmıştır. Buna *çevrel çember özelliği* denir (resimdeki panel c).

Hesapsal verimlilik sağlamak için Delaunay algoritması, algoritmanın başladığı çok uzakta bir dış sınırlayıcı üçgen yaratır. Şekil 4.35. (b), tepe noktasına doğru giden noktalı çizgilerle hayali dış üçgenini temsil eder. Şekil 4.35. (c)'de ise gerçek verilerin iki dış noktasını hayali dış üçgenin tepe noktalarından birine bağlayan çemberlerden biri de dâhil olmak üzere çevrel çember özelliğine birkaç örnek verilmiştir.



Şekil 4.35. Delaunay üçgenlemesi: (a) noktalar; (b) dıştaki sınırlayıcı üçgene trailer'lı noktalarla yapılan Delaunay üçgenlemesi; (c) çevrel çember özelliğini gösteren örnek çemberler

Artık Delaunay üçgenlemesini hesaplamak için çok sayıda algoritma geliştirilmiştir; bu algoritmalardan bazıları oldukça etkili olup karmaşık iç ayrıntıları bulunmaktadır. Daha basit algoritmalardan birinin ana fikri aşağıdaki gibidir:

1.Dış üçgen eklenmelidir ve tepe noktalarından birinde başlanmalıdır.

2.İç nokta eklenmelidir; sonra da o noktayı içeren tüm üçgenlerin çevrel çemberlerini aranıp üçgenlemeler kaldırılmalıdır.

3.Yeni kaldırılmış üçgenlemelerin çevrel çemberlerindeki yeni nokta da dâhil olmak üzere grafik yeniden üçgenlenmelidir.

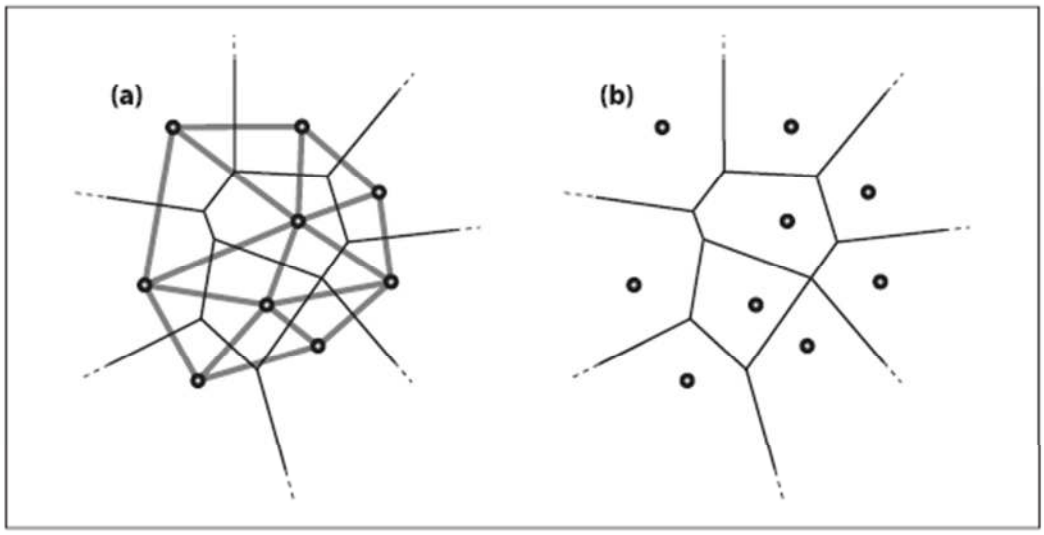
4.Eklenecek nokta kalmayana kadar ikinci adıma dönülmelidir.

Bu algoritmanın karmaşıklık derecesi veri noktaları sayısında $O(n^2)$'dir. En iyi algoritmalar (ortalama olarak) $O(n \log \log n)$ kadar düşüktür.

Algoritma hayali bir dış üçgenle başlamıştır. Dolayısıyla gerçek dış noktaların hepsi aslında o üçgenin tepe noktalarından iki tanesine bağlıdır. Çevrel çember özelliğini gereği gerçek dış noktalardan iki tanesine ve bir hayali dış tepe noktasına takılı çemberlerde başka iç nokta yoktur. Bu demekter ki bir bilgisayar, hangi noktaların üç tane hayali dış tepe nokasına bağlı olduğuna bakarak tam olarak hangi gerçek noktaların bir dizi noktanın dış kısmını oluşturduğunu doğrudan inceleyebilir. Diğer bir deyişle, bir Delaunay üçgenlemesi yapıldıktan neredeyse hemen sonra bir dizi noktanın dışbükey örtüsünü bulabiliriz.

Ayrıca noktalar arasındaki bölgeyi hangi parçanın aldığını yani hangi koordinatların Delaunay tepe noktalarının her birine en yakın komşu olduklarını bulabiliriz. Dolayısıyla

orijinal noktalarına Delaunay üçgenlemesi yapılarak yeni bir noktaya en yakın komşu çabucak bulunabilir. Buna, Voronoi tesselasyonu denir (Şekil 4.44.'e bakınız). Bu tesselasyon, Delaunay üçgenlemesinin dual görüntüsüdür çünkü Delaunay çizgileri mevcut noktalar arasındaki mesafeyi belirler ve böylece Voronoi çizgileri, noktalar arasında eşit mesafe tutmak için Delaunay çizgileriyle nerede kesişmesi gerektiklerini "anlarlar". Dışbükey örtü ve en yakın komşunun hesaplanmasıyla bu iki yöntem, nokta ve nokta dizilerinin kümelenip sınıflandırılması için önemli temel işlemlerdir.



Şekil 4.36. Delaunay üçgenlemesi: Belirli bir Voronoi hücresindeki tüm noktaların Delaunay noktasına diğer Delaunay noktasından daha yakında olduğu Voronoi tesselasyonu: (a) normal çizgilerdeki ilgili Voronoi tesselasyonuyla koyu renkteki Delaunay üçgenlemesi; (b) her Delaunay noktasının etrafında bulunan Voronoi hücreleri

Üç boyutta bir nesneyi işlediğimiz zaman görüntü projeksiyonuyla o nesnenin bir 2D görüntüsünü oluşturabilir ve böylece bu nesneyi analiz edip tanımlamak ve/veya gerçek bir nesneyle karşılaştırmak için 2D Delaunay üçgenlemesini kullanılabilir. Dolayısıyla Delaunay üçgenlemesi bilgisayar görüntüsü ve bilgisayar grafiği arasında köprü görevi görür. OpenCV'nin Delaunay üçgenlemesini sadece iki boyutta yapmaktadır. 2D Delaunay üçgenlemesi, hareket takip etme, nesne tanımlaması ya da iki farklı kamera arasındaki görüntülerin eşleştirilmesi için nesne ya da görüntü üzerinde hatların uzamsal düzenlemesini yapmak amacıyla genelde bilgisayar görüntüsünde kullanılır (stereo görüntülerden derinliğin çıkarılmasında olduğu gibi).

4. 12. 1. Bir Delaunay ya da Voronoi Altbiriminin Oluşturulması

Öncelikle Delaunay altbirimini bellekte depolamak için yer gereklidir. Ayrıca bir de bir adet dış sınırlayıcı kutu gerekmektedir. Hesaplamaları hızlandırmak için algoritma dikdörtgen şeklindeki bir sınırlayıcı kutunun dışarısına yerleştirilmiş bir hayali dış üçgenle çalışır. Bunu kurmak için noktaların bir 600x600 görüntünün içerisinde olması gerekir:

```
// STORAGE AND STRUCTURE FOR DELAUNAY SUBDIVISION
//
CvRect rect = { 0, 0, 600, 600 }; //Our outer bounding box
CvMemStorage* storage; //Storage for the Delaunay subdivsion
storage = cvCreateMemStorage(0); //Initialize the storage
CvSubdiv2D* subdiv; //The subdivision itself
subdiv = init_delaunay( storage, rect); //See this function below
```

Kod OpenCV fonksiyonundan ziyade birkaç OpenCV programının paket hali olan init_delaunay()'ı başlatır:

```
//INITIALIZATION CONVENIENCE FUNCTION FOR DELAUNAY
SUBDIVISION
//
CvSubdiv2D* init_delaunay(
CvMemStorage* storage,
CvRect rect
) {
CvSubdiv2D* subdiv;
subdiv = cvCreateSubdiv2D(
CV_SEQ_KIND_SUBDIV2D,
sizeof(*subdiv),
sizeof(CvSubdiv2DPoint),
sizeof(CvQuadEdge2D),
storage
);
cvInitSubdivDelaunay2D( subdiv, rect ); //rect sets the bounds
return subdiv;
}
```

Bu noktalar kayan(float) türden olmalıdır, 32f:

```
CvPoint2D32f fp; //This is our point holder
for( i = 0; i < as_many_points_as_you_want; i++ ) {
// However you want to set points
//
fp = your_32f_point_list[i];
cvSubdivDelaunay2DInsert( subdiv, fp );
}
```

