YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

VİDEO GÖRÜNTÜLERİNDEN ARAÇ TAKİBİ

Elektrik Müh. Aydoğan AKÇAY

FBE Elektrik Mühendisliği Anabilim Dalı Kontrol ve Otomasyon Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Doç.Dr. Abdullah BAL (YTÜ)

İSTANBUL, 2011

İÇİNDEKİLER

KISALTMA LİSTESİ

RGB	Red,Green,Blue - Kırmızı,Yeşil,Mavi
HSV	Hue,Saturation,Value - Renk tonu,Doygunluk,Değer
GK	Gauss Karışımı (GMM - Gaussian Mixture Model)
GTA	Görsel Takip Ayrıştırma yöntemi
STBA	Seyrek Temel Bileşenler Analizi
BBA	Bağlı Bileşen Analizi
TM	Template Matching - Şablon Eşleştirme
GNN	Global Nearest Neighbor yaklaşımı
JPDA	Joint Probability Data Association
MHT	Multiple Hypothesis Tracking
EMZMC	Etkileşimli Markov Zincirli Monte Carlo
CamShift	Sürekli Uyarlamalı Ortalama Değer Kayması

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Bu tez kapsamında kamera ile daha önceden kaydedilen video görüntülerindeki hareketli araçların takibi gerçekleştirilmiştir. Tezimin araştırma ve uygulama aşamalarında konu hakkındaki bilgisi ile çalışmalarıma destek veren değerli hocam ve tez danışmanım Doç. Dr. Abdullah BAL'a, hayatım boyunca gösterdikleri maddi ve manevi desteklerinden dolayı sevgili anneme, babama ve kardeşime, hayatıma renk katan değerli dostlarıma çok teşekkür ederim.

ÖZET

Nesne takibi; şüpheli şahısların izlenmesi, trafik kontrolü ve düzenlenmesi, askeri güvenlik ve video indeksleme gibi uygulamalarda kullanılan önemli çalışma alanlarından biridir.

Bu tez çalışmasında, hareket halinde araç/araçlar içeren video görüntülerinde seçilen aracın durumu zorlaştıran etkilere rağmen takibi gerçekleştirilmiştir. Kullanılan görüntüler daha önceden kamera ile kaydedilerek elde edilmiştir. Temel işlemler; görüntüdeki hareketli nesnelerin arka plan görüntüsünden ayıklanarak belirgin hale getirilmeleri ve elde edilen yeni görüntü üzerinde kullanıcı tarafından takip edilmesi istenen aracın çerçevelenerek takibin sürdürülmesi olarak sıralanabilir.

Hareketli nesnelerin arka plan görüntüsünden ayıklanması Gauss Karışımı (GK) modeli, seçilen aracın takibi ise Görsel Takip Ayrıştırma (GTA) yöntemi kullanılarak sağlanmıştır.

Hareketli nesnelerin ayıklanması sırasında her bir görüntü sahnesindeki piksellerin arka plana mı ön plana mı ait oldukları piksellerin gauss dağılımlarının belirli kriterleri sağlaması ile belirlenir. Giriş olarak verilen renkli video görüntüsü, piksel değeri 0 (arka plan) ve piksel değeri 1 (ön plan) olan video görüntüsüne dönüştürülerek kaydedilir. Takip edilmesi istenen araç elde edilen yeni görüntü üzerinde çerçevelenerek işaretlenir. İşaretlenen bölgenin renk dağılımı (HSV) ve kenar bilgilerini içeren görüntüler Seyrek Temel Bileşenler Analizi (STBA) medotu ile analiz edilerek temel gözlem modeli oluşturulur. Ardışık görüntü sahneleri arası piksellerin değişimine göre yumuşak veya sert geçiş olarak tanımlanan temel hareket modeli elde edilir.

Temel gözlem modelleri ve temel hareket modelleri ile temel izleyiciler oluşturulur. İzlenen aracın her görüntü sahnesindeki durumu için Etkileşimli Markov Zincirli Monte Carlo (EMZMC) yöntemi ile temel izleyiciler birbirleri arasında haberleşerek üretilen sonuçları karşılaştırırlar ve izlenen araç için en sağlıklı konum bilgisine karar verirler.

Anahtar kelimeler: Nesne takibi, Gauss karışımı, Görsel takip ayrıştırma, Seyrek temel bileşenler analizi, İnteraktif Markov Bağlı Monte Carlo

ABSTRACT

Object tracking is one of the most significiant field of study used in; suspicious people tracking, traffic control, military security and smart video data mining.

In this thesis, selected object has been tracked on video frames which includes moving vehicle/vehicles in challenging situations. The videos are recorded by a video camera before the process. Basic steps can be described as; making moving objects more clear by purifying from background and tracking vehicle which is bounded by user on previously extracted video.

Segmentation of moving objects from background is done by Gaussian Mixture Model (GMM) and selected vehicle tracking is provided by Visual Tracking Decomposition(VTD) method.

While separating moving objects, pixels in each frame are defined by criterias provided Gaussian Mixture Model whether the pixels are belong to background or foreground. Color video given as input, is transformed and saved to a binary movie output whose pixel values are only 0 (background) or 1 (foreground) values. Expected tracking vehicle is pointed on current movie by using bounding box. The color density (HSV) and edge information of pointed area is analysed by Sparse Principle Component Analysis (SPCA) and basic observation model is created. According to pixel value change between sequential frames basic motion model is created which includes smooth and abrupt motions.

Basic observation models and basic motion models are combined as trackers. For the state of vehicle which is being tracked on each frame, basic trackers communicate and exchange information each other by Interactive Markov Chain Monte Carlo (IMCMC) method and compare the results to decide its correct.

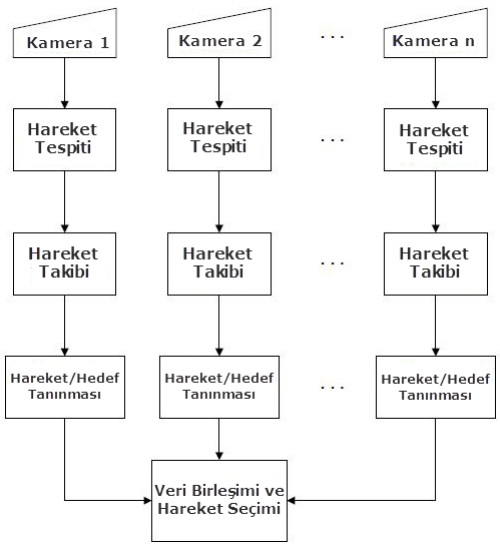
Keywords: Object tracking, Gaussian mixture model, Visual tracking decomposition, Sparse principal component analysis, Interactive markov chain monte carlo

1. GİRİŞ

Günümüz teknolojisinde görüntünün kısa sürede işlenebilir hale gelmesiyle bilgisayarla görme alanında yapılan çalışmalarda insanların günlük ihtiyaçlarını maksimum oranda karşılamak amaçlanmaktadır.

Görüntü bilgilerini değerlendirerek nesne takibi konusunda literatürde çeşitli çalışmalarla uygulamanın amacına yönelik olarak kullanılan yöntemlerde iyileştirmeler yapılmıştır. Genel olarak nesne takibi, ortamda hareket eden hedef nesnenin ardışık görüntüler üzerinde çeşitli analiz metotlarını kullanarak farklılıkları tespit eden algoritmalar yardımı ile görüntüdeki yerinin bulunmasıyla hareket doğrultusunun çıkarılarak takibi olarak tanımlanabilir.

Nesne takibi yapabilmek için öncelikle hedeflenen nesnenin görüntüdeki yerinin bulunması gerekir. Hedeflenen nesne, herhangi bir hareketli nesne olabileceği gibi uygulamanın amacına göre önceden belirlenmiş bir nesne de olabilir. Eğer hedef nesne önceden belirlenmemişse, öncelikle görüntüde bir hareket olana kadar beklenir, hareket gerçekleştikten sonra hareket eden nesne, hedef olarak belirlenir ve takip edilmeye başlanır. Eğer hedef nesne önceden belirlenmişse görüntüde bu nesne aranır ve nesne bulunduktan sonra takip aşamasına geçilir. Takip edilecek nesnenin şekil özelliklerine göre nesneyi temsil edecek bir şekil seçilir. Hedef nesne, kendisini temsil eden şekil içerisindeki özelliklerden yararlanılarak takip edilir. Bu özellikler önemli rol oynadığı için nesneyi çevreden ayırt edecek şekilde farklı olmalıdır.



Şekil 1.1 Görsel izleme sistemi akış diyagramı

Literatürde nesne takibi yapan birçok çalışma yer almaktadır. Şekil 1.1 genel bir nesne takip akış şemasını göstermektedir. Bu çalışmalar; nesneyi temsil eden şekle, kullanılan özelliklere, nesnenin hareketinin, görünümünün ve şeklinin nasıl modellendiğine göre farklılık göstermektedir.

Nesne izleme sistemleri altyapısı genel olarak;

- Hareket eden nesneleri tespit etme
- Nesne Çıkarımı
- İzlenen nesne hareketlerini analiz etme

bileşenlerinden oluşmaktadır.

Gelişen teknoloji ve beraberinde artan güvenlik ihtiyaçları nedeni ile nesne izleme sistemlerinin kullanımı her geçen gün artmaktadır. En yoğun olarak kullanıldığı noktalar;

- Hareket tabanlı tanıma
- Otomatik gözetleme sistemleri (Şüpheli şahıs ve olay izlenmesi)
- Video İndeksleme (Spor karşılaşmalarında istatistik üretilmesi)
- Trafik gözetleme ve kontrol sistemleri (Araç hız ve plaka tespiti, ışık ve şerit ihlalleri)
- Askeri Güvenlik (Askeri üs çevrelerinin gözetlenmesi)

Geliştirilen her bir nesne izleme uygulaması içerisinde bulundurduğu algoritmalar ve kullanılan teknolojilere bağlı olarak karmaşık bir yapıya sahiptir.

Nesne izleme sistemlerinde karşılaşılan en önemli zorluklar;

- Üç boyutlu ortamdaki nesnelerin iki boyuta dönüştürülmesi esnasında oluşan veri kayıpları
- Görüntüler üzerindeki gürültü
- Karmaşık nesne hareketleri ve şekilleri
- Sahne ve ışık değişimleri

olarak sıralanabilir.

İzlenilmesi istenen nesnelerin boyut, biçim, renk gibi görsel özelliklerinin çıkarılabilmesi ve bu verilerin işlenebilmesi için bugüne kadar birçok araştırma yapılmıştır ve nesnelerin bu sistemin genel performansını doğrudan etkilediği ortaya koyulmuştur. İzlenecek olan nesnenin fiziksel değişime uğraması veya bir engelle karşılaşıp kısmen ya da tamamen kamera görüş alanından çıkması durumu da dâhil olmak üzere izleme işlemini zorlaştıran koşullarla karşılaşılması durumlarında nesnenin önceki hareketlerin analiz edilmesi ve bu hareketlere göre tahmini olarak nesnenin bulunacağı yerin hesaplanması amaçlanmıştır.

Tezin içeriği şu şekilde düzenlenmiştir. 2. bölümde hareketli nesnelerin tespit edilmesi ile ilgili genel bilgiler ve nesne tespit yöntemleri, 3.bölümde nesne takibinde rol oynayan kriterler ve nesne izleme metotları, 4. bölümde araç takibi üzerine yapılan çalışma ve amacı, kullanılan yöntemler ve aşamaları, 5. Bölümde farklı durumlar karşısındaki performans analizi ve çalışmanın son kısmı olan 6. Bölümde ise sonuçlar incelenerek geleceğe yönelik çalışmalar için izlenecek yöntemler hakkında öneriler verilmeye çalışılmıştır.

2. HAREKETLİ NESNE TESPİTİ

Nesne tespit yöntemleri hareket eden nesnelerin çeşitli yöntemler kullanılarak video sahneleri içerisinde belirlenmesi ve hareket doğrultusunun çıkarılması amacı ile kullanılmaktadır. Görüntü izleme sistemlerinin ilk kısmı olarak kabul edilir ve sistem performansına etkisi büyüktür. Çizelge 2.1'de kullanılan nesne tespit yöntemleri kategoriler halinde belirtilmiştir.

Çizelge 2.1 Nesne tespit yöntemleri

Kategoriler	Yapılan çalışma
Nokta dedektörleri	Moravec's detector [Moravec 1979], Harris detector [Harris and Stephens 1988], Scale Invariant Feature Transform [Lowe 2004]. Affine Invariant Point Detector [Mikolajczyk and Schmid 2002].
Bölütleme	Mean-shift [Comaniciu and Meer 1999], Graph-cut [Shi and Malik 2000], Active contours [Caselles et al. 1995].
Arka plan modelleme	Mixture of Gaussians[Stauffer and Grimson 2000], Eigenbackground[Oliver et al. 2000], Wall flower [Toyama et al. 1999], Dynamic texture background [Monnet et al. 2003].
Denetlemeli Sınıflandırıcılar	Support Vector Machines [Papageorgiou et al. 1998], Neural Networks [Rowley et al. 1998], Adaptive Boosting [Viola et al. 2003].

Sonraki kısımda sıkça kullanılan hareketli nesne tespit yöntemleri olan arka plan çıkarımı, istatistiksel metodlar, sahne farkı alınması ve optik akış yöntemlerinden bahsedilmiştir.

2.1 Arka Plan Çıkarımı Yöntemi

Arka plan farkı alınarak hareketli nesnenin bulunması, görüntü alımı yapan kameranın sabit, hedef nesnenin hareketli olduğu durumlarda kullanılan bir yöntemdir. Arka plan görüntüsü sistem çalışmaya başlamadan önce veya sistem çalışmaya başladıktan sonra hareketsiz kalan görüntü ile belirlenir. Daha sonra gelen her bir görüntü ile arka plan görüntüsünün farkı alınır. Elde edilen farkın mutlak değeri daha önceden belirlenmiş olan bir eşik seviyesi değeri (T) ile kıyaslanır (Eşitlik 2.1).

$$\left| görüntü_{i}(x,y) - arkaplan(x,y) \right| > T \tag{2.1}$$

Eğer fark değeri bu eşik seviyesi değerinin üzerindeyse ilgili koordinatlarda bir değişiklik olduğuna karar verilir. Fakat yalnızca bu kriterle o bölgede hareket olduğu kanısına varılamaz. Çünkü ışık farklılığı sebebiyle bazı pikseller bu kriteri sağlayabilir.Bu piksellerin hareketli bir nesneyi tanımlaması için değişim olan piksellerin kapalı bir alan oluşturması ve bu alanın belirlenen bir eşik seviyesi değerinden büyük olması gerekir.

Birbirine bağlı bölgeleri ayrı ayrı belirlemek amacıyla Bağlı Bileşen Analizi (Connected Component Analysis) yöntemi kullanılabilir. Bu yöntem uygulandıktan sonra ışık farklılığı sebebiyle değişiklik gösteren pikseller elenmiş, hareket eden nesneler yani arka planda yer almayan nesneler bulunmuş olur (Şekil 2.1).

Şekil 2.1 Arka plan çıkarımı sonucu

Eşik değerin üstünde değişim gösteren pikseller hareketli cisim olarak tanımlanır. Bu cisimlerdeki gürültülerin etkisini düşürmek ve belirginliğini iyileştirmek için aşınma, genişleme, açma ve kapama gibi morfolojik işlemler uygulanır.

Referans alınan arka planın yeni görüntüler geldikçe değişen arka plana adapte olabilmesi için zamanla yenilenmelidir. Yüksek verimlilikte çalışacak olan arka plan çıkarım algoritması;

- Işık Değişimleri
- Ufak Kamera Hareketleri
- Ortamda bulunan diğer hareketler (Ağaç yaprakları, dalgalar vb.)

gibi çevresel faktörlerden etkilenmeyecek şekilde tasarlanmalıdır.

(Stauffer ve Grimson, 1999) gerçek zamanlı adaptif arka plan karışım modeli tanımlamıştır. Çalışmalarında, güncel görüntü verileri ile yenilenen Gauss karışımı yöntemini kullanarak her pikseli ayrı ayrı modellemişlerdir. Pikselin arka plana mı yoksa hareketli cisme mi ait olduğunu belirlemek için karışım modelindeki Gauss dağılımları değerlendirilmiştir.

(Haritaoglu vd., 2000) her pikseli en yüksek ve en düşük parlaklık değerleri ve ardışık sahneler arasındaki en yüksek parlaklık farklarını kullanarak temsil ettiği arka plan modeli kullanmıştır. Bu yöntemin dezavantajı aydınlatma değişikliklerine karşı hassas olmasıdır.

2.2 Sıralı Görüntüler Farkı Yöntemi

Sıralı görüntülerin farkının alınmasıyla nesne bulma, video akısında arka arkaya gelen görüntüdeki piksellerin renk değerlerinin farkını alarak bu farkı bir eşik seviyesi ile karşılaştıran bir yöntemdir (Eşitlik 2.2).

$$\left|görüntü_{i}(x,y) - görüntü_{i-1}(x,y)\right| > T \qquad (2.2)$$

Bu fark sayesinde görüntüdeki yer değişimleri ortaya çıkmaktadır. Arka plan çıkarımı yönteminde referans görüntüde yer almayan bütün nesneler bulunurken bu yöntemin farkı yalnızca hareketli bölgelerin, yani bir önceki görüntüye göre farklılık gösteren bölgelerin tespit ediliyor olmasıdır.

2.3 Bölümlere Ayırma (Bölütleme) Yöntemi

Bölümlere ayırma yöntemini kullanan algoritmalarda amaç, görüntüyü birbiri ile anlamsal ilişkisi olduğu düşünülen benzer bölgelere ayırarak hareketli nesnelerin tespitini sağlamaktır. Görüntü üzerinde anlamsal ilişki kurabilmek için görüntü üzerinde bulunan renk ve piksel değerlerinden faydalanılmaktadır. Şekil 2.2'de bölütleme yöntemine örnek gösterilmiştir.

(a) (b)

Şekil 2.2 Bölütleme metodu gösterimi a) Orjinal resim b) Bölütleme sonucu

Bölütleme ile nesne bulmada kullanılabilecek yöntemlerden biri gözetimli öğrenme yöntemi ile hedef nesnenin bulunmasıdır. Bu yöntemde, önceden özellikleri çıkarılıp sisteme öğretilmiş bir hedef nesne bulunmaktadır. Bölütleme işlemi sonucu elde edilen her bir parçanın özellikleri çıkarılır ve aday nesnenin hedef nesne olup olmadığı test edilir. Hedef nesne bulunduğunda takip aşamasına geçilir. Hedef nesneyi temsil eden özelliklerin seçimi sistemin başarısında önemli bir rol oynar.

Bölütleme ile nesne bulmada kullanılabilecek başka bir yöntem ise Şablon Eşleştirme (Template Matching) ile hedef nesnenin bulunmasıdır. Takip edilmesi amaçlanan hedef nesne ile bölütleme sonucu elde edilen her bir parça şablon eşleştirme yöntemi ile birbiriyle kıyaslanır. Hedef nesneye en çok benzeyen parça eğer benzeme oranı eşik seviyesi değerini de geçmişse hedef nesne olarak tanınır ve takibe başlanır. Bu yöntem, hedef nesnenin farklı açılardan çekilmiş görüntüleri ile sistem eğitilerek de uygulanabilir.

2.4 Ortalama Kaydırma Öbeklemesi (Mean-shift) Yöntemi

(Comanicu ve Meer, 2002) tarafından geliştirilen ortalama kaydırma öbeklemesi adlı algoritmada; öbeklerin merkezlerinin belirlenen ortalamalarda kaydırılması ile birbiri ile ilişkili bölümlerin tespit edilmesi amaçlanmıştır.

Genel yapıya bakılacak olursa, öbeklerin merkezi verilen görüntü üzerinden rastgele seçilmekte ve sonrasında bu seçilen öbeklerin renk değerlerinin ortalaması alınarak öbek merkezinin kendi değerlerine en yakın piksel değerlerine ulaşana kadar kaydırılması ile görüntüyü bölümlere ayırmak amaçlanmıştır.

Ortalama kaydırma yönteminde, hedefin bir sonraki durumu mevcut sahnedeki ve bir sonraki sahnedeki nesne histogramlarının karşılaştırılması ile bulunur. Ortalama kaydırma iterasyonları, mesafe tabanlı Bhattacharrya katsayısının belirttiği hedef nesne adayı ve hedef nesne arasındaki benzerliği bulmaktadır. Ayrık yoğunluk, q hedef model, p hedef adayı. Bhattacharyya katsayısı;

$$\rho(y) \equiv \rho\left[\rho(y), q\right] = \sum_{u=1}^{m} \sqrt{\rho_u(y) q_u} \tag{2.3}$$

Ve bu katsayıyı maksimum yapan mesafe ölçümü

$$d(y) = \sqrt{1 - \rho\left[\rho(y), q\right]} \tag{2.4}$$

olarak bulunur. Maksimum Bhattacharya katsayısını bulmak için, yükselen vektör yoğunluk fonksiyonu ile hesaplanan ortalama kayma vektörü izlenmelidir. Ortalama kayma vektörü, Bhattacharya katsayısı maksimum yapan y bölgesine toplandığında, y bir sonraki sahnede hedef nesnenin yeni konumu haline gelir.

2.5 Sürekli Uyarlamalı Ortalama Değer Kayması (SUODK) Yöntemi

Sürekli Uyarlamalı Ortalama Değer Kayması (SUODK) (CamShift-Continuously Adaptive Mean Shift) (Bradski, 1998) yöntemiyle takip edilecek nesnenin renk dağılımından elde edilen histogram arama kriteri olarak kullanılmaktadır.

Arama alanı olarak verilen görüntü seçilmekte ve ardışık görüntülerde hedef nesnenin renk dağılımını yoğun olarak taşıyan bölge iteratif olarak gerçekleştirilen işlemlerle bulunmaktadır.

SUODK algoritması, yalnızca statik dağılımlara çözüm getirebilen Ortalama - Ötelemesi (Mean-Shift) algoritması temel alınarak üretilmiş efektif bir takip algoritmasıdır. SUODK yönteminde, Mean-Shift algoritmasından farklı olarak iterasyonlar boyunca dağılımın

merkezine doğru moment hesabı yapılır. Nesnenin olasılık dağılımının yeri ve büyüklüğü; nesnenin hareketine, ışık değişimine, görüş açısının değişimine, gölgeye göre değişiklik gösterir.

Hedef nesneyi bulma amacıyla kullanılan özellik, nesnenin renk olasılık dağılımıdır. Ardışık görüntülerin her birinin renk olasılık dağılımı hedef nesnenin renk olasılık dağılımından yararlanılarak çıkartılmakta ve hedef nesnenin yeri görüntü üzerinde bulunmaktadır. Bu amaç doğrultusunda çalışılan görüntü üzerindeki bütün piksellerin momentleri renk olasılık ve yer bilgisi kullanılarak hesaplanmaktadır. Hesaplanan bu momentler ışığında, hedef nesnenin görüntü üzerinde bulunduğu aday bölge belirlenir. Bu aday bölge ile hedef nesnenin renk olasılık dağılım fonksiyonları, Bhattacharyya uzaklığı ile karşılaştırılır.

Elde edilen benzerlik sonucu, önceden belirlenmiş sabit eşik seviyesinin üzerindeyse hedef nesnenin bulunmuş olduğuna karar verilir. Nesnenin elde edilen yer bilgisi nesne takibi aşamasına aktarılır ve nesne takibine başlanır. Eğer elde edilen benzerlik sonucu, eşik seviyesi değerinin altındaysa bu sonuç eşik seviyesi değerinin üzerine çıkana kadar nesne bulma aşamasına devam edilir.

Nesne takibi aşaması, nesne bulma aşamasından gelen hedef nesnenin bir önceki görüntüdeki yer bilgisini ve hedef nesnenin renk olasılık dağılım bilgisini kullanarak nesne takibine başlamaktadır. Nesne takibi aşamasında gerçek zamanlı olarak alınan ardışık görüntüler üzerinde anlık sonuçlar üretilmektedir. Hedef nesne, gerçek zamanlı olarak alınan her bir ardışık görüntü üzerinde SUODK yöntemi ile aranmaktadır. O anki görüntü üzerinde hedef nesnenin aranacağı alanın merkezi bir önceki görüntü üzerinde nesnenin bulunmuş olduğu yerin merkezi olacak şekilde belirlenir. Arama alanının büyüklüğü ise piksellerin renk olasılığı ve piksellerin yer bilgisi kullanılarak hesaplanan momentler üzerinden bulunmaktadır. Hedef nesne, büyüklüğü ve yeri belirlenen ilgili alan içinde aranmaktadır. Arama işlemi yine SUODK yönteminde yer alan moment hesabı kullanılarak yapılmaktadır.

Bu momentlerden yararlanılarak arama alanında hedef nesnenin renk olasılık dağılımının en yoğun olduğu bölge bulunmaktadır ve hedef nesnenin yeri bu bölgenin merkezi olarak belirlenmektedir. Yine momentlerden yararlanılarak hedef nesnenin görüntü üzerinde kapladığı alan bulunmaktadır. Hedef nesnenin görüntü üzerindeki yer bilgisi, nesnenin görüntüde kapladığı alan ve arama alanının boyutu; bir sonraki nesne takibi aşamasına aktarılır. Görüntü alımı devam ettiği sürece nesne takibi adımları da bu şekilde devam eder.

3. HAREKETLİ NESNE TAKİBİ

Önemli ve zor bir problem olan nesne takibi bilgisayarla görme alanında çalışan araştırmacılar için önemli bir çalışma konusudur. Takibin amacı ardışık video sahneleri arasındaki nesne ve nesne kısımları iletişimini kurmaktır.

Hedef takibinde iki yaygın yaklaşım bulunmaktadır. Bunlardan biri iletişim eşleştirme, diğeri ise konum veya hareket tahmini temeline dayanır. Çizelge 3.1'de kullanılan nesne tespit yöntemleri kategoriler halinde belirtilmiştir.

Çizelge 3.1 Nesne takip yöntemleri

Kategoriler	**Yapılan Çalışma**
Nokta Takibi	
• Belirleyici (Determinist) Metotlar	MGE izleyicisi [Salari ve Sethi 1990], GOA izleyicisi [Veenman ve diğerleri 2001].
• İstatistiksel Metotlar	Kalman filtresi [Broida ve Chellappa 1986], JPDAF (Joint Probabilistic Data Association Filter) [Bar-Shalom ve Foreman 1988], PMHT (Probabilistic Multiple Hypothesis Tracking) [Streit ve Luginbuhl 1994].
Çekirdek Takibi	
• Şablon ve yoğunluk tabanlı görünüm modelleri	Mean-shift [Comaniciu ve diğerleri 2003], KLT (Kanade Lucas Tomasi Feature Tracker) [Shi ve Tomasi 1994], Layering [Tao ve diğerleri 2002].
• Çoklu görüşlü görünüm modelleri	Eigentracking [Black ve Jepson 1998], SVM (Support Vector Machine) izleyicisi [Avidan 2001].
Siluet Takibi	
• Çevre çizgisi yayılımı	Durum uzay modelleri [Isard ve Blake 1998], Varyasyon metotları[Bertalmio ve diğ. 2000], Sezgisel metotlar [Ronfard 1994].
• Şekil eşleştirme	Hausdorff [Huttenlocher ve diğerleri 1993], Hough dönüşümü [Sato ve Aggarwal 2004], Histogram [Kang ve diğerleri 2004].

Genel olarak hatalı bölütleme problemlerinin sebepleri; ışık değişimleri, çevresel etkiler(ağaç dallarının ve yapraklarının sallanması), gölgeler, nesnenin bir kısmının veya tamamının diğer nesneler veya sabit cisimler tarafından engellenmesi durumlarıdır. Güçlü bir takip sistemi için bu problemler dikkatlice göz önüne alınarak çözümlenmelidir.

Model tabanlı beden kısımları takip sistemine örnek olarak, Pfinder (Wren vd., 1997) gerçek zamanlı çalışan insanların el ve kafa kısımlarının renk ve şekil özelliklerini çok sınıflı istatistiksel model kullanarak uygulamıştır.

(Bradski, 1998) literatürde SUODK olarak da adlandırılan olasılık dağılımlarını işleyen ortalama kayma algoritmasını yüz takip sistemi için tasarlamıştır. Ardışık video görüntülerindeki renkli nesneleri takip etmek için, renkli görüntü verileri histogramlar kullanılarak olasılık dağılımları ile ifade edilmiştir.

(Haritaoglu vd., 2000) hareket tahmin ve iletişim eşleme yöntemlerini birlikte kullanmıştır. Ayrıca, beden kısımlarının takibi için kafa, el, ayak, gövde bölgelerinin göreceli konum ve boyutlarını karton modelini (Ju vd., 1996) kullanarak belirtmiştir. Birleşme ve ayrılma durumlarındaki eşlemeyi düzgün yapabilmek için ayrık nesnelerin görünüş şablonunu dikkate almıştır.

(Stauffer ve Grimson, 2000) lineer olarak tahmin edilebilen hipotez takip algoritmasını kullanmıştır. Algoritma hareket tahmini için nesnenin boyut ve pozisyonunu Kalman filtreleri sayesinde birleştirmiştir. Hareket yönü tahmini için (Rosales ve Sclaroff, 1998) çalışmalarında genişletilmiş Kalman filtrelerini kullanmıştır.

(Comaniciu vd., 2000,2003) esnek nesnelerin istatistiksel dağılımlarını nitelendiren renk ve doku gibi görsel özelliklerini kullanmıştır. Ortalama kayma iterasyonları, metrik tabanlı Bhattacharya katsayısını kullanarak verilen hedef modele benzer olan hedef adaylarını bulmaktadır. Algoritma, takip edilecek nesnenin ardışık her bir görüntüde Ortalama Değer Kayması (Mean-shift) iterasyonları ile yerinin belirlenmesine dayanır.

(Amer, 2003) takip için lineer olmayan oylama tabanlı stratejiyi tanıtmıştır. Boyut, şekil, kütle merkezi ve hareket gibi nesne özelliklerini oylama ile bütünleştirmiş ve nesne iletişimi ile son eşleme durumuna karar vermiştir. Bu yöntem nesnenin yapısal olarak ayrılmasını ve birleşmesini de tespit ederek bu engelleri ortadan kaldırmıştır.

Bazı çalışmalarda nesne takibinde karşılaşılan problemler üzerinde durulmuş ve bu problemlere çözümler üretilmeye çalışılmıştır. Bu çalışmalardan birinde, görüntü özelliklerinin değişimine ve hedef nesnenin tamamının görüntülenememesine rağmen, hedef nesne, enerji fonksiyonu kullanılarak takip edilmiştir (Yilmaz vd., 2004).

(Bal ve Alam, 2005) kızılötesi görüntü dizilerinde hedefin yoğunluk özelliği, çevreleyen arka plan ve şekil bilgilerini kullanarak nesne takibi işlemini gerçekleştirmişlerdir. (Dawoud vd., 2006) havadan çekilen görüntülerin takibindeki başarısızlık durumlarına ağırlıklı çok parçalı referans fonksiyonu kullanarak çözüm üretmişlerdir.

Bir başka uygulama da, otonom mobil robot üzerine yerleştirilmiş kameradan görüntü alarak, nesne bulma ve takibi gerçekleştiren sistemde hedef nesnenin ve kameranın hareketli olması, takibini zorlaştırmıştır (Kim vd., 2005). Problemi çözmek için kamera hareketinin doğrultusu ve hızı kullanılarak takip edilecek nesnenin hareket doğrultusu ve hızı hesaplanmıştır.