Uygun makroyu, *cxtypes.h*'deki cvPoint2D32f(double x, double y) ya da

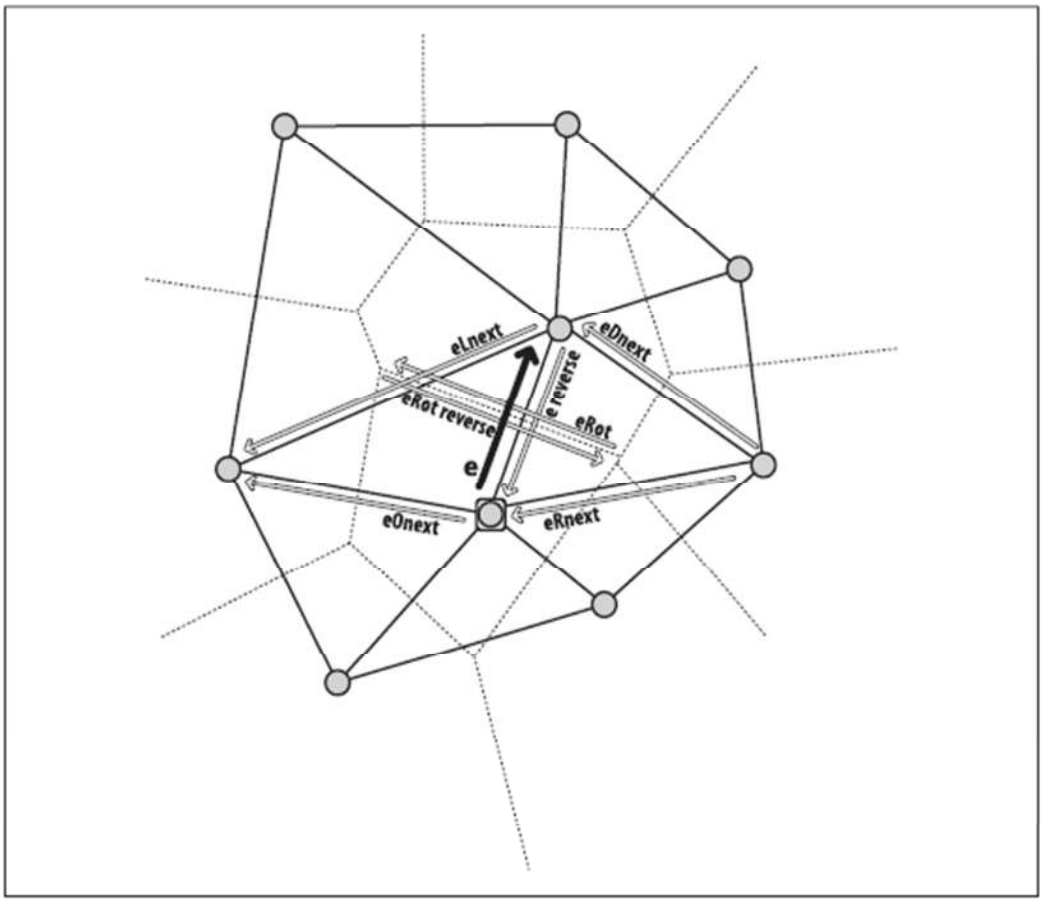
cvPointTo32f (CvPoint point), kullanarak integer noktalarını 32f'e çevrilebilir. Delaunay üçgenlemesi elde etmek için noktalara girebildiğimize göre aşağıda verilen iki komut ile ilgili Voronoi tesselasyonunu ayarlayıp temizleyebiliriz:

```
cvCalcSubdivVoronoi2D( subdiv ); // Fill out Voronoi data in subdiv
cvClearSubdivVoronoi2D( subdiv ); // Clear the Voronoi from subdiv
```

Her iki fonksiyonda da subdiv, CvSubdiv2D tipidir. Şimdi iki boyutlu nokta dizinlerinin Delaunay altbirimlerini oluşturup ardından onlara Voronoi tesselasyonlarını ekleyip temizlenebilir. Bu yapıların içerisindeki iyi şeylere subdiv'de kenardan uca ya da kenardan kenara geçerek yapabiliriz. Daha sonra altbirimdeki ilk kenar ya da noktaları iki farklı yoldan bir tanesini kullanarak bulunmaktadır. Birincisi bir kenarı ya da tepe noktayı belirlemek için bir dış noktayı kullanarak, ikincisi bir nokta ya da kenar zincirinden geçerek alt birimdeki iki kenar bulunabilir.

4. 12. 2. Delaunay Altbirimlerinin (Delaunay Subdivisions) Navigasyonu

Şekil 4.37'de görüldüğü üzere bir altbirim grafiği üzerinde hareket etmek için kullanacağımız iki veri yapısı birleştirilmiştir. CvQuadEdge2D yapısında iki Delaunay ve iki Voronoi noktasından ve onların bağlantılı kenarlarından oluşan bir grup bulunmaktadır; Şekil 4.37 ‘ye bakınız. CvSubdiv2DPoint yapısı, Şekil 4.38 de gösterildiği gibi bağlantılı tepe noktasıyla beraber Delaunay kenarını içermektedir. Resimden sonraki kodda dörtkenarlı yapı tanımlanmıştır.



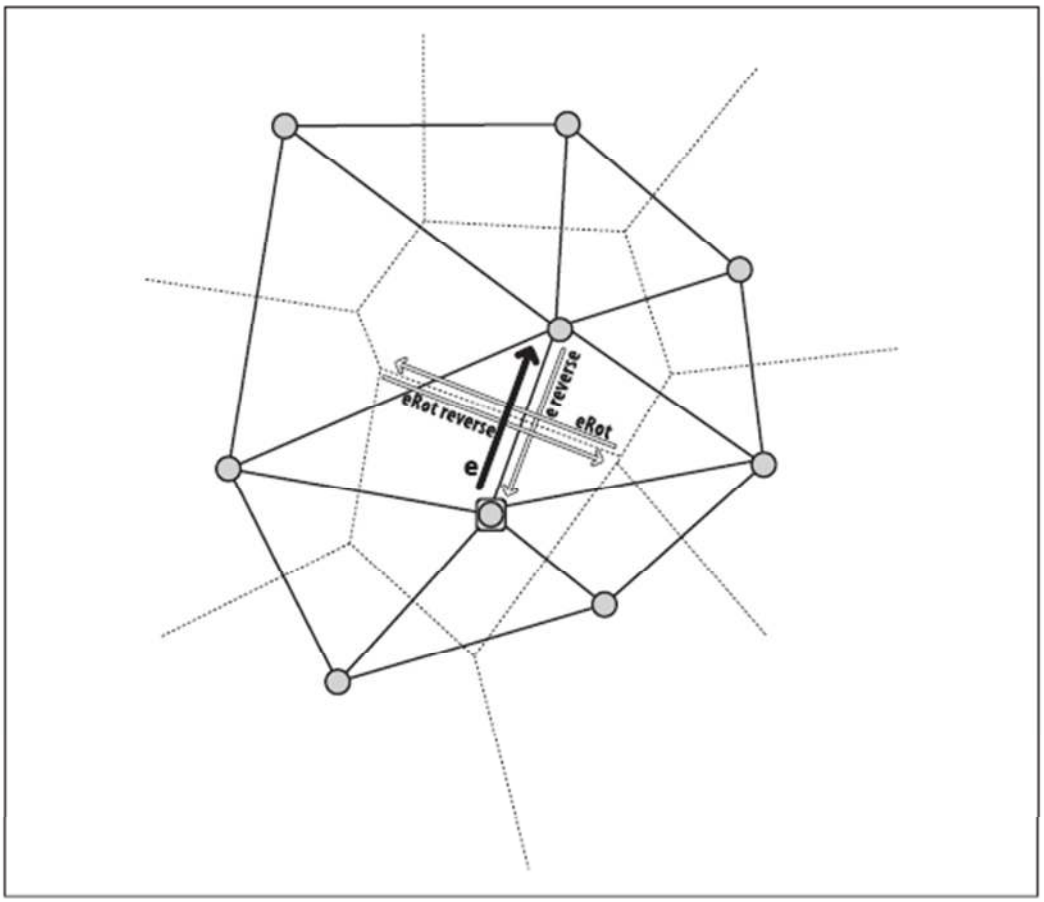
Şekil 4.37. “e” olarak gösterilmiş belirli bir kenara ve onun tepe noktasına(kareyle gösterilmiştir) bağlı kenarlar

```
// Edges themselves are encoded in long integers. The lower two bits
// are its index (0..3) and upper bits are the quad-edge pointer.
//
typedef long CvSubdiv2DEdge;
// quad-edge structure fields:
//
#define CV_QUADEDGE2D_FIELDS() /
int flags; /
struct CvSubdiv2DPoint* pt[4]; /
CvSubdiv2DEdge next[4];
typedef struct CvQuadEdge2D {
CV_QUADEDGE2D_FIELDS()
} CvQuadEdge2D;
```