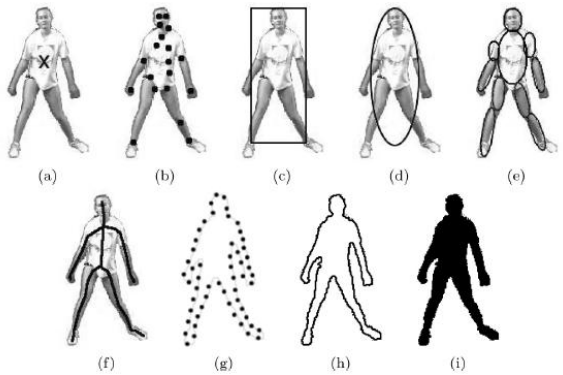
Literatürde, gerçek zamanlı video görüntüleri üzerinde değil, önceden kaydedilmiş görüntüler üzerinde nesne takibi yapan çalışmalar da mevcuttur. Bu çalışmalardan birinde, takip edilerek bulunan hareketli nesnelerin silinmesi amaçlanmıştır (Park vd., 2005). Bu çalışmada kullanılan kameranın hareketli olmasından dolayı arka arkaya gelen görüntüler birleştirilerek bir mozaik oluşturulmuş, böylece kamera hareketi elimine edilmiştir.

3.1 Nesne Takibinde Rol Oynayan Faktörler

Nesne takibi işlemine başlamadan önce hedef nesneye, ortama ve uygulamaya göre belirlenmesi gereken bazı kriterler vardır. Bunlar temel olarak nesne gösterimi, takip amacıyla kullanılan özellik ve kullanılacak nesne takibi metodudur.

3.1.1 Nesne Gösterimi

Nesne izleme sistemlerinde izlenecek olan nesnenin gösterimi görsellik açısından olduğu gibi uygulama kolaylığı açısından da farklılık gösterebilmektedir. (Yılmaz vd., 2006)'a göre izleme sistemlerinde kullanılan nesne gösterim metotları Şekil 3.1'de gösterilmiştir.



Şekil 3.1 Nesne gösterim şekilleri a) merkez nokta b) çoklu nokta c) dikdörtgen d) elips e) parça-tabanlı çoklu elips f) iskelet g) genel dış çizgiler h) detaylı dış çizgiler i) silüet

Noktalar ile Gösterim:

Nesne, merkezinde yer alan bir nokta ile veya noktalar kümesi ile gösterilir. Nokta gösterimi görüntüde az yer kaplayan nesneleri takip etmek için uygundur.

Geometrik Şekil ile Gösterim:

Nesne dikdörtgen, elips gibi bir geometrik sekil ile gösterilir. Bu şekillerle gösterilen nesnenin hareketi genellikle ötelemeli, doğrusal veya izdüşümsel olmaktadır. Geometrik sekil ile gösterim genellikle katı, esnek olmayan nesneler için uygun olmakla birlikte, esnek nesneleri takip etmek için de kullanılır.

Nesne Silüeti ve Çevre Çizgisi ile Gösterim:

Çevre çizgisi nesnenin sınır çizgilerini temsil eder. Çevre çizgisinin içinde kalan bölge ise nesnenin silüeti olarak tanımlanır. Çevre çizgisi ve silüet gösterimlerinin karmaşık ve esnek yapıda nesnelerin takibinde kullanımı uygundur.

Eklemli Şekil Modelleri ile Gösterim:

Eklemli nesneler birbirine eklem ile bağlanan parçalardan oluşur. Örneğin insan; gövdesi, kolları, bacakları, kafası, elleri ve ayaklarının eklemlerle birbirine bağlanmasından oluşan bir canlıdır. Bu gösterimde parçalar arası ilişkiler kinematik hareket denklemleri ile çıkartılabilir. Eklemli bir nesnenin gösteriminde, nesnenin eklemler ile bağlanan her bir parçası silindir veya elips ile ifade edilebilir.

İskelet Model ile Gösterim:

Nesnenin iskeleti, nesnenin silüetine orta eksen dönüşümü uygulanarak çıkarılabilir. Bu model genellikle nesne tanımada sekil gösterimi olarak kullanılır. İskelet gösterimi, hem eklemli hem de esnek yapılı nesnelerin gösterimi için kullanılabilir.

3.1.2 Özellik Seçimi

Nesne takibinde özellik seçimi kritik bir rol oynar. Seçilecek özelliğin veya özelliklerin takip edilecek nesneyi ayırt edici kılması gerekir. Görsel özelliklerin kullanılmasındaki en önemli sebep de her bir nesnenin kendine özel görsel özellikleri olmasıdır. Özellik seçimi, nesne gösterimi ile yakından alakalıdır. Örneğin histogram tabanlı gösterimlerde renk özelliği kullanılırken, çevre çizgisi tabanlı gösterimlerde kenar özelliği kullanılmaktadır.

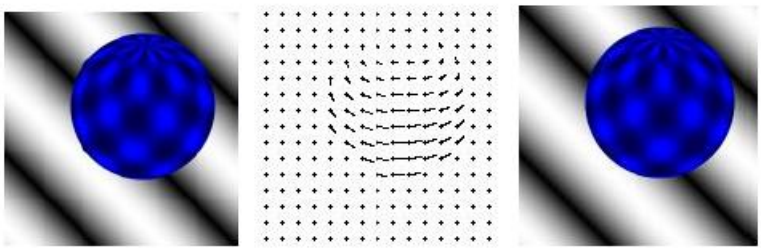
- ***Renk;*** nesnenin görünen rengi iki fiziksel etken tarafından belirlenir (aydınlatmanın spektral güç dağılımı, nesnenin yansıma yüzeyi özellikleri). Görüntü isleme çalışmalarında, rengi temsil etme amacıyla genellikle RGB (Red, Green, Blue – Kırmızı, Yeşil, Mavi) renk uzayı kullanılır. Bunun yanında HSV (Hue, Saturation, Value – Renk tonu, Doygunluk, Değer) uzayları da kullanılmaktadır.

- ***Kenarlar;*** nesne sınırları genellikle görüntü canlılığında güçlü değişimler oluşturur. En popüler kenar bulma yaklaşımları Sobel, Roberts ve Canny kenar bulma yöntemleridir. Şekil 3.2'de Canny kenar bulma yönteminin uygulama sonucu gösterilmiştir. Nesne dış çizgileri görüntü yoğunluğunda genellikle büyük değişimlere yol açar. Kenarların en önemli özelliği renk özellikleri ile kıyaslandığında ışık değişimlerine daha az duyarlı olmasıdır.

(a) (b)

Şekil 3.2 Canny kenar bulma metodu sonucu a) orjinal resim b) kenar gösterimi

- ***Optik akış;*** her bir piksel değişiminin hareket vektörleriyle gösteriminin bütünüdür (Şekil 3.3). Optik akış, bir bölgedeki her bir pikselin ötelemesini tanımlayan yer değiştirme vektörlerinin yoğunluk alanıdır. Arka arkaya gelen görüntüdeki eş piksellerin parlaklıkları kullanılarak hesaplanır. Optik akış, hareket tabanlı bölütleme ve takip uygulamalarında sıklıkla kullanılan bir özelliktir.



(a) (b) (c)

Şekil 3.3 Optik akış gösterimi a) ilk durum b) optik akış vektörleri c) sonraki durum

(Okada vd., 1996) tarafından geliştirilen optik akışa dayalı nesne izleme sistemi cismin optik akışını, (Srinivasan, 1990) tarafından geliştirilen genellenmiş eğim metodunu kullanarak belirler. Çalışmada;

$f(x,y,t)$, t anında görüntü üzerinde bulunan x ve y noktalarındaki ışık oranı olarak belirlenmiştir. Eğer nesne dt sürede dx ve dy kadar yer değiştirirse formül;

$$f(x,y,t) = f(x+dx, y+dy, t+dt) \tag{3.1}$$

şeklinde yazılabilir.

Taylor serileri kullanılarak elde edilen yeni formül :

$$f(x,y,t) = f(x,y,t) + f_x(x,y,t)dx + f_y(x,y,t)dy + f_t(x,y,t)dt + e \tag{3.2}$$

şeklinde olacaktır.

Bu formül:

$$f_x = \frac{\partial f}{\partial x}, f_y = \frac{\partial f}{\partial y}, f_t = \frac{\partial f}{\partial t} \tag{3.3}$$

ile gösterilecek olursa ve e ihmal edilirse yeni formül :

$$f_x(x,y,t)u + f_y(x,y,t)v + f_t(x,y,t) = 0 \tag{3.4}$$

olacaktır. Akış vektörleri olan u ve v

$$u = \frac{d_x}{d_t}, v = \frac{d_y}{d_t} \tag{3.5}$$

şeklinde yazılır. g ve h filtreleri bu son form üzerinde uygulanacak olursa ;

$$\begin{pmatrix} (g*f)x & (g*f)y \\ (h*f)x & (h*f)y \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} (g*f)t \\ (h*f)t \end{pmatrix} \tag{3.6}$$

formülü elde edilir. Akış vektörleri *u,v* yukarıdaki denklemin çözümünden elde edilmekte ve çalışmada nesne takibi elde edilen akış vektörlerine göre gerçekleştirilmektedir.

Optik akış hareketli cisimleri belirleyen etkili bir yöntemdir, fakat hesaplama yöntemleri karmaşık ve gürültülere karşı hassastır. Bu yüzden gerçek zamanlı uygulamalarda özel donanımlara ihtiyaç duyarlar.

- ***Doku;*** pürüzsüzlük ve düzenlilik gibi özellikleri belirten yüzey yoğunluk değişiminin ölçüsüdür (Şekil 3.4). Renk ile karşılaştırıldığında doku tanımlayıcılarını oluşturabilmek için filtre uygulama gibi ön işlemlere ihtiyaç duyar. Kenar özellikleri gibi doku özellikleri de ışık değişimlerine daha az duyarlıdır.

(a) (b)

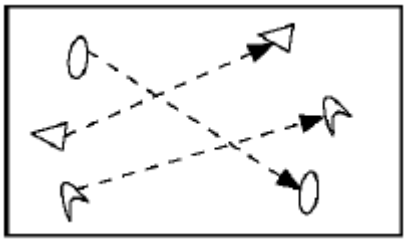
Şekil 3.4 Doku özelliğinin gösterimi a) orjinal resim b) doku gösterimi

3.2 Nesne Takip Metotları

Çevresel koşulların uygun olması durumunda bu yaklaşımların birden fazlasını bir arada kullanarak hibrit izleme algoritmaları oluşturabilmek mümkündür (Cavallaro vd., 2005).

3.2.1 Nokta Takibi

Arka arkaya gelen görüntülerde, bulunan nesneler noktalar ile temsil edilir (Şekil 3.5). Bu noktalar topluluğu, nesne konumunu ve hareketini içeren önceki nesne durumuna dayanır. Genellikle merkezi noktalar veya köşeler izlenen noktalar olarak kullanılır. Bu yaklaşım her bir görüntüde hedef nesneyi bulacak harici mekanizmaya ihtiyaç duyar. Nokta iletişimi; özellikle kaybolma, yanlış bulma, nesnelerin görüntüye giriş ve çıkışlarının var olduğu durumlarda karmaşık bir problemdir.

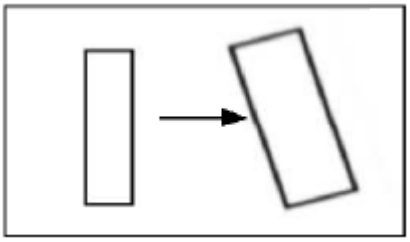


Şekil 3.5 Nokta takibi

Nokta iletişimli yöntemler deterministik(aynı girişe aynı sonucu veren) ve istatistiksel olarak iki kategoriye ayrılır. Deterministik yöntemler iletişim sorununu engellemek için niteliksel hareketlerin deneme-yanılma yöntemini kullanır. Diğer taraftan, olasılıklı yöntemler iletişimi sağlamak için nesne ölçümünü ve belirsizlikleri dikkate almaktadır.

3.2.2 Çekirdek Takibi

Çekirdek, objenin görünümünü ve şeklini temsil eder. Örneğin, çekirdek o bölgenin histogramı ile ilişkilendirilmiş dikdörtgen bir şablon veya eliptik bir şekil olabilir. Bu yöntemde nesne takibi, arka arkaya gelen görüntülerdeki çekirdeğin hareketinin hesaplanmasıyla gerçekleştirilir. Bu hareket genellikle öteleme ve dönme gibi parametrik dönüşümler şeklinde olmaktadır (Şekil 3.6).

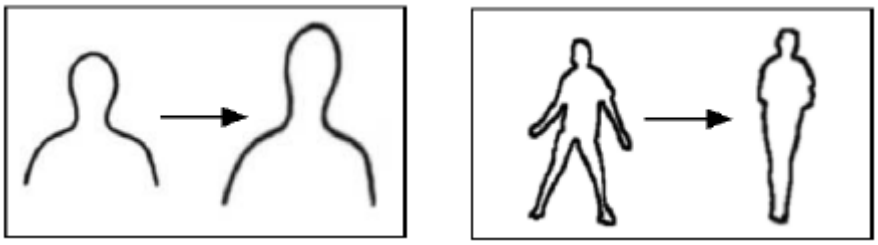


Şekil 3.6 Dikdörtgen şeklin parametrik dönüşümü

Bu algoritmalar görünüşün temsil edilmesi, takip edilen nesne sayısı ve nesne hareketini tahmin etmekte kullanılan yöntemler bakımından farklılıklar gösterirler.

3.2.3 Silüet Takibi

Nesneler çok karmaşık şekillere sahip olabilirler. Örneğin insan bedeni eller, kafa, omuzlar gibi eklemlerle bağlanmış parçalardan oluşmaktadır. Bu yapıda bir nesnenin basit geometrik şekillerle tanımlanması zordur. Silüet tabanlı yöntemler bu tarz nesneler için kesin şekil tanımlamaları sunar (Şekil 3.7).



Şekil 3.7 Çevre çizgisi oluşturma örnekleri

Silüet tabanlı obje takibinde amaç, önceki görüntülerde tanımlanmış nesne modelinin kullanılmasıyla arka arkaya gelen her bir görüntüdeki nesne bölgesinin bulunmasıdır. Silüet takip yöntemleri nesne bölgesi içindeki kod bilgisini kullanır. Kullanılan bu nesne modeli genellikle nesne dış çizgisi içerisinde bulunan görünüş yoğunluğu renk histogramı, nesne kenarları veya nesne dış çizgisi şeklinde olabilir. Silüet takibi, şekil eşleştirme ve çevre çizgisi takibi olmak üzere iki kategoriye ayrılabilir. Şekil eşleştirme yaklaşımları, o anki görüntüde nesnenin silüetini aramaktadır. Çevre çizgisi yaklaşımları ise durum uzay modellerini veya enerji fonksiyonlarının doğrudan minimizasyonunu kullanarak o anki görüntüdeki nesnenin yeni pozisyonuyla bir çevre çizgisini ilişkilendirir.

3.2.4 Kalman Filtresi ile Nesne Takibi

Nesne takibi amacıyla kullanılan yöntemlerden en bilineni Kalman filtresidir. Bu yöntemin uygulamasının basit olmasından ve gerçek zamanlı çalışabilmesinden dolayı literatürde birçok çalışmada kullanılmıştır. Kalman filtresi, nesne takibinde hedef nesnenin doğrusal hareket ettiğini ve sistemde Gauss paraziti bulunduğunu kabul eder. Nesne takibi amacıyla kameralardan elde edilen ölçümler çoğu zaman parazit içermektedir.

Bunun yanında nesne hareketleri de rastgele düzensizlikler içerebilir. Kalman filtresinin de dahil olduğu istatistiksel eşleme metotları, nesne durum tahmini boyunca ölçüm ve model belirsizliklerini de hesaba katarak bu gibi nesne takibi problemlerine çözüm üretir (Şekil 3.8).

Kalman Filtresi de diğer istatistiksel esleme metotları gibi pozisyon, hız ve ivme gibi nesne özelliklerini belirleyebilmek için durum alan yaklaşımını kullanır (Yılmaz vd., 2006). Kalman filtresi, Gauss dağılımı olan durumlarda doğrusal bir sistemin durumunu tahmin etmek için kullanılır.

Kalman filtrelemesi üç adımdan oluşur: önerme, doğrulama ve asimilasyon.

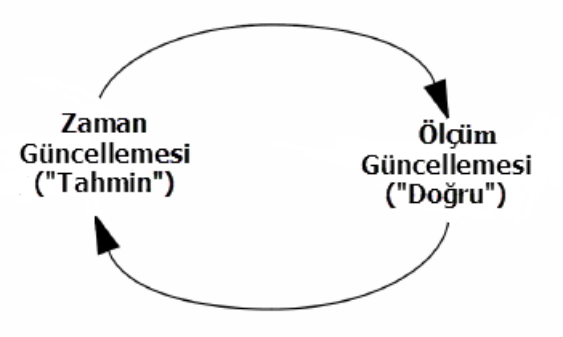
Bu algoritmada amaç, $t-1$ anındaki sistem model önermesi ile t anındaki ölçümler kullanılarak t anındaki sistemin durumunu bulmaktır.

* Önerme aşamasında, $t-1$ anındaki sistem modeline ve nesne durumuna dayanılarak t anındaki nesnenin durumu önerilir.

* Ölçüm aşamasında, t anındaki görüntü üzerindeki özellikler çıkarılır. Bu özellikler, nesne durumunun doğrusal dönüşümü olarak kabul edilir.

* Asimilasyon aşamasında ise, önerilmiş durumlar ile ölçülmüş durumlar kombine edilerek nesnenin yeni durumu çıkarılır (Neil Alldrin).

Zaman güncelleme, zaman içindeki durum tahminini öngörür. Ölçüm güncelleme , o andaki mevcut ölçüm ile tahmin edilen durumu öngörür.



Şekil 3.8 Kalman filtre döngüsü

Önceki (x_{k-1}) ve sonraki (x_k) durum tahmin hesaplama eşitlikleri:

Zaman güncelleme eşitliği (tahmin):

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.7}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3.8}$$

Ölçüm güncelleme eşitliği (doğru):

$$K_k = P_k^- H^T \left(H P_k^- H^T + R \right)^{-1} \tag{3.9}$$

$$\hat{x}_k^- = \hat{x}_k^- + K_k \left(z_k - H\hat{x}_k^- \right) \tag{3.10}$$

$$P_k = \left(I - K_k H \right) P_k^- \tag{3.11}$$

Kalman filtresinin, tek şekilli ve Gauss olasılık dağılımının olduğu durumlarda çalışması bu algoritmanın önemli bir sınırlamasıdır. Çünkü genellikle nesneler Gauss dağılımlı değildir.

Kalman filtresinin başka bir dezavantajı da karmaşık arka planlara karşı hassas olmasıdır. Kalman Filtresi, bir sonraki görüntü üzerinde nesnenin yeri hakkında tek bir tahmin yapar. Bu sebepten ötürü Kalman Filtresi ile yalnızca tek bir nesnenin takibi yapılabilir.

Kalman Filtresi yönteminde hedef nesne, nokta ile temsil edilmektedir. Eğer hedef nesne geometrik bir şekil ile temsil edilebilseydi takip işlemi sırasında nesnenin boyutlarındaki değişiklik ölçülebilirdi. Bu nedenle Kalman Filtresi yöntemi ile yapılan nesne takibinde hedef nesnenin boyutları, kapladığı alan ölçülemez yalnızca görüntü üzerindeki yeri bulunabilir.

3.2.5 Koşullu Yoğunluk Yayılımı (KYY) Yöntemi ile Nesne Takibi

Koşullu Yoğunluk Yayılımı (KYY) (CONDENSATION - Conditional Density Propagation) algoritmasının temel kullanım alanı karmaşık bir ortamda hareket eden nesnelerin dış çizgilerinin bulunması ve takip edilmesidir. Görüntü üzerindeki piksellerin hangilerinin nesnenin kenarlarına ait olduğunu bulmak önemli bir problemdir.

KYY bu problemi çözmeye çalışan olasılıksal bir algoritmadır (Isard M., Blake A., 1998). Bu algoritma, parçacık filtresinin bir varyasyonu olup doğrusal olmayan, Gauss olmayan takip problemlerinin de üstesinden gelmektedir.

Algoritmanın farklı yönlerinden biri görüntüdeki her bir piksel üzerinde işlem yapmıyor olmasıdır. Bunun yerine, işlenecek pikseller rastgele seçilmekte ve yalnızca bu piksellerin bir alt kümesi işleme sokulmaktadır.

KYY, her bir iterasyonunda üç tane adım içeren iteratif bir algoritmadır. Bu adımlar sırayla önerme, güncelleme ve yeniden örneklemedir. KYY, sıralı Monte Carlo önem örneklemesini kullanan bir Bayes özyinelemeli tahmincidir (Barrera vd., 2005).

Kısaca anlatılacak olursa, sıralı gözlemler kullanılarak o anki çok boyutlu durum tahmin edilir. Gözlemler, durum ile olasılıksal gözlem modeli üzerinden ilişkilidir. Durum dinamik olabilir, bu dinamizm hareket modeli içerisinde yakalanır. Algoritmanın sıralı doğası iteratif eşitlikler sağlar, örnekleme doğası ise N parçacıktan oluşan kümeyi yönetme yetisi sağlar. Her bir parçacık bir durum tahminini temsil eder ve buna ilişkilendirilmiş bir ağırlığa sahiptir.

Global tahminler parçacık kümesinin tamamına dayanarak yapılabilir. KYY'nin her iterasyonundaki önerme adımında, yeni bir parçacık elde edilerek ve parçacık kümesi oluşturularak her bir parçacığın hareket modeli örneklenir.

Güncelleme adımında her bir parçanın ağırlığı gözlem modeli kullanılarak hesaplanır. O anki gözlemle aynı sonucu üreten parçacıkların ağırlıkları arttırılır. Yeniden örnekleme adımında, o anki parçacıkların ağırlıklı dağılımının örneklemesinden yararlanılarak yeni bir parçacık kümesi oluşturulur. Ağırlığın yüksek olması, parçacığın bir sonraki kümede yer alma olasılığının yüksek olması demektir.

KYY algoritmasının dezavantajları; olasılık tavan değerine çıktığında veya yeni ölçümler öncekilerin kuyruğunda yer aldığında başarısız olduğu görülebilir (Yalçın ve Gökmen, 2005).

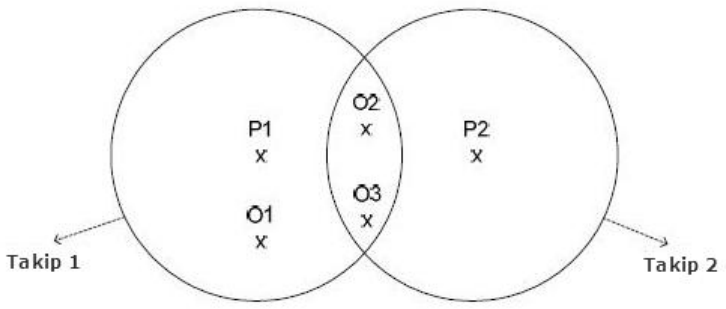
3.3 Çoklu Hedef Takibi

Hemen hemen bütün görsel izleme sistemleri önemli bir kısım olan çoklu hedeflerin takip edilmesi durumu ile uğraşır. Hareketli nesnelerin tespit edilmesinin bir sonraki aşamasıdır.

Çoklu hedef takip yöntemleri her bir görüntü sahnesindeki gözlemleri değerlendirir ve gözleme yeni bir hedef girişi veya var olan hedefte güncellemeler olup olmadığına karar verir. Bu kararı verirken hedeflerin tahmin edilen bölgeleri içindeki gözlemler olması dikkate alınır.

En popüler ve temel hedef birleştirme problemi kapalı alan içerisindeki 1.hedefin ve 2.hedefin tahmin edilmiş bölgelerinde paylaştıkları iki gözlemin bulunma durumudur (Şekil 3.9).

$P1$ ve $P2$ hedeflerin tahmin edilen konumunu, $O1$, $O2$ ve $O3$ gözlemlerin konumunu gösterir.



Şekil 3.9 Çoklu hedef durum gösterimi

Bu iletişim sorununun çözümünde Global Yakın Komşuluk Yaklaşımı (GYKY), Ortak Eklemli Veri Birleşimi (OEVB) ve Çoklu Hipotez Takibi (ÇHT) yöntemleri geniş oranda kullanılmaktadır.

3.3.1 Global Yakın Komşuluk Yaklaşımı (GYKY)

Global Yakın Komşuluk Yaklaşımı (GYKY) (GNN - Global Nearest Neighbor yaklaşımı), $O1$'i 1.hedef ile $O2$'yi ($O2$'nin $O3$'e göre 2.hedefe daha yakın olduğunu düşünerek) 2.hedef ile bağlantılandıracaktır ve $O3$ yeni bir hedef olarak kabul edilecektir.

Global Nearest Neighbor yaklaşımında, takip düzeni genelde 5(değişebilir) ardışık sahne içerisindeki izleme tespiti ile yapılır. Ondan sonra tüm izlerin bir sonraki durumları tahmin edilir.

Kalman filtresi tahmin değişimi, hedeflere geçit sağlamak ve onları bağlantılandırmak için gerekli olan belirsizliği sağlamaktadır.Hedefler ne kadar uzak olursa o kadar iyi sonuç verir. Yakın hedeflerde çalışma performansını iyileştirmek için Kalman filtresi değişim matrisi arttırılarak belirsizlik yansıtılmalıdır.

3.3.2 Ortak Eklemli Veri Birleşimi (OEVB)

Ortak Eklemli Veri Birleşimi (OEVB) (JPDA - Joint Probability Data Association), hedefe yönelik bir yaklaşımdır. Bilinen sayıda hedef olduğunda hedef özelliklerinin ölçümünü değerlendirir ve bu özelliklere ilgili durum tahminlerini ekler.

OEVB algoritmalarının başlıca sınırlamaları görüş alanına giren yeni nesneler ile veya daha önceden takip edilen nesnelerin görüş alanı dışına çıkma durumları ile başa çıkmada yetersizlikleridir.

OEVB yöntemi hedefin tüm gözlemlerinin kapısı içerisindeki ağırlıklı toplamı ile güncellenmesini önerir.1.hedef $O1$, $O2$, $O3$ ile ve 2.hedef $O2$ ve $O3$ ile güncellenir.

3.3.3 Çoklu Hipotez Takibi (ÇHT)

Çoklu Hipotez Takibi (ÇHT)(MHT - Multi Hypothesis Tracking) algoritması çoklu nesne takibi durumu için Reid tarafından geliştirilmiştir. ÇHT yaklaşımı ölçüme yöneliktir ve yerleşik hedef veya yeni bir hedef olasılığı kesın ölçüm dizilerinin ele alınmasına yol açar. ÇHT yaklaşımı ölçüm dizilerinin ilişkilerini dikkate alır ve tüm dizinin(hipotez) olasılıklarını değerlendirir. Bu durum zamanla exponansiyel olarak artan hesap karmaşıklığına yol açar.

Yaratılacak hipotez sayısını sınırlandırmak için birleştirme ve kaldırma gibi uygun teknikler kullanılmalıdır. İlk yaklaşım, sahneden sahneye hipotez yapısını sürdürür ve durmadan yeni veri alındıkça hipotezi genişletir ve değiştirir. Her bir sahnedeki hipotez seti, bir önceki sahneden bir sonrakine taşınır ve hipotezdeki tüm izlenimlerle uyumlu olan bir veya daha fazla izlenimden oluşur.

Alternatif yaklaşımda,(Örten, 2005) her sahnede şekillenen izlenimler hipotezlere dönüştürülür ve kaldırma sonrası kalan izlenimler işlemin devam ettiği sonraki sahnede tahmin edilir. İzlemeye yönelik yaklaşımın önemli avantajı hipotezin oluşumu daha yüksek kaliteli izlenimlere karşı sınırlandırılabilmesidir. Düşük skorlu izlenimler hipotez şekillenmeden önce silinir ve bu özellik sayesinde hesap yükü azaltılır.

4. ARAÇ TAKİBİ

4.1 Gauss Karışımı(GK) Yöntemi

Arka plan modeli, karmaşık durumlar için sahnedeki kademeli ve hızlı değişimlere karşı uyum sağlamalıdır. O halde, arka plan modeli çoklu model dağılımlar ile başa çıkabilmelidir.

Takip sistemleri aynı zamanda aydınlatma değişimlerine karşı güçlü olmalıdır. Hem dağınık bölgelerdeki hareketleri çözebilmeli hem de sallanan ağaçlar gibi tekrarlanan hareketlerin ve çıkarılan nesnelerin üstesinden gelebilmelidir. Geleneksel yöntemler bu durumlarda başarısızdır. Gauss karışım modeli, arkaplan değişimlerine karşı uyum sağlayan bir arka plan modeli yaratır (Stauffer ve Grimson, 1999).

Gauss karışım model yaklaşımı, bütün arka planı dağılımın belli kısımları ile modellemek yerine her bir pikseli gauss karışımı olarak modeller. Her piksel, onu en etkili şekilde temsil eden gauss dağılımının arka plan modeline uygun olup olmamasına göre sınıflandırılır.

Hangi gauss dağılımının arka plana uygun olduğu, sürekliliğine ve değişimine bağlı olarak karar verilir. Arka plan dağılımlarına uymayan piksel değerleri, bunun aksini yeterli ve tutarlı kanıtlarla ispatlayan Gauss dağılımı olmadığı sürece ön plan olarak düşünülür. Tekrarlanan hareketler sistem tarafından öğrenilir. Bu yöntemin önemli iki parametresi vardır, öğrenme sabiti ve arka plan tarafından açıklanması gereken veri oranıdır.

Gauss dağılım parametrelinin her güncellenmesinde, arka plan modeli işleminin muhtemel bir parçası olduğu varsayımının yapılmasına Gauss dağılımlarının basit deneme yanılma kullanılarak değerlendirilmesiyle karar verilir.

Arka plan piksel dağılımları ile eşleşmeyen piksel değerleri ön plan olarak belirlenir ve bu piksellerin gruplandırılması ile hareketli nesneler bulunmuş olur.

K değeri, hesaplama gücü ve hafıza durumunun uygunluğuna göre belirlenir ve genelde 3 ile 5 değeri arasında değişir. Her pikselin son geçmişi $\{X_1,...,X_t\}$, K Gauss karışımı ile modellenir. Anlık piksel değeri gözlenme olasılığı ;

$$P(X_t)=\sum_{i=1}^{K} w_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{4.1}$$

$w_{i,t}$ tahmini ağırlık(bu gauss dağılımı ile hangi oranda verinin açıklanması gerektiği) ve bu ağırlığın t anındaki karışımda bulunan i.gauss dağılımı, bu dağılımdaki $\mu_{i,t}$ ortalama değer ve $\Sigma_{i,t}$ kovaryans matrisidir.

η Gauss olasılık yoğunluk fonksiyonu;

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t-\mu_t)^T \Sigma^{-1}(X_t-\mu_t)} \tag{4.2}$$

Hesaplama sonuçlarından, kovaryans matrisinin olduğu varsayılan biçimi ;

$$\Sigma_{k,t} = \sigma_k^2 I \tag{4.3}$$

Bu kırmızı, yeşil ve mavi piksel değerlerinin bağımsız olduğunu ve aynı değişimlere sahip olduğunu farz eder. Bu yüzden, sahnedeki her bir pikselin son gözlenen değerlerinin dağılımı Gauss karışımı ile tanımlanır.

K dağılımlarının t anındaki öncelikli ağırlıkları aşağıdaki eşitliğe göre düzenlenir.

$$w_{k,t} = \alpha * M(k,t) + (1-\alpha) * w_{k,t-1} \tag{4.4}$$

α öğrenme oranı, $M(k,t)$ eşlenen modeller için "1" ve geri kalan modeller için "0".

Ortalama u ve değişim r parametreleri eşlenmeyen dağılımlar için aynı kalır.