Delaunay altbirim noktası ve bağlantılı kenar yapısını aşağıdaki kod verir:

```
#define CV_SUBDIV2D_POINT_FIELDS() /
int flags; /
CvSubdiv2DEdge first; //*The edge “e” in the figures.*/
```

```
CvPoint2D32f pt;
#define CV_SUBDIV2D_VIRTUAL_POINT_FLAG (1 << 30)
typedef struct CvSubdiv2DPoint
{
CV_SUBDIV2D_POINT_FIELDS()
}
CvSubdiv2DPoint;
```

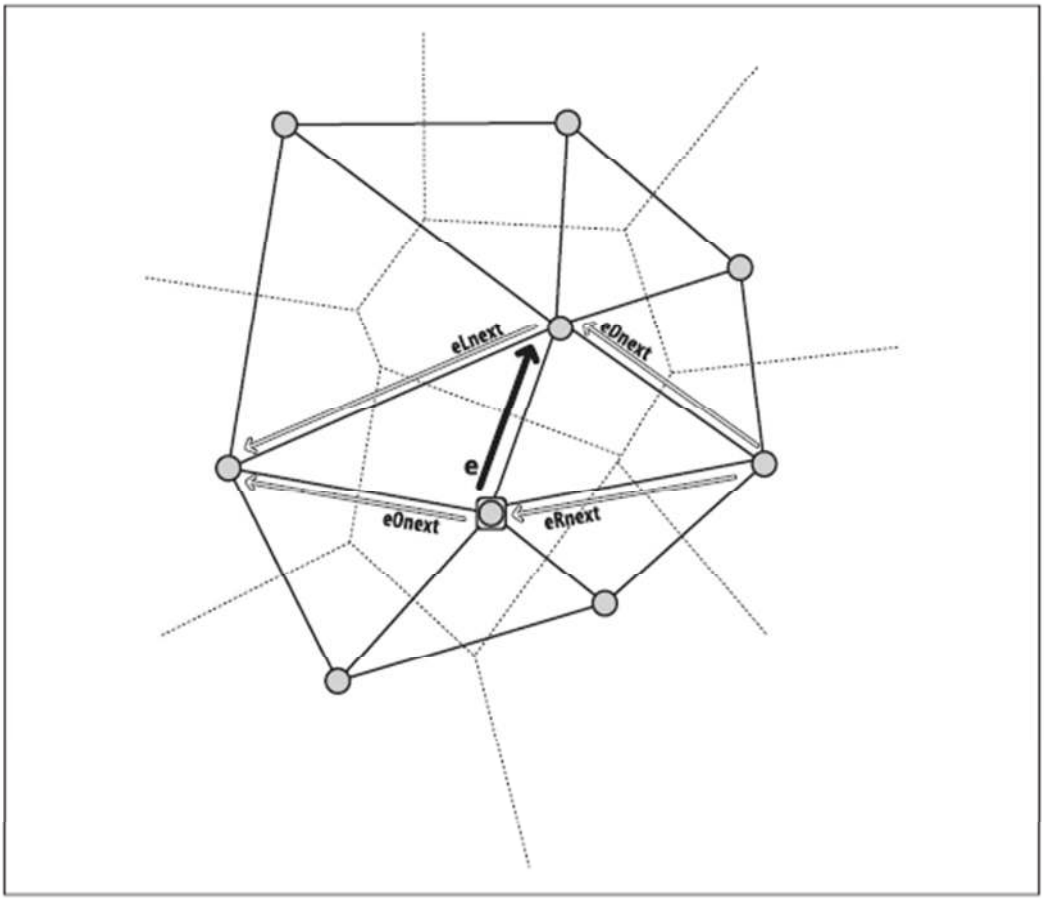


Şekil 4.38. e tarafından erişilebilecek dörtlü kenarlar(squad edges) arasında hemen bağlantılı Voronoi kenar ve noktaları hem de (bağlantılı tepe noktalarıyla birlikte)Delaunay kenarı ve onun ters yüzü bulunmaktadır.

4. 12. 3. Kenarlarda yürümek

Şekil 4.39'da da görüldüğü üzere aşağıdaki kodu kullanarak dörtlü kenarlar(quad-edges) etrafında dolaşabiliriz.

```
CvSubdiv2DEdge cvSubdiv2DRotateEdge(
CvSubdiv2DEdge edge,
int type
);
```



Şekil 4.39. cvSubdiv2DGetEdge() aracılığıyla erişilebilecek bir CvSubdiv2DPoint tepe noktası ve diğer bağlantılı kenarlarla beraber ona bağlı e kenarı

edge (kenar) göz önüne alındığında aşağıda verilen bağımsız değişkenlerden birini alan type (tür) parametresini kullanarak bir sonraki kenara ulaşabiliriz:

- 0, giriş kenarı (eğer e giriş kenarındaysa resimdeki e)
- 1, döner kenar (eRot)
- 2, ters çevrilmiş kenar (ters çevrilmiş e)
- 3, ters çevrilmiş döner kenar (ters çevrilmiş eRot)

Şekil 4.47'yi kaynak alırsak, aşağıdaki kodlamayı kullanarak Delaunay grafiğinin etrafından geçebiliriz.

```
CvSubdiv2DEdge cvSubdiv2DGetEdge(
CvSubdiv2DEdge edge,
CvNextEdgeType type
);
```

```
#define cvSubdiv2DNextEdge( edge ) /
cvSubdiv2DGetEdge( /
edge, /
CV_NEXT_AROUND_ORG /
)
```

Burada type aşağıdaki adımlardan bir tanesini belirler:

CV_NEXT_AROUND_ORG

Kenar orijin noktasından ileri (e, giriş kenarı ise Resim 9-17'deki eOnext)

CV_NEXT_AROUND_DST

Kenar tepe noktasından ileri (eDnext)

CV_PREV_AROUND_ORG

Kenar orijin noktasından geri (ters çevrilmiş eRnext)

CV_PREV_AROUND_DST

Kenar destinasyonundan geri (ters çevrilmiş eLnext)

CV_NEXT_AROUND_LEFT

Sol taraftan ileri (eLnext)

CV_NEXT_AROUND_RIGHT

Sağ taraftan ileri (eRnext)

CV_PREV_AROUND_LEFT

Sol taraftan geri (ters çevrilmiş eOnext)

CV_PREV_AROUND_RIGHT

Sağ taraftan geri (ters çevrilmiş eDnext)

Bir tepe noktasıyla bağlantılı bir kenar söz konusu olduğunda o tepe noktasından diğer tüm kenarları bulmak için uygun makroyu cvSubdiv2DNextEdge(edge)'yi, kullanabiliriz.

Diğer önemli hareket türleri CV_NEXT_AROUND_LEFT ve CV_NEXT_AROUND_RIGHT 'tır. Bunları, bir Delaunay kenarındaysak Delaunay üçgeni etrafında dolaşmak ya da bir Voronoi kenarındaysak Voronoi hücresinin etrafında dolaşmak

için kullanılmaktadır.

4. 12. 4. Kenarlardan çıkan noktalar

Gerçek noktaların Delaunay ya da Voronoi tepe noktalarından nasıl tekrar alınacağını bilinmelidir. Her Delaunay ya da Voronoi kenarının kendisine bağlı iki noktası vardır: org, orijin noktası, ve dst, hedef noktası. Bu noktaları, aşağıdaki kodlama sayesinde kolayca edinebilinir:

```
CvSubdiv2DPoint* cvSubdiv2DEdgeOrg( CvSubdiv2DEdge edge );
CvSubdiv2DPoint* cvSubdiv2DEdgeDst( CvSubdiv2DEdge edge );
```

4. 12. 5. Kenar ya da tepe noktasının belirlenmesi için bir dış noktanın kullanılması

Kenar ya da tepe noktasının belirlenmesi için bir dış noktanın kullanılması gelişigüzel bir noktayla başlayıp o noktanın yerini altbirimde belirlemektir. Bu noktanın daha önceden üçgenlenmiş olması gerekmez; herhangi bir nokta olabilir. CvSubdiv2DLocate() fonksiyonu, bu noktanın içine düştüğü üçgen ya da Voronoi tarafın (istendiği takdirde) bir kenar ya da tepe noktasını doldurur.

```
CvSubdiv2DPointLocation cvSubdiv2DLocate(
CvSubdiv2D* subdiv,
CvPoint2D32f pt,
CvSubdiv2DEdge* edge,
CvSubdiv2DPoint** vertex = NULL
);
```

En yakın kenar ya da tepe noktası bunlar değildir; üçgende ya da yüzeyde olmaları yeterlidir. Bu fonksiyonun çıkışta döndürülen değeri, aşağıda gösterildiği gibi noktanın nereye indiğini belirtir.

CV_PTLOC_INSIDE

Nokta, bir yüzeye düşer; kenar, yüzeyin kenarlarından birini içine alır.

CV_PTLOC_ON_EDGE

Nokta, kenarın üzerine düşer; kenar, bu kenarı içine alır.

CV_PTLOC_VERTEX

Nokta, altbirim tepe noktalarından biriyle çakışır; tepe noktası bir tepe noktası pointer'ını içine alır.

CV_PTLOC_OUTSIDE_RECT

Nokta, altbirim kaynak dikdörtgeninin dışarısındadır; fonksiyon geri döner ve hiçbir pointer doldurulmaz.

CV_PTLOC_ERROR

Girdi bağımsız değişkenlerinden biri geçersizdir.

4. 12. 6. Bir dizi nokta ve kenar içinden dolaşmak

Bir dizi noktadan oluşan bir Delaunay altbirimi oluşturduğumuzda ilk üç nokta ve kenar, hayali dış sınırlayıcı üçgenin tepe noktalarını ve kenarlarını oluşturur. Oradan itibaren gerçek veri noktalarının dışbükey örtüsünü oluşturan dış nokta ve kenarlara direk erişim sağlanabilir. Bir Delaunay altbirimi (adına subdiv diyelim) oluşturduğumuz zaman ilgili Voronoi tesselasyonunu hesaplamak için cvCalcSubdivVoronoi2D(subdiv)'de çağrılmalıdır. O zaman aşağıdaki kodlamayı kullanarak dış sınırlayıcı üçgenin üç tepe noktasına erişilebilinir.

```
CvSubdiv2DPoint* outer_vtx[3];
for( i = 0; i < 3; i++ ) {
outer_vtx[i] =
(CvSubdiv2DPoint*)cvGetSeqElem( (CvSeq*)subdiv, I );
}
```

Benzer şekilde dış sınırlayıcı üçgenin üç kenarını da elde edebiliriz:

```
CvQuadEdge2D* outer_qedges[3];
for( i = 0; i < 3; i++ ) {
outer_qedges[i] =
(CvQuadEdge2D*)cvGetSeqElem( (CvSeq*)(my_subdiv->edges), I );
}
```

4. 12. 7. Dışbükey örtüdeki sınırlayıcı üçgen ya da kenarların tanımlanması ve örtüde hareket etmek

cvInitSubdivDelaunay2D(subdiv, rect) fonksyonu Delaunay üçgenlemesini başlatmak için bir sınırlayıcı dikdörtgen olan rect'i kullanmaktadır. Bu durumda aşağıdaki durumlar geçerlidir.

1.Hem orijin hem de hedef noktalarının rect sınırları dışında olduğu bir kenarda iseniz, o zaman o kenar altbirimin hayali sınırlayıcı üçgenindedir.

2.Bir nokta rect sınırları içerisinde bir noktanın da bu sınırların dışarısında olduğu bir kenarda isenizi o zaman sınırlarda bulunan nokta dizinin dışbükey örtüsündedir; bu örtü üzerindeki her nokta hayali dış sınırlayıcı üçgenin iki tepe noktasına bağlı olup bu iki kenar birbiri ardına oluşur.

İkinci durumdan hedef(dst) noktası sınırlar içerisinde olan ilk kenarın üzerine geçebilmek için cvSubdiv2DNextEdge()'i kullanılabilinir. Dışbükey örtüsü üzerine gelince o zaman örtünün etrafında aşağıda belirtildiği gibi hareket edebilirsiniz.

1.Dışbükey örtünün çevresini dolaşana kadar cvSubdiv2DRotateEdge(CvSubdiv2DEdge edge, 0) aracılığıyla örtü üzerinde bulunan bir sonraki kenara gidilir.

2.cvSubdiv2DNextEdge() makrosuna iki çağrı daha yapılmasıyla örtünün bir sonraki kenarına ulaşılır. Tekrar Adım 1'e dönülür.

Örnekler

Bir Delaunay üçgeni kenarları etrafında dolaşmak için cvSubdiv2DLocate()'i kullanılabilir:

```
void locate_point(
CvSubdiv2D* subdiv,
CvPoint2D32f fp,
IplImage* img,
CvScalar active_color
) {
CvSubdiv2DEdge e;
CvSubdiv2DEdge e0 = 0;
CvSubdiv2DPoint* p = 0;
cvSubdiv2DLocate( subdiv, fp, &e0, &p );
```

```
if( e0 ) {
e = e0;
do // Always 3 edges -- this is a triangulation, after all.
{
// [Insert your code here]
//
// Do something with e ...
e = cvSubdiv2DGetEdge(e,CV_NEXT_AROUND_LEFT);
}
while( e != e0 );
}
}
```

Aşağıdaki kodlamayı da kullanarak bir girdi noktasına en yakın nokta bulunabilir:

```
CvSubdiv2DPoint* cvFindNearestPoint2D(
CvSubdiv2D* subdiv,
CvPoint2D32f pt
);
```

cvSubdiv2DLocate()'in aksine cvFindNearestPoint2D() Delaunay altbirimindeki en yakın tepe noktasını geri getirmektedir. Bu noktanın, üzerine indiği yüzey ya da üçgende olması gerekmemektedir.

Aynı şekilde aşağıdaki kodlamayı da kullanarak bir Voronoi yüzeyi etrafında dolaşılabilinir:

```
void draw_subdiv_facet(
IplImage *img,
CvSubdiv2DEdge edge
) {
CvSubdiv2DEdge t = edge;
int i, count = 0;
CvPoint* buf = 0;
// Count number of edges in facet
do{
count++;
t = cvSubdiv2DGetEdge( t, CV_NEXT_AROUND_LEFT );
} while (t != edge );
// Gather points
//
buf = (CvPoint*)malloc( count * sizeof(buf[0]))
t = edge;
for( i = 0; i < count; i++ ) {
CvSubdiv2DPoint* pt = cvSubdiv2DEdgeOrg( t );
if( !pt ) break;
```

```
buf[i] = cvPoint( cvRound(pt->pt.x), cvRound(pt->pt.y));
t = cvSubdiv2DGetEdge( t, CV_NEXT_AROUND_LEFT );
}
// Around we go
//
if( i == count ){
CvSubdiv2DPoint* pt = cvSubdiv2DEdgeDst(

cvSubdiv2DRotateEdge( edge, 1 ));
cvFillConvexPoly( img, buf, count,
CV_RGB(rand()&255,rand()&255,rand()&255), CV_AA, 0 );
cvPolyLine( img, &buf, &count, 1, 1, CV_RGB(0,0,0),
1, CV_AA, 0);
draw_subdiv_point( img, pt->pt, CV_RGB(0,0,0));
}
free( buf );
}
```

Son olarak altbirim yapısına erişmenin bir diğer yolu ise bir kenar dizini içerisinden geçmek için bir CvSeqReader kullanmaktır. Tüm Delaunay ya da Voronoi kenarlarının içinden nasıl geçileceği aşağıda gösterilmiştir:

```
void visit_edges( CvSubdiv2D* subdiv){
CvSeqReader reader; //Sequence reader
int i, total = subdiv->edges->total; //edge count
int elem_size = subdiv->edges->elem_size; //edge size
cvStartReadSeq( (CvSeq*)(subdiv->edges), &reader, 0 );
cvCalcSubdivVoronoi2D( subdiv ); //Make sure Voronoi exists
for( i = 0; i < total; i++ ) {
CvQuadEdge2D* edge = (CvQuadEdge2D*)(reader.ptr);
if( CV_IS_SET_ELEM( edge )) {
// Do something with Voronoi and Delaunay edges ...
//
CvSubdiv2DEdge voronoi_edge = (CvSubdiv2DEdge)edge + 1;
CvSubdiv2DEdge delaunay_edge = (CvSubdiv2DEdge)edge;
// …OR WE COULD FOCUS EXCLUSIVELY ON VORONOI…
// left
//
voronoi_edge = cvSubdiv2DRotateEdge( edge, 1 );
// right
//
voronoi_edge = cvSubdiv2DRotateEdge( edge, 3 );
}
CV_NEXT_SEQ_ELEM( elem_size, reader );
}
}
```

En sonunda uygun bir satıriçi makroyla(inline macro) işlemi bitirebiliriz: bir

Delaunay üçgeninin tepe noktalarını bulduğumuz zaman aşağıdaki kodlamayı kullanarak alanını bulabiliriz:

```
double cvTriangleArea(
CvPoint2D32f a,
CvPoint2D32f b,
CvPoint2D32f c
)
```

5. İNSAN BİLGİSAYAR ETKİLEŞİMİ UYGULAMASI

Gerçek zamanlı görüntü işlemede video analiz teknikleri önemli bir rol oynamaktadır. Video analizindeki önemli asamalardan biri videolardaki hareketli nesnelerin tespit edilmesidir [35]. Gerçek zamanlı görüntüde hareketli nesnelerin bulunması için görüntüde ön plan arka plan ayrımı gerçeklestirilir.