Yeni gözlemlerle eşlenen dağılım parametreleri aşağıdaki gibi güncellenir :

$$\mu_t = (1-\rho)\mu_{t+1} + \rho X_t \tag{4.5}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T (X_t - \mu_t) \tag{4.6}$$

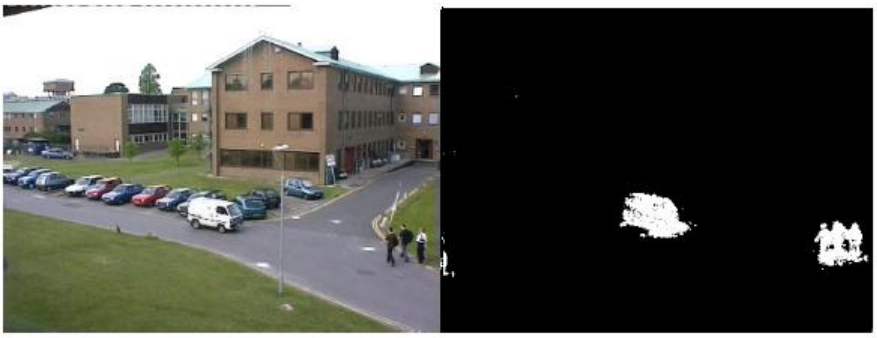
Sahnedeki herhangi bir değişimi modele yansıtarak ölçümler düzenlenmelidir. Gauss dağılımları değerine göre sıralanır. Bu değer, dağılım kazancı daha fazla olduğunda ve değişim azaldığında artış gösterir. En muhtemel arka plan dağılımları listenin başında, sürekli değişen ve daha az muhtemel olanlar aşağıya doğru dizilir.

Sonra, ilk B dağılımları arka plan modeli olarak seçilir.

$$B = \arg\min_b \left(\sum_{k=1}^{b} w_k > T \right) \tag{4.7}$$

T , arka plan tarafından açıklacak en düşük veri oranı ölçüsüdür. T değeri küçük seçildiğinde arka plan genelde tek modeldir, büyük seçildiğinde tekrarlanan arka plan(ağaçtaki yapraklar, rüzgardaki bayrak gibi…) hareketinden kaynaklanan çoklu model dağılımında birden fazla renk içererek sonuçlanır. Bu durum, iki veya daha farklı rengin arka plan tarafından kabul

edilmesine izin veren şeffaflık etkisidir. Şekil 4.1'de Gauss Karışımı uygulaması gösterilmiştir.



(a) (b)

Şekil 4.1 Gauss karışımı uygulaması sonucu a) orjinal resim b) Gauss karışımı sonrası

Gauss karışımı modeli uygulama adımları;

1. Gauss dağılımı bileşenleri sayısı, arka plan bileşenleri sayısı, pozitif sapma eşik değeri, öğrenme oranı, ön plan eşik değeri ve ilk sapma eşik değeri belirlenir.

2. Her bir sahnedeki her Gauss dağılımı için mevcut piksel değerleri ile ortalama Gauss dağılım değerleri ve standart sapmaları bulunur. Bu hesaplanan değerler standart sapma değerleri ile karşılaştırılır. Fark değeri belirlenen limitler arasında ise Gauss bileşenleri güncellenir. Eğer bu durum sağlanmıyor ise yeni Gauss bileşenleri yaratılır.

3. Her bir sahnedeki her Gauss dağılımı için, kırmızı-yeşil-mavi standart sapma değerlerinin aritmetik ortalaması kullanılarak bileşen mertebeleri hesaplanır. Hesaplanan mertebeler sıralanır.

4. Ön plan piksellerinin belirlenmesi, piksel değerinin(mevcut piksel ve Gauss dağılım ortalamaları farkı) mertebelenmiş Gauss değerinin 2,5 standart sapma değeri içinde olup olmamasına göre yapılır.

4.1.1 Gölge Çıkarımı

Hareketli nesne ayıklanmasında gölgeler problemli kısımlardır. Gölgeler hatrı sayılır derecede parlaklık değişimine neden olabilirler. Bu yüzden gölgeler tarafından etkilenen sahne bölümleri ön plan nesneleri olarak seçilebilir.

Gölge çıkarımı görsel izleme sistemlerinde önemli bir rol oynar. Uygulama basitliği ve hesaplamalarındaki verimlilikten dolayı aşağıdaki algoritmanın (Salvador vd., 2001) uygulanması önerilmiştir.

Algoritmanın temel mantığı, gölgelerin arka plan parlaklığını değiştirmesine rağmen normalleştirilen parlaklık değerlerinin arka plan değerleriyle hemen hemen aynı olmasıdır.

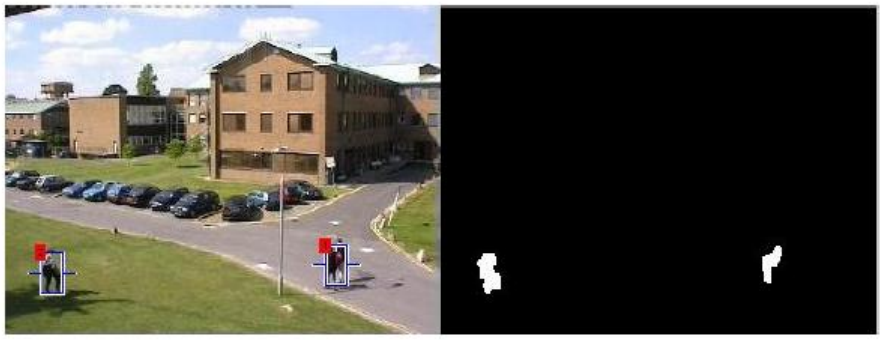
$$\frac{R_s}{R_s+G_s+B_s} \cong \frac{R}{R+G+B}$$

$$\frac{G_s}{R_s+G_s+B_s} \cong \frac{G}{R+G+B} \qquad (4.8)$$

$$\frac{B_s}{R_s+G_s+B_s} \cong \frac{B}{R+G+B}$$

$$I_s(x,y) = \alpha I(x,y) \qquad (4.9)$$

$I(x,y)$, (x,y) noktasındaki parlaklık değeri ve "s" simgesiyle ifade edilenler gölge sonrası değerleri belirtir. Şekil 4.2'de gölge çıkarım sonucu örnekle gösterilmiştir.



a) b)

Şekil 4.2 Gölge çıkarımının sonucu a) orijinal resim b) gölge çıkarımı sonrası

4.1.2 Morfolojik İşlemler

Morfolojik işlemler, hareketli nesne bölütlenmesi işleminden elde edilen görüntüdeki gürültülerin ortadan kaldırılması için kullanılır. Hareket bölütleme işleminden sonra "1" ve "0" piksel değerlerini içeren ön plan görüntüsüne sahip oluruz.

Morfolojik işlem, görüntü üzerinde hareket ettirilen ve her bir pikselinin görüntü elemanları ile karşılaştırıldığı bir matristir. İki setteki piksellerin eşleşmesi koşulu sağlandığında yapısal eleman merkezinin altında kalan piksel değerine daha önceden belirlenen bir değer atanır.

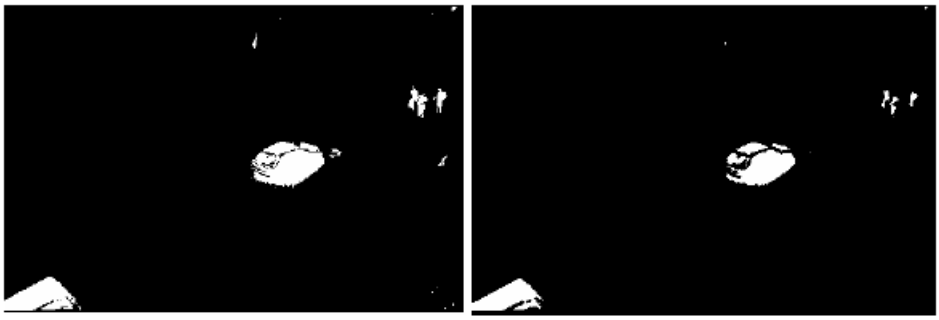
Aşınma ve genişleme, binary görüntülere uygulabilen temel morfolojik işlemlerdir.

4.1.2.1 Aşınma işlemi

Aşınma işlemi, ön plan piksel görüntüsünü üzerindeki küçük gürültülerden arındırır.

$$SE_{aşınma} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{4.10}$$

Giriş görüntüsündeki her ön plan pikseli yapı eleman matrisinin merkezi ile aynı hizaya getirilir. Beyaz pikseller tarafından tam olarak çevrelenmemiş ön plan pikselleri silinmiştir (Şekil 4.3). Sonuç olarak ön plan bölgeleri küçülmüş ve içindeki boşluklar büyümüştür.



(a) (b)

Şekil 4.3 Aşınma işleminin etkisi a) Arkaplan çıkarımı sonrası b) Aşındırılmış görüntü

4.1.2.2 Genişleme İşlemi

Genişleme; kalınlaştırma ve büyütme işlemidir. Aşınma işleminden sonra uygulanarak ön plan görüntüsünde kaybolan orijinal nesne sınırlarını geri getirir. Aşınma işlemi ile arınılan gürültüler genişleme işlemi sonrası geri gelmezler (Şekil 4.4).Genişleme işlemi yapı matrisi:

$$SE_{yayma} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{4.11}$$

(a) (b)

Şekil 4.4 Genişleme işleminin etkisi a) Aşındırılmış görüntü b) Genişlemiş görüntü

Görüntüye önce aşınma, sonra genişleme işlemi uygulanmasına Açma (opening) denir.Bu yöntemle görüntüdeki küçük nesneler ve boşluklar ortadan kaldırılır,dış çizgiler yumuşatılır.

$$A \circ B = (A \Theta B) \oplus B \tag{4.12}$$

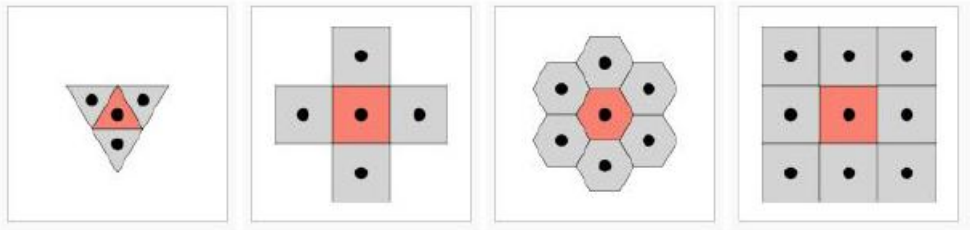
Önce genişleme, sonra aşınma işlemi uygulanmasına Kapama (closing) denir.Bu yöntem küçük boşlukları doldurur.

$$A \bullet B = (A \oplus B) \Theta B \tag{4.13}$$

4.1.3 Bağlı Bileşenler Etiketlemesi

Hareket eden nesneleri ayrı ayrı bulabilmek için bağlı bileşenler etiketlemesi kullanılır. Grafik teorisinde (Gibbons, 1985), eğer iki tepe noktası birbiri ile bağlantılı ise ve bu noktalara bağlı başka bir tepe noktası yoksa, bu tepe noktaları "bağlı bileşenler"dir (Asano vd., 1996).

Bağlı olduğu "n" nokta sayısına sahip ise göre n-bağlılık olarak isimlendirilir.



Şekil 4.5 Bağlılık örnekleri

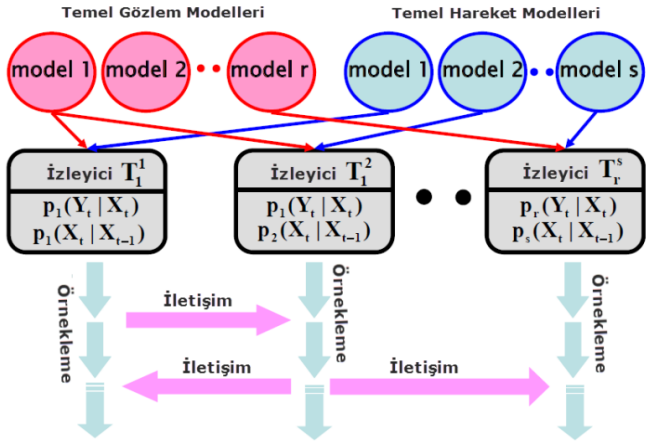
Hareket takibinde, uygun yöntemler uygulandıktan ve gürültülü pikseller temizlendikten sonra,bağlı bileşenler görüntüdeki hareketli nesnelerin net olarak görünmesini sağlar. Her bir nesne farklı isim ile etiketlenir.(Samet ve Tamminen, 1988).

Grafik teorisinde (Stauffer ve Grimson, 2000), görüntüdeki bir nokta veya nesne "damla" olarak isimlendirilir. Hareketi tespit etmek için ilk olarak ilgilendiğimiz damlanın hareketi belirlenmelidir. Bu aşamadan sonra diğer damlalar ortadan kaldırılarak işlem kolaylaştırılır. Damlaların tespiti genellikle görüntüdeki her pikselin komşu piksellerinin analiz edilmesidir.

Bağlılık sayısına göre analiz edilmesi gereken pikseller Şekil 4.5'te gösterilmiştir. Analiz edilen piksellerden hiçbiri siyah piksel değilse , bu noktalar "bölge değeri" olarak, sadece bir siyah piksel var ise , bu nokta "bölgenin elemanı" olarak, birden fazla siyah piksel var ve isimlendirilmemiş bölgede ise, tüm noktaları "bölge" olarak isimlendirilir. Birden fazla piksel grupları ve isimlendirmeler var ise onlara "diğer bölgeler" denir.

4.2 Görsel Takip Ayrıştırma Yöntemi (GTA)

Yöntemin mantığı, gözlem, hareket ve takip modellerinin temel ayırt edici bileşenlerden faydalanarak etkili birleşik modeller haline getirilmesidir. Şekil 4.6'da yöntemin çalışma sistemi gösterilmiştir.

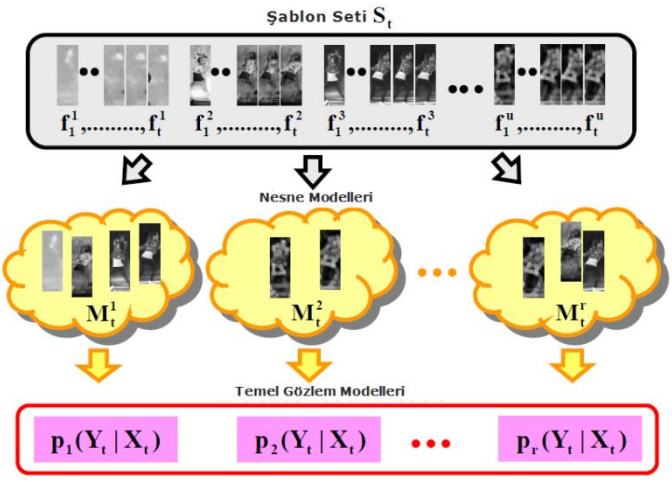


Şekil 4.6 Görsel takip ayrıştırma yöntemi çalışma sistemi

Gözlem modeli sayısı 4 (renk tonu, doygunluk, değer, kenar bilgisi) ve hareket modeli sayısı 2 (yumuşak, sert) olarak seçilmiştir. Buna göre gözlem ve hareket modellerinden oluşacak temel takip sistemimizin sayısı 8 olacaktır.Her temel takip sistemi, bir çift temel gözlem ve hareket modeli ile modellenen Markov bağı oluşturur ve MAP(Maximum a Posteriori) tahmini için Metropolis Hastings algoritması ile örnekler üretir. (4.20)

4.2.1 Temel Gözlem Modelleri

Yöntemde kullanılacak özellikler; renk tonu, doygunluk, değer, kenar bilgileridir.



Şekil 4.7 Temel gözlem modellerinin oluşturulması

Görüntü şablonlarından faydalanılarak u sayıda farklı özellik için t zamanına kadar şablon S_t seti oluşturulur. Sonra STBA yöntemi belli bir S_t şablon seti seçerek M_t^i nesne modelleri yaratır (Şekil 4.7).

$$S_t = \left\{ f_m^n \middle| m = 1,...,t, n = 1,...,u \right\} \tag{4.15}$$

S_t : şablon seti

t: görüntü parçası, sahne – zaman (t = 5)

u : özelliklerin cinsi, hue – saturation – intensity – edge (u =4)

$|S_t| = tu$ = 5.4 = 20

$|S_t|$: S_t şablon setindeki toplam özellik şablonu sayısı

4.2.2 Seyrek Temel Bileşenler Analizi (STBA)

M_t^i nesne modelinin izleme performansının ve veriminin iyi olması için üç koşul vardır.