Hareketli nesnelerin bulunması savunma ve güvenlik sistemleri, trafik kontrolü, otopark kontrolü, yoldaki araçların hareketleri ve bu hareketlerin ürettiği alarmlar; sollama, yakın takip, kaza, asırı sürat, direksiyon başında uyuma tespiti v.b çalışmalarda ön plan arka plan ayrımı kullanılmaktadır. Hareket halindeki bir nesnenin takip edilmesi kontrol uygulamalarında önemlidir. Örneğin seyir halindeki araçları otomatik yönlendirme, hareketli nesnenin takip etme ve izleme gibi uygulamalar hareket yönü, hareket hızının bulunmasına ihtiyaç duyar. Gerçek zamanlı görüntülerde ön ve arka plan ayrımı bilgisayarlı görü uygulamalarının ilk basamağını oluşturur.

Son yıllarda yeni teknolojilerin de gelişmesiyle insan-bilgisayar etkileşimi önem kazanmaya başlamıştır. Bu durum, insanların bilgisayarla etkileşime geçmesinin bir yolu gibi görünen el tanıma sistemlerini ön plana çıkarmaktadır. Sözü edilen bu etkileşimin gerçekleşebilmesi için yapılması gereken, el şekillerinin tanınmasıdır [39].

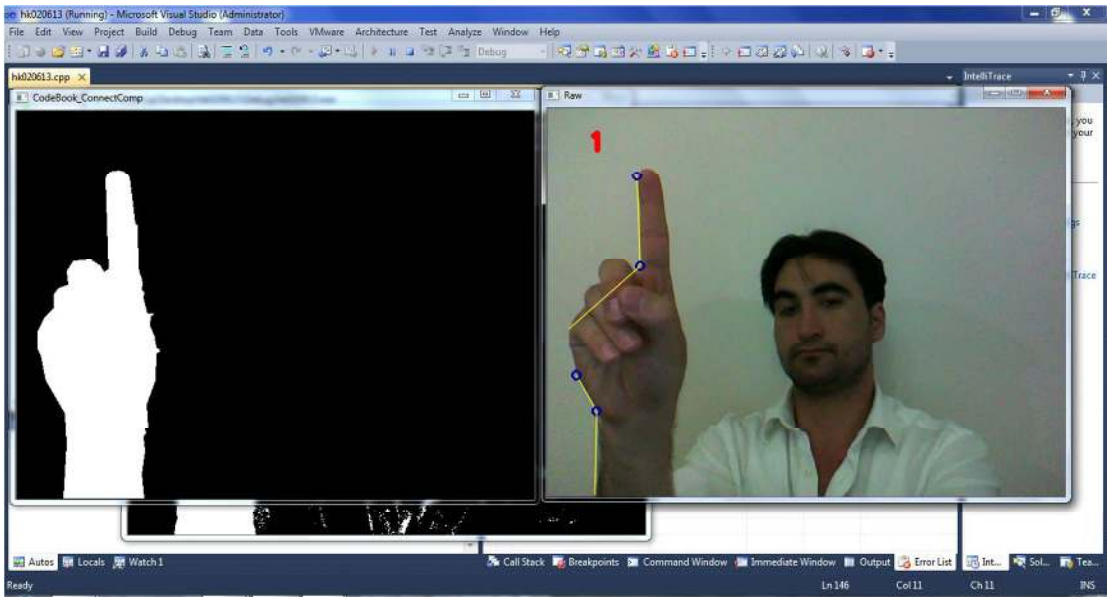
Tez çalışması kapsamında yapılan çalışma insan bilgisayar etkileşimi üzerinedir. Yapılan uygulamada web cam aracılığı ile alınan gerçek zamanlı görüntüden el işaretleri ile sayılar gösterilerek bilgisayarın bu el işaretlerini tanıması sağlanmıştır. Bu çalışma geliştirilerek engelli bir insanın akıllı sistemlerle döşenmiş bir evde el hareketleri ile komut vermesi gibi pek çok alanda kullanılabilinir.

5. 1. Uygulamanın Geliştirilmesi

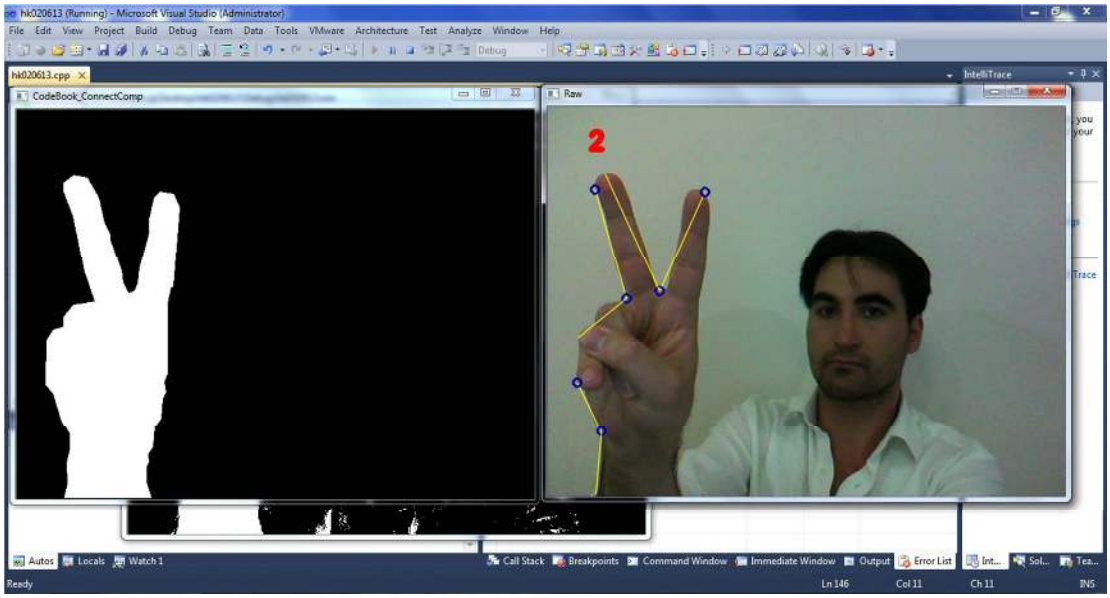
Yapılan çalışma; gerçek zamanlı görüntüde OpenCV görüntü işleme aracı kullanılarak Visual Studio 2010 ortamında C++ programlama dilinde gerçekleştirilmiştir. Gerçek zamanlı uygulamalarda hız önemli olduğu için, makine diline en yakın programlama dili olması ve OpenCV'nin programlandığı uyumlu bir dil olması C++ programlama dilinin seçiminin nedenidir.

Çalışma gerçekleştirilirken Intel® Core™2 Duo Processor P8800 @ 2.53 GHz işlemcili, Windows 7 Ultimate SP1 32 bit işletim sistemine sahip, 4 GB Ram bellek, 1 GB harici ekran kartlı, 1.3 Mega Pixel web kameralı kişisel bilgisayar kullanılmıştır.

Bilgisayara ait web kamerasından gerçek zamanlı görüntü alınmıştır. Arka plan çıkarma metodlarından kod çizelgesi (CodeBook) metodu ile el bölgesi görüntüden çıkarılmıştır. Çıkarılan ön plan nesnesinden el hareketi tanımlanmıştır. El işareti ile rakam gösterileceğinden parmak uçlarını bulmak için Convex Hull kullanılmıştır. Dış bükey el bölgesi çerçevelenmiştir. Parmaklar arasındaki çukurlar için dış bükey noksanlığı kullanılmıştır. Dış bükey noksanlık sayısı katlanmamış parmakların sayısını vermiştir.