İlk olarak M_t^i , zaman içindeki nesnedeki en fazla görünüş değişimini kapsamalıdır. İkinci koşul, bu oluşum iyi performansını sürdürürken mümkün olduğunca kompakt olmalıdır. Son koşul ise, $M_t^i, i = 1,...,r$ nesne modelleri arasındaki ilişkiler birbirini tümleyici olmalıdır.

Bu koşulları sağlayan M_t^i nesne modelinin oluşturulması için STBA metodu kullanılmıştır. STBA metodu (Aspremont vd., 2007) verilen A_t gramian matrisinde seyrek c temel bileşenlerini bulur. Bu bileşenler yakalanan en büyük değişim değerinde sınırlı sayıda sıfır olmayan girdilere sahiptir.

$$maksimum \quad c^T A_t c - \rho |c|^2 \tag{4.16}$$

$|c|$, c deki sıfır olmayan girdilerin sayısıdır ve ρ bu girdilerdeki cezaları kontrol eder.

ρ değeri arttıkça, daha fazla seyrek temel bileşenlere (c^2) sahip oluruz. İzleme problem için t anındaki A_t gramian matrisinin oluşturulması:

$$\begin{aligned} A_t &= a^T a \\ a &= \left(f_1^1 \ldots f_t^1 \ldots f_1^u \ldots f_t^u \right) \end{aligned} \tag{4.17}$$

a matrisinin boyutu $|S_t|$ olduğu için, A_t gramian matrisinin boyutu $|S_t| \times |S_t|$ değerindedir.

Geleneksel dışbükey optimizasyon araçları (Aspremont vd., 2007) ile, (4.16) denklemine yaklaşık olarak etkili çözümler sağlayabiliriz. Bu bileşenlerin arasından , azalan özdeğerlere göre r tane temel bileşen seçeriz. $c_i, i = 1,...,r$. Seçilen bileşenler tarafından (4.24)'deki her bir M_t^i nesne modelinin oluşumu:

$$M_t^i = \left\{ f_m^n \,\middle|\, f_m^n = a(x), c_i(x) \neq 0 \right\} \tag{4.18}$$

Eğer c_i'nin x.elementi sıfır olmayan değere sahipse, M_t^i nesne modeli (4.17)'deki a matrisinin x.sütununda bulunan f_m^n şablonunu içerir. Her model anlamlı özvektörlerden oluştuğu sürece her bir M_t^i nesne modeli, nesnedeki önemli görünüş değişimlerini yakalar.

Az sayıda şablon olduğunda özvektörün seyrekliği modele yoğunluk katar. Özvektörler dikey yapıya sahip olduğunda dolayı, nesne modelleri arasında tümleyici ilişki vardır.

$$f_m^n = \frac{F^n\left(I\left(\hat{X}_m\right)\right)}{\left\|F^n\left(I\left(\hat{X}_m\right)\right)\right\|}, m = 1,...,t, n = 1,...,u \tag{4.19}$$

f_m^n : m anındaki n.tip özellik şablonu

F^n : özellik çıkarıcının n.tipi

$I\left(\hat{X}_m\right)$: m anındaki görüntü parçası

Nesnenin en iyi yapılanmış şekli $\hat{X}_t$, her t anındaki N sayıda örnek üzerinden Maximum a Posteriori (MAP) tahmini yapılarak elde edilebilir.

$$\hat{X}_t = \underset{X_t^{(l)}}{\arg\max}\, p\left(X_t^{(l)} \middle| Y_{1:t}\right) \; for \; l = 1,...,N \tag{4.20}$$

$X_t^{(l)}$: X_t durumunun l.örneği

Verilen sabit sayıdaki örneklerde , $p\left(X_t \middle| Y_{1:t}\right)$ soncul olasılığı tam değer verdiğinde, (4.20)'deki MAP tahmini doğruluk derecesi artar.

$$p\left(X_t \left| Y_{1:t}\right.\right) \alpha\ p\left(Y_t \left| X_{1:t}\right.\right) \int p\left(X_t \left| X_{t-1}\right.\right) p\left(X_{t-1} \left| Y_{1:t-1}\right.\right) dX_{t-1} \tag{4.21}$$

Yöntem, (4.21)'deki $p\left(X_t \left| Y_t\right.\right)$ gözlem modelinin ve $p\left(X_t \left| X_{t-1}\right.\right)$ hareket modelinin ayrıntılarını kullanarak posterior olasılığının tam değerini elde eder.

Gözlem modeli :

$$p\left(Y_t \left| X_t\right.\right) = \sum_{i=1}^{r} w_t^i p_i\left(Y_t \left| X_t\right.\right), \sum_{i=1}^{r} w_t^i = 1 \tag{4.22}$$

r : gözlem modelindeki temel bileşenlerin sayısı

$p_i\left(Y_t \left| X_t\right.\right)$: i.temel gözlem modeli ($i = 1,...,r$)

w_t^i : t anındaki ağırlık değişkeni

Hareket modeli :

$$p\left(X_t \left| X_{t-1}\right.\right) = \sum_{j=1}^{s} w_t^j p_j\left(X_t \left| X_{t-1}\right.\right), \sum_{j=1}^{s} w_t^j = 1 \tag{4.23}$$

s : hareket modelindeki temel bileşenlerin sayısı

$p_j\left(X_t \left| X_{t-1}\right.\right)$: j. Temel hareket modeli ($j = 1,...,s$)

w_t^i : t anındaki ağırlık değişkeni

(4.22)'deki her temel gözlem modeli $p_i\left(Y_t \left| X_t\right.\right)$, kendisinin t anındaki M_t^i nesne modelini S_t şablon setinden alır.

$$M_t^i \subset S_t, i = 1,...,r \tag{4.24}$$

Ve, bu eşitlik tarafından belirlenir ;

$$p_i\left(Y_t \left| X_t\right.\right) = \exp^{-\lambda DD\left(Y_t, M_t^i\right)}, i = 1,...,r \tag{4.25}$$

λ : ağırlık parametresi ('5')

Y_t : F^n özellik çıkarıcısı ile elde edilen u adet gözlem

DD fonksiyonu t anındaki Y_t gözlem modeli ve M_t^i nesne modeli arasındaki yayılmayı geri getirir. Yayılım uzaklığını, gözlemin niceleme etkilerinin yanı sıra deformasyona karşı güçlü olduğu sürece farklılık ölçüsü olarak değerlendiririz. Çünkü Y_t çoklu gözlemlerden ve M_t^i çoklu şablonlardan meydana gelir.

$DD\left(Y_t, M_t^i\right)$ Y_t 'deki her bir gözlem ve M_t^i 'deki her bir şablon arasındaki farklılıkların toplamı olarak hesaplanır. (4.25)'deki $p_i\left(Y_t \middle| X_t\right)$ tasarımını bitirmek için, geriye kalan iş r sayıda farklı $M_t^i, i = 1,...,r$ setlerinin elde edilmesidir.

4.2.3 Temel Hareket Modelleri

(4.23)'deki her bir $p_j\left(X_t \middle| X_{t-1}\right)$ temel hareket modeli, farklı değişkenlikli Gauss karışımı tarafından yapılan farklı tip hareketleri tanımlar.

$$p_j\left(X_t \middle| X_{t-1}\right) = G\left(X_{t-1}, \sigma_j^2\right), j = 1,...,s \tag{4.26}$$

G , X_{t-1} ortalamalı ve σ_j^2 değişkenlikli Gauss dağılımını temsil eder. Nesnenin hareketinin yumuşak ve sert olmak üzere iki tür harekete dönüştürülebildiğini farz ederek $p_1\left(X_t \middle| X_{t-1}\right)$ ve $p_2\left(X_t \middle| X_{t-1}\right)$ hareket modellerini meydana getiririz. $p_1\left(X_t \middle| X_{t-1}\right)$,küçük σ_1^2 ile yumuşak hareketleri açıklar. Bu tür hareket modeli, yerel minimum yakınındaki görünürde iyi hareketleri daha ayrıntılı canlandırır. Diğer taraftan $p_2\left(X_t \middle| X_{t-1}\right)$, yüksek σ_2^2 li sert hareketleri kapsar. Bu durumda model, çok keşfedilmemiş hareketleri daha ayrıntılı canlandırır.

4.2.4 Temel İzleyici Modeli

Bileşik izleyici, $r \times s$ sayıda temel izleyici tarafından oluşmuştur. $T_i^j, i = 1,...,r, j = 1,...,s$ ve tüm $p_i\left(Y_t \middle| X_t\right)_{i=1,...,r}$ gözlem modeli çiftlerinden ve $p_j\left(X_t \middle| X_{t-1}\right)_{j=1,...,s}$ hareket modeli çiftlerinden faydalanmaktadır.

STBA yöntemi kullanılarak seçilen az sayıda güçlü temel gözlem modellerinden dolayı temel gözlem modelleri sayısı (4.15)'deki S_t şablon setinin boyutu kadar artmamıştır. Ve temel hareket modeli sayısı 2'ye sabitlenmiştir. O halde, yöntem genelde az sayıda temel izleyici ile sürdürülür ve büyük şablon setinde bile ölçeklenebilirlik bakımından iyi performans gösterir.

4.2.5 Metropolis Koşturma Algoritması

Algoritma önerme ve kabul adımları olmak üzere iki ana adımdan oluşur. Önerme adımında, öneri yoğunluk fonksiyonu ile yeni bir durum önerilir.

$$Q_j\left(X_t^{j*}; X_t^j\right) = p_j\left(X_t^{j*} \middle| X_t^j\right) \tag{4.27}$$

Q_j : (4.26)'daki j.hareket modelinden yararlanan öneri yoğunluk fonksiyonu

X_t^{j*} : t anında Q_j tarafından önerilen yeni durum

Önerilen durumda, T_i^j izleyicisi kabul adımındaki kabul oranı ile durumun kabul edilip edilmediğine karar verir.

$$\gamma_{parallel} = \min\left[1, \frac{p_i\left(Y_t \middle| X_t^{j*}\right) Q_j\left(X_t^j; X_t^{j*}\right)}{p_i\left(Y_t \middle| X_t^j\right) Q_j\left(X_t^{j*}; X_t^j\right)}\right] \tag{4.28}$$

Bu iki adım iterasyon sayısı önceden belirlenen sayıya ulaşana kadar tekrar edilir.

4.2.6 Etkileşimli Markov Zincirli Monte Carlo (EMZMC) Metodu

Örnekleme işlemi sürerken, temel izleyiciler nesnenin iyi yapılanması için kendi aralarında bilgi alışverişi yaparlar. Her bir temel izleyici farklı çift gözlem ve hareket modeli kullanmasından dolayı karşılıklı değişilen bilgiler tüm bu modellerin birleşiyle ve (4.22)'deki w_t^i ile (4.23)'deki w_t^j ağırlıklarının hesaplanmasının etkisiyle sonuçlanır.

Metodun paralel ve etkileşimli olmak üzere iki farklı kullanım durumu vardır. Paralel durumda, yöntem Metropolis Hastings algoritması gibi davranır. Etkileşimli durumda izleyiciler birbirleri arasında haberleşerek nesnenin daha iyi durumunu bulmak için sıçramalar yaparlar. Temel izleyici, T_i^j izleyicisinin durumunu kendi durumunun aşağıda belirtilen olasılık eşitliğine göre kabul eder :

$$\gamma_{interacting} = \frac{p_i\left(Y_t \middle| X_t^j\right)}{\sum_{i=1}^{r}\sum_{j=1}^{s} p_i\left(Y_t \middle| X_t^j\right)} \tag{4.29}$$

$p_i\left(Y_t \middle| X_t^j\right)$, j.hareket modelinden elde edilen i.gözlem modelinin olasılık skoru,

α , canlandırma devam ettikçe linear olarak 1'den 0.5'e kadar azalan olasılıktır.

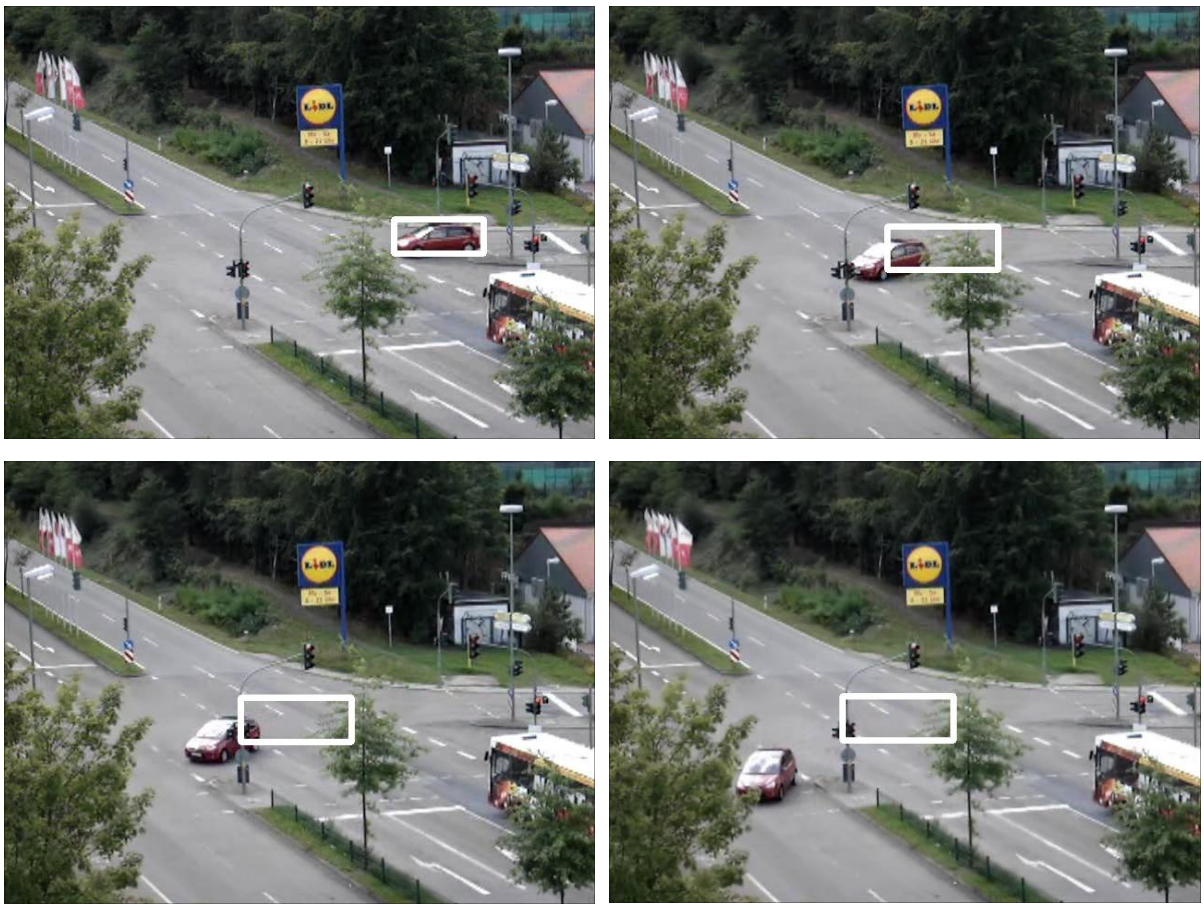
5. PERFORMANS

1- Engel olması durumu

a) GTA yöntemi ile izleme

Renkli görüntü üzerinden Görsel Takip Ayrıştırma (GTA) yöntemi kullanılarak aracın takip edilmesi istenmiştir. Test edilen görüntü, takip edilmeyi zorlaştıran; aracın şeklinin ve boyutunun değişmesi ve bir engel tarafından kısmen görülememesi durumlarını içermektedir. Yapılan test sonucu aracın takip edilme işleminin ağaç tarafından engellendiği kısmında başarısız olduğu görülmüştür.

Başlıca nedenler; aracın dönerken şekil, boyut değerlerinin azalması ve engel tarafından bir kısmının görülememesi yüzünden sistemin aracı tanıması için gerekli piksel değerine sahip olmamasıdır.

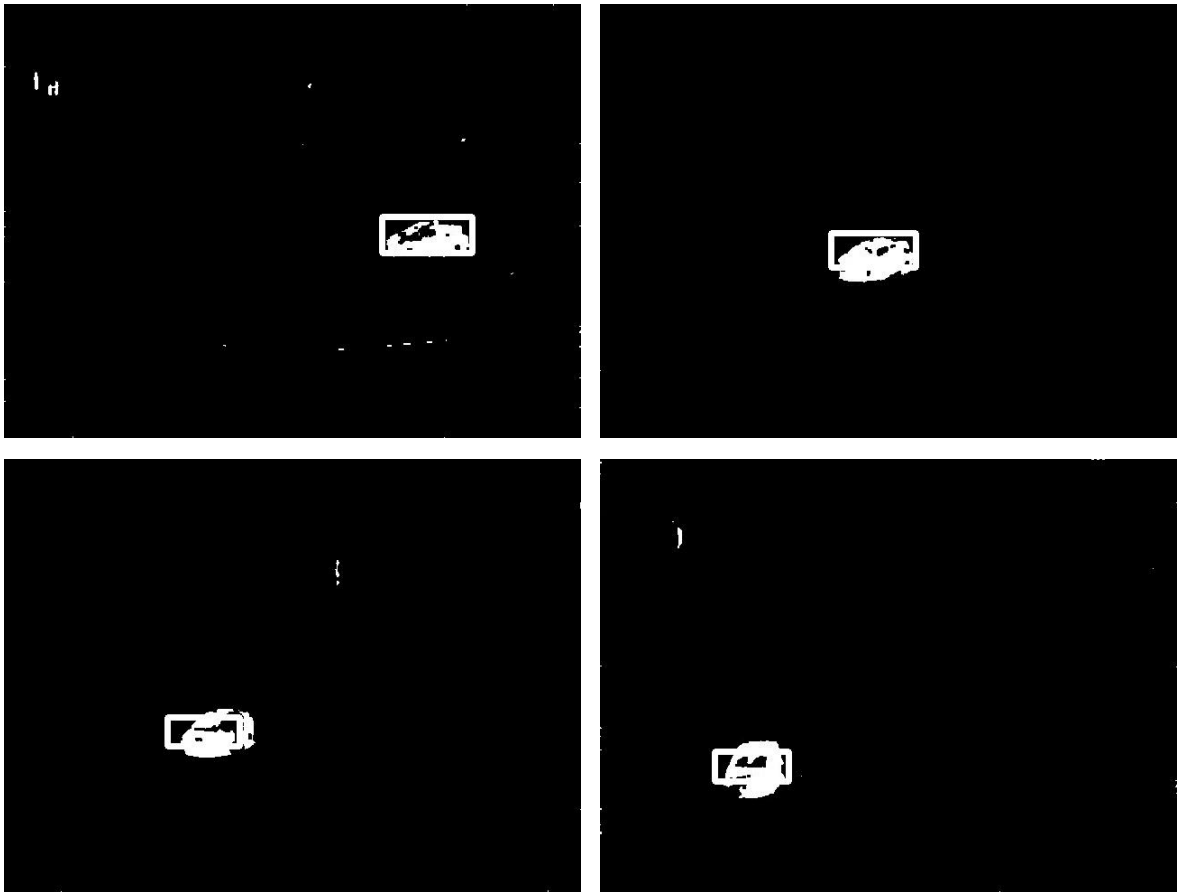


Şekil 5.1 Engel olması durumunda GTA yöntemi sonucu

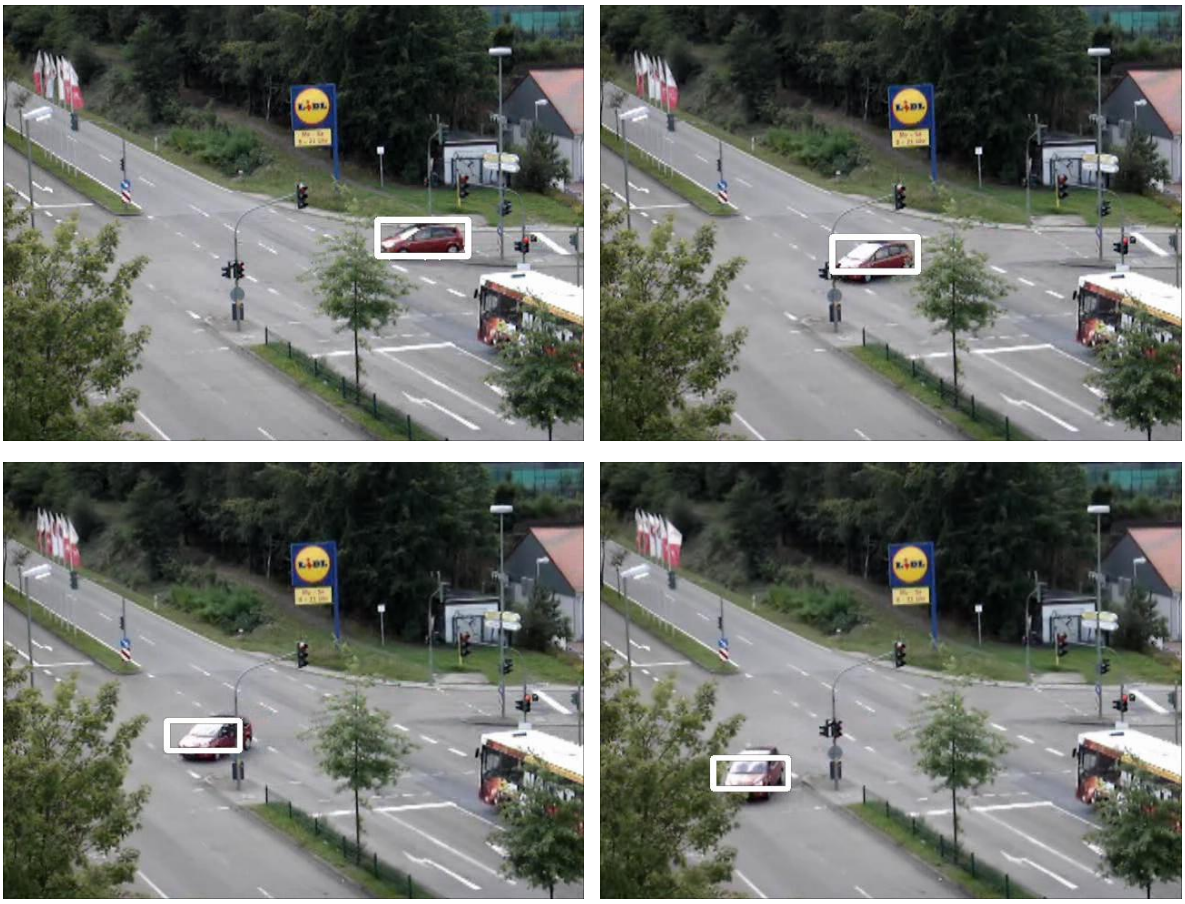
b) GK uygulandıktan sonra GTA yöntemi ile izleme

Görsel Takip Ayrıştırma yöntemi ile takip edilmeye çalışılan ve başarısız olan aynı test görüntüsü öncelikle bir arka plan modeli olan Gauss karışımı (GK) işlemine tabi tutulmuş ve hareketli cisimlerin rahatça seçilebildiği siyah-beyaz piksellerden oluşan başka bir görüntüye dönüştürülmesi sağlanmıştır.

Elde edilen yeni görüntüye Görsel Takip Ayrıştırma metodu uygulandığında aracın karşılaşılan zorluklarla başa çıkabildiği ve başarılı bir şekilde takibinin yapıldığı sonucu görülmüştür.



Şekil 5.2 Engel olması durumunda GK ve GTA yöntemi uygulanması sonucu

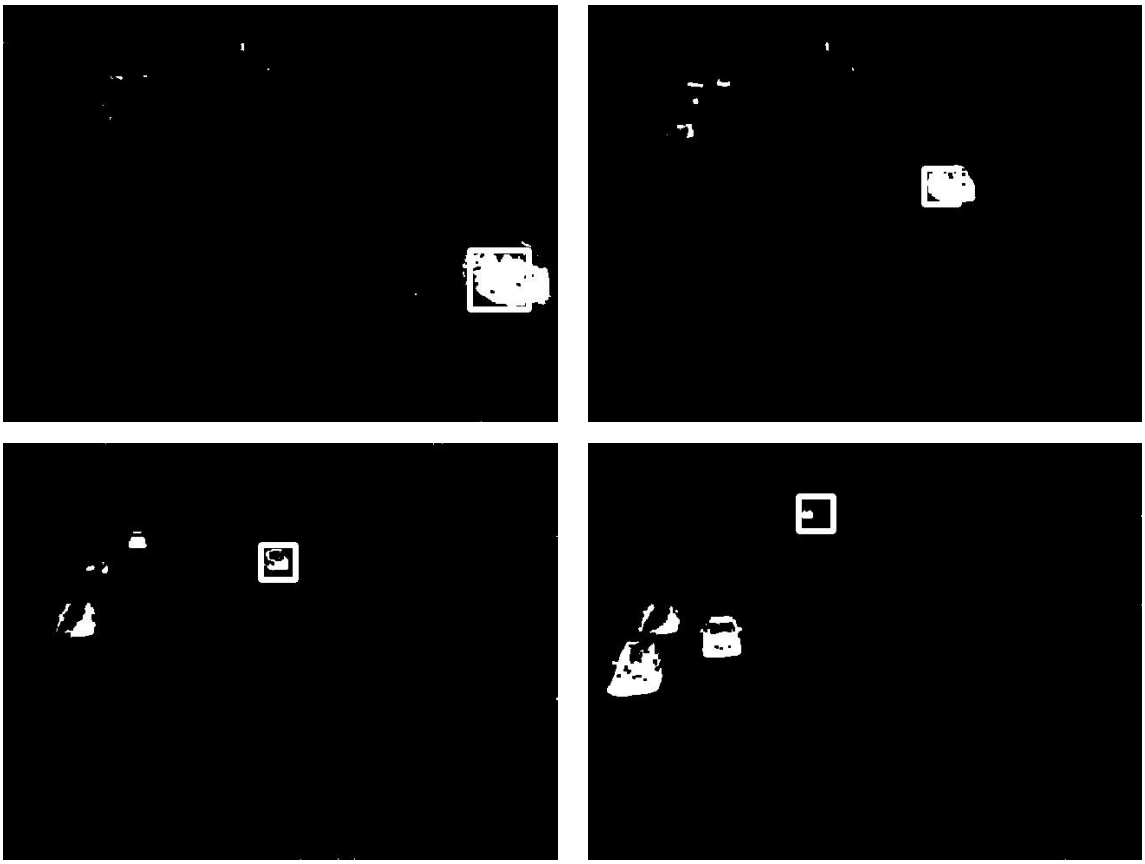


Şekil 5.3 GK ve GTA yöntemi sonucunun orjinal videoda gösterimi

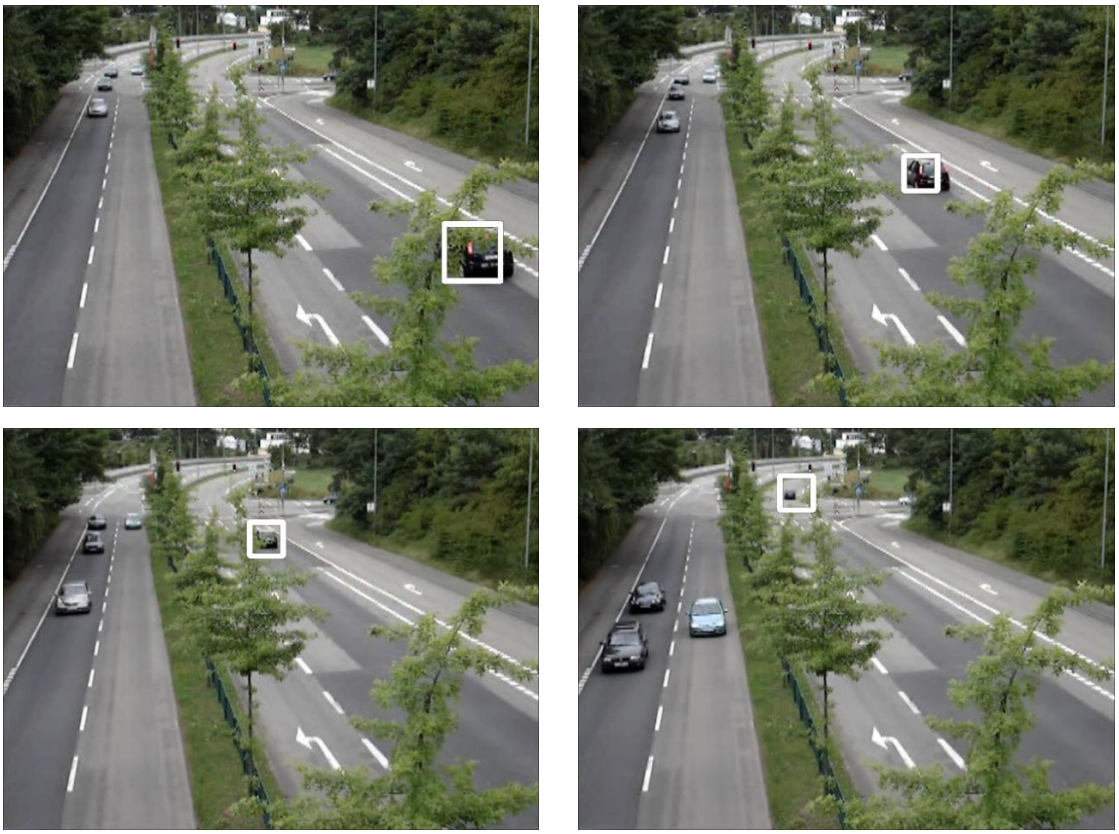
2 - Nesne boyutunun değişmesi durumu

Test edilen görüntüde birden fazla hareket eden araç bulunmakta ve takip edilmesi istenen aracın boyutlarında araç görülemeyinceye kadar süren bir azalma söz konusudur.

Uyguladığımız yöntemin takip işlemini başarıyla sağladığı görülmektedir. Takip edilen araç dışındaki hareketli araçların bulunması ve büyük orandaki boyut azalması yöntemi etkilememiştir.