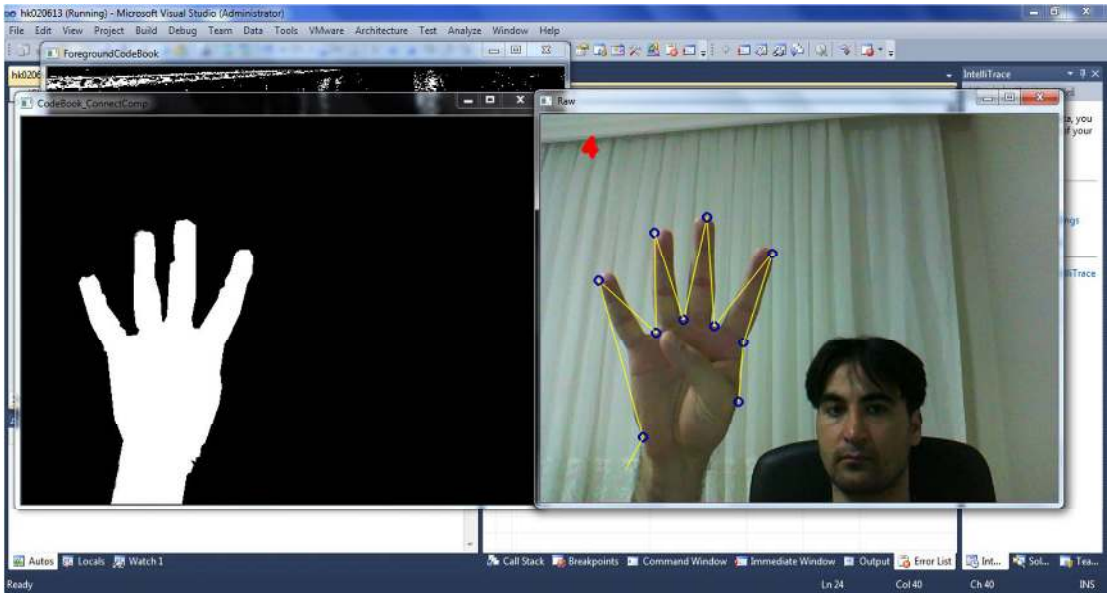


Şekil 5.1. Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 1 komutu verme

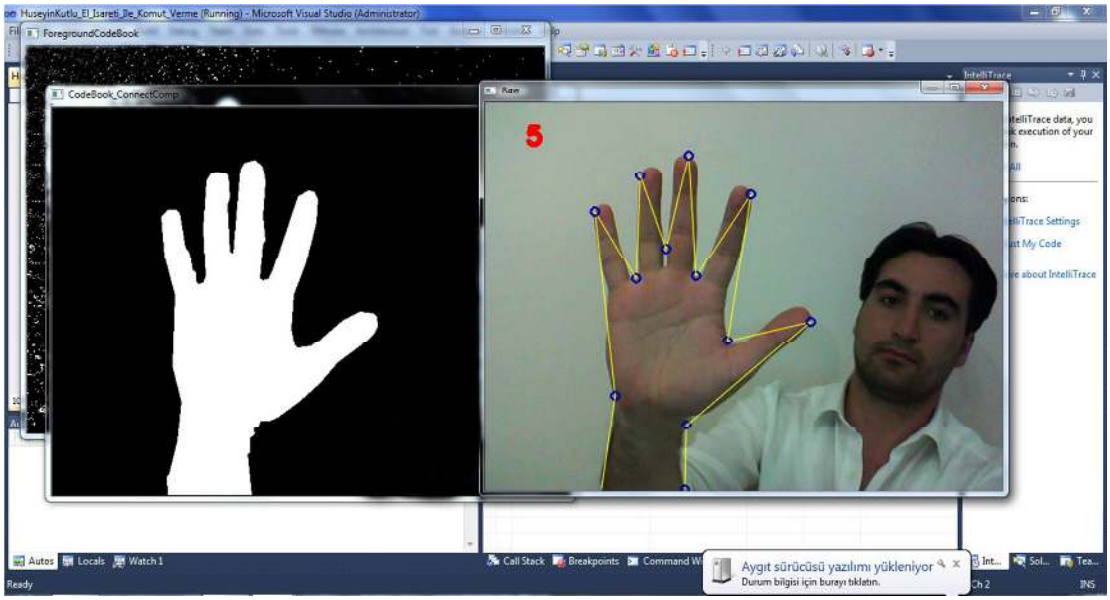


Şekil 5.2. Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 2 komutu verme

Şekil 5.3. Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 3 komutu verme

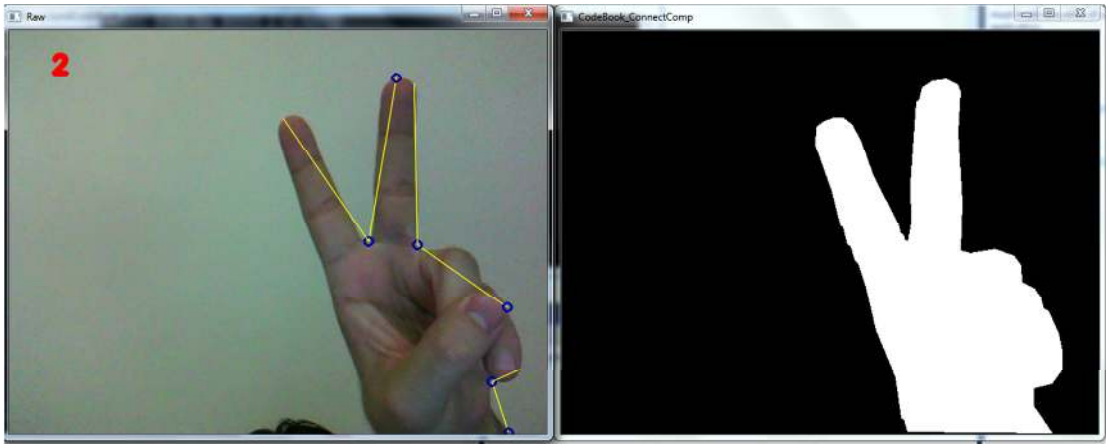


Şekil 5.4. Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 4 komutu verme

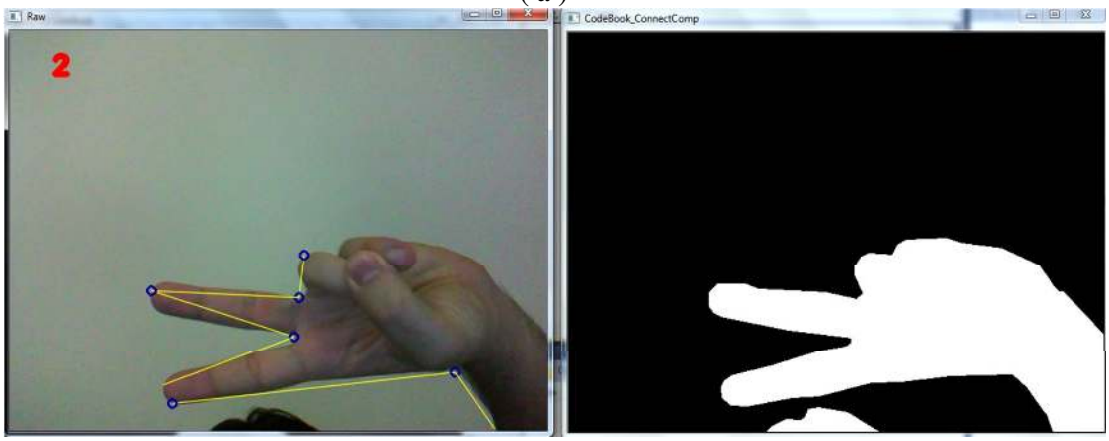


Şekil 5.5. Web Cam'den alınan gerçek zamanlı görüntüde el işareti ile bilgisayara 5 komutu verme

Uygulama esnasında ön plan nesnesi olan el yatay ve dikey doğrultuda hareket ettiğinde uygulama doğru sonuçlar vermiştir. Ön plan nesnesi olan el takip edilmiştir. Uygulamada sağ ve sol eller ile ayrı ayrı ve birlikte komut verildiğinde uygulama el işaretini tanımıştır. El işareti yatay ve dikey açılarla verildiğinde uygulamanın doğru sonuçlar ürettiği görülmüştür.



(a)



(b)

Şekil 5.6. Web Cam'den alınan gerçek zamanlı iki görüntüde el işaretinin hareket ettirilmesi: Görüntüde el işareti hareket ettirilerek dikey konumdan (a), yatay konuma (b) getirilmiştir ve uygulamanın doğru sonuç üretmeye devam ettiği görülmüştür.

6. SONUÇLAR

Gerçek zamanlı görüntülerde önemli aşamalardan biri hareketli nesnelerin tespit edilmesidir. Ön plan arka plan ayrımı gerçek zamanlı görüntülerde hareketli nesnelerin tespit edilmesinde yaygın olarak kullanınılan bir yöntemdir. Gerçek zamanlı görüntülerin uygulama alanlarının ve görüntü ortamlarının değişmesi ön plan arka plan ayrımında çeşitli algoritmaların gelişmesine sebep olmuştur. Bu algoritmaların bazıları bellek kullanımı açısından iyi olmalarına karşın algoritma karmaşıklıkları ve işlem gücü yüksektir. Bazı algoritmaların ise karmaşıklığı az olmasına rağmen bellek kullanımı yüksektir.

Gerçek zamanlı görüntülerde arka plan çıkarımında görüntünün alındığı ortamın yani görüntünün arka planının durağan olması veya hareketli olması veya peryodik hareketlilik göstermesi ön plan arka plan ayrımı algoritmalarının seçimini etkilemektedir. Değişen arka plana adaptasyonu uzun süren algoritmalar olduğu gibi kısa süren algoritmalar da mevcuttur. Gerçek zamanlı uygulamalarda ön plan arka plan çıkarımında algoritma seçiminde algoritma hızı ve sağlamlığı önemlidir ve uygulamaya göre doğru algoritma seçilmelidir.

Bu yüksek lisans tez çalışmasında web camdan alınan gerçek zamanlı bir görüntüde el hareketleri ile sayılar işaret edilmiş ve bilgisayarın bu işaretleri komut olarak alması sağlanmıştır. Bu işlem için gerçek zamanlı görüntülerde ön plan arka plan ayrımında kullanılan bazı yöntemler uygulanmıştır. Ön plan arka plan ayrımında kullanılan yöntemlerden bazıları incelenmiştir. Açık kaynak kodlu görüntü işleme aracı olan OpenCV anlatılmış, OpenCV'nin kurulumu, bazı fonsyonları anlatılmış, temel uygulamalar yapılmış ve tez çalışmamız kapsamında kullanılan OpenCV ön plan arka plan ayrım fonksyonları anlatılmıştır.

Tez çalışması kapsamında yapılan uygulamada ön plan, arka plan ayrımında gerçek zamana uyumluluğu, etkin bellek kullanımı, doğru sonuç üretmesi dolayısı ile kod çizelgesi (CodeBook) yöntemi OpenCV aracı ile kullanılmıştır. Kod çizelgesi yönteminin diğer yöntemlere göre daha başarılı olduğu görülmüştür. Yapılan uygulamada kulanılan "Finding Contour and Convex Hull" yöntemi ile el işareti ile gösterilen bir ile on arası işaretler gibi sınırlı sayıda el işareti tanınmıştır.

Bilgisayar sistemlerinin gelişmesi ile beraber hareketli nesnelerin tespitinde kullanılan yaygın bir yöntem olan ön plan arka plan ayrımı algoritmaları dahada gelişecektir. Örneğin renkli görüntülerde ön plan arka plan ayrımı algoritma olarak daha kolay olmasına rağmen bellek yetersizliği ve işlem süresi uzunluğu dolayısıyla tercih

edilmemektedir. Bir başka örnek ise derinlik bilgisinden ön plan arka plan ayrımı yapılmasıdır. Buradaki dezavantaj çift kamera kullanım zorunluluğu ve işlem yükünün iki katına çıkmasıdır.

KAYNAKLAR

[1] Duyuncu,İ., OpenCV, http://ihsanduyuncu.blogspot.com/search/label/opencv 12.04.2012

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

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

[4] OpenCV, http://derindelimavi.blogspot.com/search/label/OpenCv, 07.06.2012.

[5] Kameradan Hareket Algılama, http://e-atolye.net/2012/02/26/opencv-motion-finderkameradan-hareket algilama/ , 07.06.2012.

[6] Bradski, G. and Kaehler, A., "Learning OpenCV: Computer Vision with the OpenCV Library", O'Reilly Media, Amerika Birleşik Devletleri, 16-17 (2008).

[7] Erişti, E., Görüntü İşlemede Yeni Bir Soluk, OPENCV,

[8] OpenCV, http://hasanfehmi.wordpress.com/category/opencv/, 07.06.2012.

[9] Ayan, T., OpenCV İle Kamera Üzerine Noktalar Yerleştirme, http://tuna-ayan.com/opencv-ile-kamera-uzerine-noktalar-yerlestirme/ ,07.06.2012.

[10] OpenCV Görüntü Takibi, http://www.mcu-turkey.com/?p=17997, 07.06.2012.

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

[12] Yılmaz İ., "Renk Sistemleri, renk uzayları ve donusumler", Selcuk Universitesi Jeodezi ve Fotogrametri Muhendisliği Oğretiminde 30. Yıl Sempozyumu, Konya 2002.

[13] Jack K. ,"Video Demystified", LLH Technology Publishing,1995.

[14] McIvor, A.M., Background subtraction techniques. Proc. Image Vision Comput., 1: 155-163. ,2000

[15] Cheung, S. S. and Kamath, C.. Robust techniques for background subtraction inurban traffic video, Visual Communications and Image Processing 2004, Vol.5308, No. 1., pp. 881-892. 2004

[16] Deane, S., A Comparison of Background Subtraction Techniques COMP6009 Individual Research Project, 2007

[17] Benton, S., Background Subtraction , http://www.dspdesignline.com, 07/06/2013

[18] Cutler R., and Davis, L. 1998. View-based detection, in Proceedings Fourteenth International Conference on Pattern Recognition, 1, pp. 495-500, (Brisbane,

Australia)

[19] Cucchiara, R., Grana, C., Neri, G., Piccardi, M. and Prati, A. 2002. “The Sakbot System for Moving Object Detection and Tracking,” 10th ACM International Conference on Multimedia, ACM Press, France, pp. 223-226

[20] Toyama, K., Krumm, J.,Brumitt, B. and Meyers, B. Wallflower: Principles and Practiceof Background Maintenance, Seventh International Conference on Computer Vision, Kerkyra, Greece, pp. 255-261, IEEE Computer Society Press

[21] Elgammal, A., Harwood, D. and Davis L. 1999. Non-parametric model for background subtraction, in Proceedings of IEEE ICCV'99 Frame-rate workshop

[22] Oliver, N., Rosario, B. and Pentland, A. 2000. A Bayesian computer vision system for modeling human interactions,"IEEE Transactions on Pattern Analysis and Machine Intelligence 22, pp. 831-843

[23] KaewTraKulPong, P.and Bowden, R. 2001. An improved adaptive background mixture model for real-time tracking with shadow detection," in Proceedings of the 2nd European Workshop on Advanced Video-Based Surveillance Systems

[24] Magee, D. R. 2002. Tracking multiple vehicles using foreground, background, and motion models," in Proceedings of the Statistical Methods in Video Processing Workshop, pp. 7-12, (Copenhagen, Denmark)

[25] Ridder, C., Munkelt, O. and Kirchner, H. 1995. Adaptive Background Estimation and Foreground Detection using Kalman-Filtering

[26] Wren, C., Azabayejani, A., Darrel, T. and Pentland, A. 1997. Real-time tracking of the human body, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 780-785

[27] Heikkila J. and Silven O. 1999. A real-time system for monitoring of cyclists andpedestrians, in Second IEEE Workshop on Visual Surveillance, pp. 246-252, (Fort Collins, Colorado)

[28] Sun, J., Zhang, W., Tang, X. and Shum, H. Y. 2006. Background Cut, in Microsoft Research

[29] Kim, K., Chalidabhongseb, T. H., Harwooda D. and Davis L. 2005. Realtime foreground–background segmentation using codebook model Real-time Imaging, Volume 11, Issue 3, Pages 167-256

[30] Peker M., “Görüntü İşleme Tekniği Kullanılarak Gerçek Zamanlı Hareketli Görüntü Tanıma”, Yüksek Lisans Tezi, 93-94, Sakarya 2009.

[31] Pakyürek M., " A Comperative Evalation Of Background/Foreground Segmentasyon Algorithms" , Yüksek Lisans Tezi, 93,94, Ankara 2012.

[32] Gül G. G. , "Anlık Görüntülerden Nesne Çıkarımı Ve Otomatik Nesne Takibi Uygulamaları", Yüksek Lisans Tezi, 100,102, Ankara 2009.

[33] Özkan D. , "Detection of Moving Targets from Video Sequences" , Yüksek Lisans Tezi , 93,94 . Ankara 2011.

[34] Yılmaz M., "Multiple Target Tracking Using Multiple Cameras", Yüksek Lisans tezi , 93,94, .Ankara 2008.

[35] Zaibi R. 1999. Change Detection in Digital Signals, M. Sc Thesis, Institute of Engineering and Sciences, Bilkent University

[36] ViSOR repository. 2007. Web Sitesi:http://www.openvisor.org Erisim Tarihi:10/06/2013

[37] Tu, Q., Xu, Y. and Zhou, M. 2008. Box-based Codebook Model for Real-time Objects Detection, Proceedings of the 7th

[39] Duman, D., Hand Posture Classification And Recognition. http://dogukanduman.com/CV/. 16.07.2013

EKLER

EK – I :Opencv'de Resmi Açmak

```
// PROGRAM 1

#include "stdafx.h"
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include "stdio.h"
int _tmain(int argc, _TCHAR* argv[])
 {
IplImage *img = cvLoadImage("c:/temp/hk.jpg",1 );
cvNamedWindow("Hüseyin KUTLU",1);
cvShowImage("Hüseyin KUTLU",img);
cvWaitKey(0);
cvDestroyWindow("Hüseyin KUTLU");
cvReleaseImage(&img);
cvDestroyWindow ("Hüseyin KUTLU");
return 0;
```

EK – II: Opencv'de Resmi Matris Olarak Almak Ve İşlemek

```
// PROGRAM 2

Mat im = imread("C:\\huseyinkutlu02j.jpg");  //Resim Okundu ve bir matrise atandı
Mat imGray; //Gri resmi atayacağımız değişken tanımlandı
cvtColor(im,imGray,CV_RGB2GRAY);//renk dönüşümleri cvtColor(kaynak,hedef,kod)
fonksyonu ile yapılır
imshow("Orjinal Resim",im);// im resiminigöster
imshow("Gri Resim",imGray);// imGray resimleri göster
Mat cokkontrast= imGray ;// imGray' resmini cokkontrast değişkenine ata
imGray = imGray / 3; // İmGray'in kontrast değerini aritmetik bir işlemle düşürüp imGray'e
ata
imshow("Az Kontrast",imGray);//Kontrast değeri düşmüş imGray'i göster
cokkontrast=cokkontrast * 10;
//imGray'i attığımız cokkontrast değerini aritmetik bir işlemle kontrastını arttır ve
cokkontrast'a at
imshow("Çok Kontrast",cokkontrast );//cokkontrast'resmini göster
//Gelen ekranlarda bekleme yapıyor, bir tuşa basılana kadar.
cvWaitKey(0);
return 0;
```

EK – III: OpenCV'de Bazı Morfolojik İşlemler

```
//PROGRAM 3

Mat im= imread("C:\\hk.jpg",0);
//Resmi al bir matris veri türünde im isimli bir değişkene at 0 direkt gri olarak aldırır
Mat thresh,eroded,dilated,open,close;
// matris veri türünde thresh,eroded,dilated,open,close isimli değişkenler tanımladık
//Threshold ile "im" resimini binary e çeviriyoruz
//cvThreshold( src, dst, thresh_val, 255, CV_THRESH_BINARY);
//cvThreshold(kaynak,hedef,eşik değeri,en üst değer,Code)
threshold(im,thresh,120,255,THRESH_BINARY_INV);
//Çok çeşitli threshold yapma imkanı var OpenCV de
//THRESH_BINARY + _INV
//THRESH_TOZERO + _INV
//THRESH_TRUNC
//Structure element-kernel oluşturma
Mat element = getStructuringElement(MORPH_CROSS,Size(5,5));
//Kernel oluşturulurken;
//MORPH_RECT, MORPH_ELLIPSE gibi diğer yapılarda kullanılabilir
//Erosion yap
erode(thresh,eroded,element);
//Görüldüğü gibi erosion bazı kısımların silinmesine
//Ve alanların daha belirgin olmasına yaradı.
//Dilation yap
dilate(thresh,dilated,element);
//Dilation ise bağlı yerlerin daha bağlı olmasını sağladı.
//Opening
morphologyEx(thresh,open,MORPH_OPEN,element);
//Opening de aynı amaca hizmet eden erosion ile
//benzer bir görüntü oluşturdu
//Closing
morphologyEx(thresh,close,MORPH_CLOSE,element);
//Closing çok da bir şey yapmış gibi gözükmese de
//Bazı yerlerin birleşmesini, kapanmasını sağladı.
//Sonra da hepsini gösterelim.
imshow("im",im);
imshow("thresh",thresh);
imshow("erosion",eroded);
imshow("dilation",dilated);
imshow("open",open);
imshow("close",close);
// bir tuşa basıncaya kadar bekleyelim.
waitKey();
return 0;
```

EK – IV: OpenCV'de Bazı Kenar Bulma Fonksyonlarının Kullanımı

```
int sec;
IplImage *rgb, *gry;
rgb=cvLoadImage("C:/temp/hk02.JPG",1 );
cvShowImage("Kaynak Resim",rgb);
gry = cvCreateImage(cvGetSize(rgb),8,1);
cvCvtColor(rgb,gry,CV_RGB2GRAY);
cvNamedWindow("Gri-Tonlu Resim",1);
cvShowImage("Gri-Tonlu Resim",gry);
cvWaitKey(0);
printf("1-Canny,2-Sobel,3-Laplace");
printf("\nYontem Seciniz…\n");
scanf("%d",&sec);
if (sec==1){ //Canny
IplImage* cny;
cny = cvCreateImage(cvGetSize(rgb),8,1);
cvNamedWindow("Canny Uygula",1);
cvCanny(gry,cny,45,120,3);
cvShowImage("Canny Uygula",cny );
cvWaitKey(0);
cvReleaseImage(&cny);
cvDestroyWindow("Canny Uygula");}
else if (sec==2){ //Sobel
IplImage* sbl;
rgb=cvCloneImage(gry);
sbl = cvCreateImage(cvGetSize(rgb),IPL_DEPTH_16S,1);
cvSobel (rgb, sbl, 1, 0, -1);
cvConvertScaleAbs (sbl, gry,1.0,0.0);
cvNamedWindow("Sobel Uygula",1);
cvShowImage("Sobel Uygula",gry);
cvWaitKey(0);
cvReleaseImage(&sbl);
cvDestroyWindow("Sobel Uygula");}
else if (sec==3){ //Laplace
IplImage* lplc;
lplc = cvCreateImage(cvGetSize(rgb),IPL_DEPTH_16S,1);
cvLaplace (gry, lplc, 3);
cvConvertScaleAbs (lplc, gry,1.0,0.0);
cvNamedWindow("Laplace Uygula",1);
cvShowImage("Laplace Uygula",gry);
cvWaitKey(0);
cvReleaseImage(&lplc);
cvDestroyWindow("Laplace Uygula");}
else printf("Yanlis Girdiniz."); cvWaitKey(0);
cvReleaseImage(&rgb);
cvReleaseImage(&gry);
cvDestroyWindow("Kaynak Resim");
cvDestroyWindow("Gri-Tonlu Resim");
return 0;
```

EK – V: OpenCV'de Bazı Filitreleme Fonksyonlarının Kullanımı

```
// PROGRAM

IplImage* img;
img = cvLoadImage("C:\\hh1.jpg");
cvNamedWindow("Filtre_Oncesi",1);
cvShowImage( "Filtre_Oncesi",img);

IplImage* cvblurnoscale;
IplImage* cvblur;
IplImage* cvgaussian;
IplImage* cvmedian;
IplImage* cvbilateral;

cvblurnoscale=cvCreateImage(cvGetSize(img),8,3);
cvSmooth(img,cvblurnoscale,CV_BLUR_NO_SCALE,5,5);
cvNamedWindow("CV_BLUR_NO_SCALE",1);
cvShowImage( "CV_BLUR_NO_SCALE",cvblurnoscale);

cvblur=cvCreateImage(cvGetSize(img),8,3);
cvSmooth(img,cvblur,CV_BLUR,5,5);
cvNamedWindow("CV_BLUR",1);
cvShowImage( "CV_BLUR",cvblur);

cvgaussian=cvCreateImage(cvGetSize(img),8,3);
cvSmooth(img,cvgaussian,CV_GAUSSIAN,5,5);
cvNamedWindow("CV_GAUSSIAN",1);
cvShowImage( "CV_GAUSSIAN",cvgaussian);

cvmedian=cvCreateImage(cvGetSize(img),8,3);
cvSmooth(img,cvmedian,CV_MEDIAN,5,5);
cvNamedWindow("CV_MEDIAN",1);
cvShowImage( "CV_MEDIAN",cvmedian);

cvbilateral=cvCreateImage(cvGetSize(img),8,3);
cvSmooth(img,cvbilateral,4,5,5);
cvNamedWindow("CV_BILATERAL",1);
cvShowImage( "CV_BILATERAL",cvbilateral);
waitKey();
return 0;
```

EK – VI: OpenCV ile Kamerada Gerçek Zamanlı Görüntü Yakalama

```
//PROGRAM 4

cvNamedWindow( "GORUNTU",1);
CvCapture* video=cvCaptureFromCAM(0);
if (video==NULL)
{
 printf("Dosya okunamadi..\n");
 return 0;
}
IplImage* frame;
while(1)
{
 frame=cvQueryFrame(video);
 if ( !(frame)  ) break;
 cvShowImage( "GORUNTU", frame);
 char c = cvWaitKey(30);
 if ( c == 27 ) break;
}
printf("Okuma islemi bitmistir.\n");
cvReleaseCapture( &video );
cvDestroyWindow( "GORUNTU" );
cvWaitKey(0);
return 0;
}
```

ÖZGEÇMİŞ

1983 yılında Adıyaman'da doğdu. İlk, orta ve lise öğrenimini Adıyaman'da tamamladıktan sonra 2004 yılında Fırat Üniversitesi, Teknik Eğitim Fakültesi, Elektronik Bilgisayar Eğitimi Bölümü, Bilgisayar Öğretmenliğini kazandı. 2008 yılında bu bölümden mezun oldu. Yine 2011 yılında Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik Bilgisayar Eğitimi Bölümü, Bilgisayar Sistemleri Anabilim Dalında yüksek lisans eğitimine hak kazandı. 2008 yılında Milli Eğitim Bakanlığında Bilişim Teknolojileri Teknik Öğretmeni olarak göreve başladı. 2011 yılında Adıyaman Üniversitesi Besni Meslek Yüksekokuluna Öğretim görevlisi olarak arandı. Halen Adıyaman Üniversitesi Besni Meslek Yüksekokulundaki görevini sürdürmektedir. Yabancı dili İngilizce'dir.