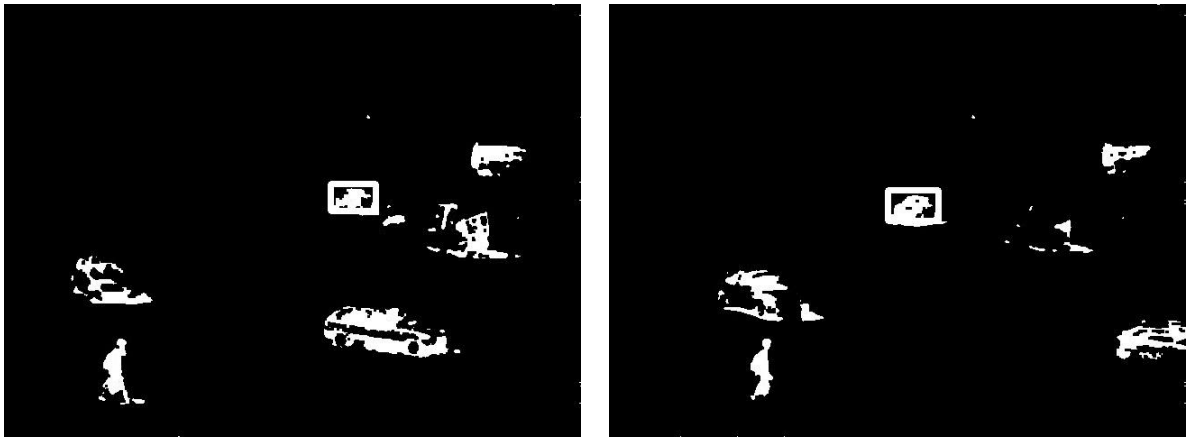
Şekil 5.4 Nesne boyutunun değişmesi durumundaki uygulama sonucu



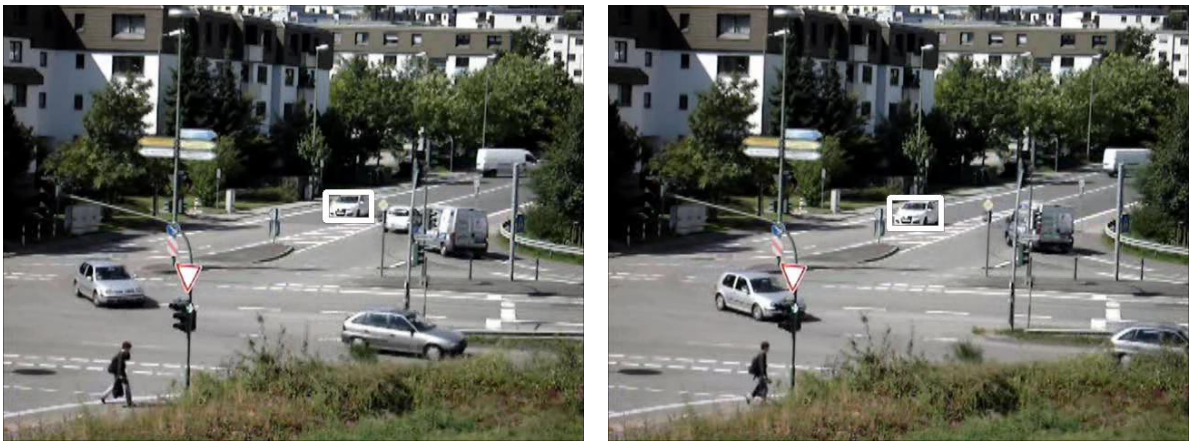
Şekil 5.5 Nesne boyutunun değişmesi uygulamasının orjinal videoda gösterimi

3 - Çoklu nesne olma durumu

Çoklu hareket içeren bir test görüntüsünde boyut ve şekil değişikliğine uğrayan, az da olsa görünüşü kapanan araç başarılı bir şekilde takip edilmiştir.



Şekil 5.6 Çoklu nesne olma durumundaki uygulama sonucu

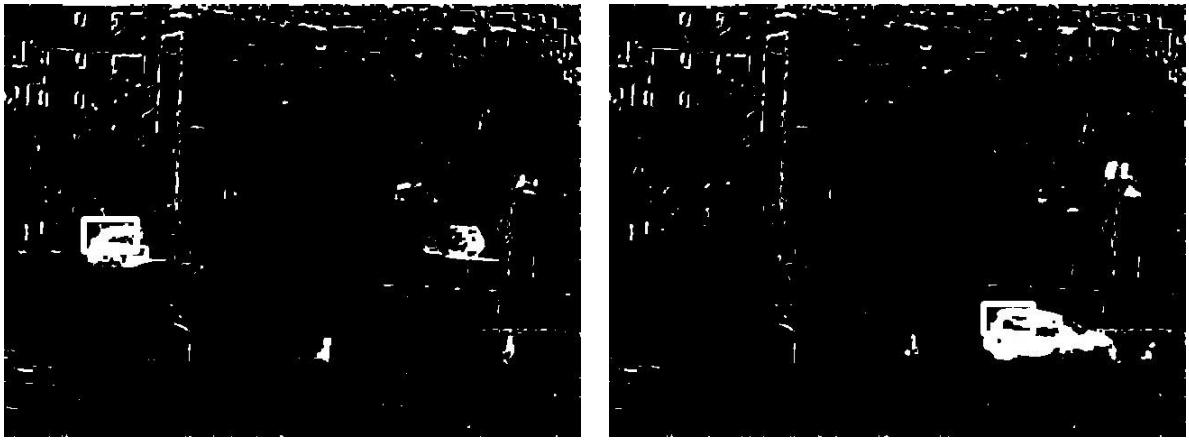


Şekil 5.7 Çoklu nesne olma uygulamasının orjinal videoda gösterimi

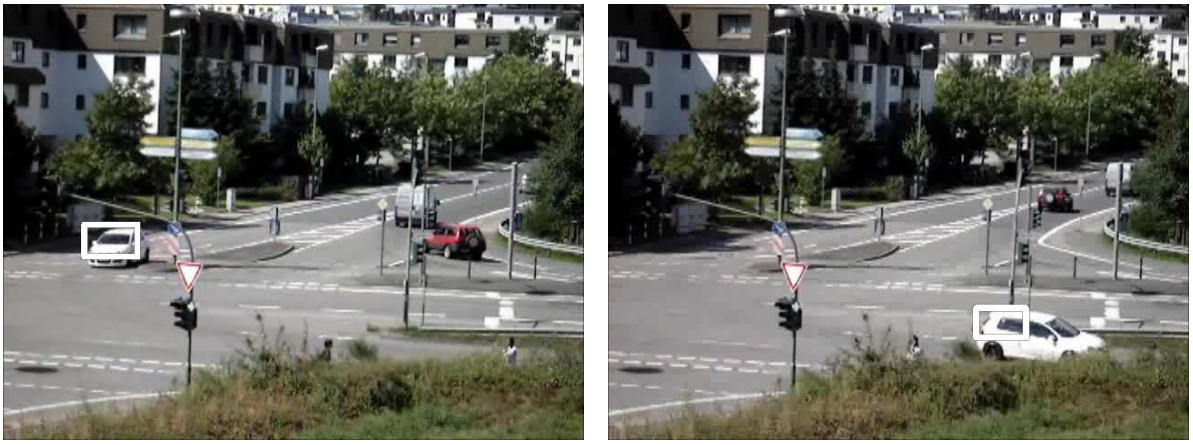
4 - Kameranın hareket etmesi durumu

Kameranın görüntü kaydı sırasında titremesi veya arka planda belli etkenler tarafından görüntü oynaması durumlarında yöntemin davranışı test edilmiştir.

Gürültü olarak görünen piksel değerleri aracın takip edilmesine bir engel teşkil etmemektedir.



Şekil 5.8 Kameranın hareket etmesi durumundaki uygulama sonucu



Şekil 5.9 Kameranın hareket etmesi uygulamasının orjinal videoda gösterimi

6. SONUÇ

Bu tez kapsamında gerçek zamanlı olmayan video görüntüleri üzerinden hareketli nesne takibi gerçekleştirilmiştir. Kullanılan video görüntüsü, günışığının yeterli olduğu bir ortamda doğal trafik akışıdır ve dijital bir fotoğraf makinesi ile kayıt edilmiştir. Seçilen görüntülerin, araç takip sistemlerinde sık karşılaşılan bazı olumsuz durumları içermesine özellikle dikkat edilmiştir:

1. Aracın boyutunun ve şeklinin değişmesi

2. Arka planın hareketli nesneler içermesi

3. Aracın bir engel tarafından görülememesi

4. Görüntüde birden fazla hareketli araç olması

gibi bu olumsuz durumlar karşısında yöntemin başarısı incelenmiştir. Görsel Takip Ayrıştırma yönteminin çerçeve içindeki kısmın piksel özelliklerini analiz ederek takibi gerçekleştirmesi ve hedef nesneyi muhtemel komşu pikseller arasında araması dolayısıyla aracın boyut ve şeklinin değişmesi ve görüntüde birden fazla hareketli araç olması durumlarıyla başa çıkabildiği görülmüştür. Aracın bir engel tarafından görülememesi halinde hedefin takip edilmesi sorunu Gauss Karışımı yöntemi ile hareketli ve sabit nesneler daha belirgin hale getirilerek çözülmüştür. Ayrıca Gauss Karışımı yönteminin karmaşık durumlar için sahnedeki kademeli ve hızlı değişimlere karşı uyum sağlayabilme özelliği sayesinde arka planın hareketli nesne içermesi(kameranın kayıt sırasında oynaması, rüzgarlı havada sallanan yapraklar, dalgalanan bayrak, vb.) durumundaki takip probleminin üstesinden gelinmiştir.

Daha önce bu alanda yapılan çalışmalar ile karşılaştırıldığında tatmin edici bir sonuç ortaya çıkmaktadır. Buna rağmen yöntemin dezavantajları da bulunmaktadır. Yöntemin dezavantajlar; gerçek zamanlı çalışmaya uygun olmaması, hareketli cismin kullanıcı tarafından takip edilmeden önce işaretlenmesi ve birden fazla aracı aynı anda takip edememesi olarak sıralanabilir. Bu eksikliklerin giderilmesi için alternatif algoritmalar geliştirildiği takdirde çok daha başarılı bir araç tespiti ve takibi uygulaması olacağı açıktır.

KAYNAKLAR

Akkuş, H. E., (2009), "Following a Moving Object in a Bounded Area", Yüksek Lisans Tezi, Çankaya Üniversitesi, Ankara.

Akman, O., (2007), "Multi-Camera Video Surveillance, Detection, Occlusion Handling, Tracking and Event Recognition", Yüksek Lisans Tezi, ODTÜ, Ankara.

Ali, S. S. ve Zafar, M. F., (2009), "A Robust Adaptive Method for Detection and Tracking of Moving Objects", Department of Electronics Engineering, International Islamic University, Islamabad, 2009 International Conference on Emerging Technologies.

Amer, A., (2003), "Voting-based simultaneous tracking of multiple video objects", In Proc. SPIE Int. Symposium on Electronic Imaging, pages 500–511, Santa Clara.

Asano, T., Igarashi, Y., Nagamochi, H., Miyano, S. ve Suri, S., (1996), "7th International Symposium ISAAC 96", Osaka.

Aspremont, A., Ghaoui, L. E., Jordan, M. ve Lanckriet, G., (2007), "A direct formulation for sparse PCA using semidefinite programming", SIAM Review, 49(3).

Bal, A. ve Alam, M., (2005), "Automatic target tracking in FLIR image sequences using intensity variation function and template modeling", IEEE Transactions on Instrumentation and Measurement, 54:1846-1852.

Barrera, P., Canas, J. M. ve Matellan, V., (2005), "Visual Object Tracking in 3D with Color Based Particle Filter", International Journal of Information Technology, 2(1).

Boykov, Y. ve Kolmogorov, V., (2004), "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision", IEEE Transactions on PAMI, 26(9):1124-1137.

Bradski, G. R., (1998), "Real Time Face and Object Tracking as a Component of a Perceptual User Interface", Proceedings of the 4th IEEE Workshop on Applications of Computer Vision, 214.

Cavallaro, A., Steiger, O. ve Ebrahimi, T., (2005), "Tracking video objects in cluttered background", Circuits and Systems for Video Technology, IEEE Transactions on, 15(4):575–584.

Comaniciu, D., Ramesh, V. ve Meer, P., (2000), "Real-Time Tracking of Non-Rigid Objects Using Mean Shift", Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2:142-149.

Comaniciu, D. ve Meer, P., (2002), "Mean shift: A robust approach toward feature space analysis", IEEE Trans. Patt. Analy. Mach. Intell.

Comaniciu, D., Ramesh, V. ve Meer, P., (2003), "Kernel-Based Object Tracking", IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(5):564-575.

Corander, J., Ekdahl, M. ve Koski, T., (2008), "Parallell interacting MCMC for learning of topologies of graphical models", Data Min. Knowl. Discov., 17(3).

Dawoud, A., Alam, M., Bal, A. ve Loo, C., (2006), "Target tracking in infrared imagery using weighted composite reference function based decision fusion", IEEE Transactions on Image Processing, 15:404-410.

Dedeoğlu, Y., (2004), "Moving Object Detection, Tracking and Classification For Smart Video Surveillance", Bilkent Üniversitesi, Ankara.

Ergezer, H., (2007), "Visual Detection and Tracking of Moving Objects", Yüksek Lisans Tezi, ODTÜ, Ankara.

Gibbons, A., (1985), "Algorithmic Graph Theory", Cambridge University Press.

Gül, G. G., (2009), "Anlık Görüntüden Nesne Çıkarımı ve Otomatik Nesne Takibi Uygulaması", Yüksek Lisans Tezi, TOBB Ekonomi ve Teknoloji Üniversitesi, Ankara.

Haritaoglu, I., Harwood, D. ve Davis, L. S., (2000), "W4: Real - Time Surveillance of People and Their Activities", IEEE Trans. on Patt. Anal. and Machine Intell, 22(8):809–830.

Ju, S., Black, M. ve Yaccob, Y., (1996), "Cardboard people: a parameterized model of articulated image motion", In Proc. of the IEEE International Conference on Automatic Face and Gesture Recognition, 38–44.

Khan, Z., Balch, T. ve Dellaert. F., (2005), "MCMC-based particle filtering for tracking a variable number of interacting targets", PAMI, 27(11):1805–1918.

Kim, K. K., Cho, S. H., Kim, H. J. ve Lee, J. Y., (2005), "Detecting and Tracking Moving Object Using an Active Camera", Digital Object Identifier 10.1109/ICACT,817-820.

Kohli, P., Torr ve P. H. S., (2005), "Efficiently Solving Dynamic Markov Random Fields using Graph Cuts", Department of Computing, Oxford Brookes University.

Kwon, J. ve Lee, K. M., (2010), "Visual Tracking Decomposition", Department of EECS, ASRI, Seoul National University, Seoul, 151-742.

Leibe, B., Schindler, K., Cornelis, N. ve Van Gool, L., (2008), "Coupled object detection and tracking from static cameras and moving vehicles", PAMI, 30(10):1683–1698.

Ling, H. ve Okada, K., (2006), "Diffusion distance for histogram comparison", CVPR.

Mahalakshmi, M., (2010), "Real Time Vision Based Object tracking using CAMSHIFT Algorithm with enhanced Color Image Segmentation", Embedded System Technologies, College of Engineering, Anna University, Guindy.

Okada, R., Shirai, Y. ve Miura, J., (1996), "Object Tracking Based on Optical Flow and Depth", Proceedings of the 1996 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems.

Örten, B.B., (2005), "Moving Object Identification and Event Recognition in Video Surveillance Systems", Yüksek Lisans Tezi, ODTÜ, Ankara.

Park, S. Y., Park, C. J. ve Lee, I., (2005), "Moving Object Removal and Background Completion in a Video Sequence", Image and Vision Computing, New Zealand.

Rosales, R. ve Sclaroff, S., (1998), "Improved tracking of multiple humans with trajectory prediction and occlusion modeling", In Proc. of IEEE CVPR Workshop on the Interpretation of Visual Motion, Santa Barbara.

Salvador, E., Cavallaro, A. ve Ebrahimi, T., (2001), "Shadow Identification and Classification Using Invariant Color Model", ICASSP 2001, 7-11.

Samet, H. ve Tamminen, M., (1988), "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees", TIEEE Trans. Pattern Anal. Mach. Intell.

Srinivasan M. V., (1990), "Generalized gradient schemes for the measurement of two dimensional image motion", Biolgical Cybernetics, 63:421-431.

Stauffer, C. ve Grimson, W., (1999), "Adaptive background mixture models for realtime tracking", In Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 246-252.

Stauffer, C. ve Grimson, W., (2000), "Learning patterns of activity using real-time tracking", IEEE Pattern Recognition and Machine Intelligence, 22(8):747–757.

Su, Y., Qian, R. ve Ji, Z., (2009), "Surveillance Video Sequence Segmentation Based on Moving Object Detection", School of Electronic and Information Engineering Tianjin University, Tianjin.

Wren, C. R., Azarbayejani, A., Darrell, T. J. ve Pentland, A. P., (1997), "Pfinder: Real-time tracking of the human body", IEEE Pattern Recognition and Machine Intelligence, 19(7):780–785.

Xia, Y., Ning, S. ve Shen, H., (2010), "Moving Targets Detection Algorithm Based on Background Subtraction and Frames Subtraction", Dept. of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhen Zhou.

Yalçın, İ. K. ve Gökmen, M., (2005), "License Plate Tracking from Monocular Camera View by Condensation Algorithm", ICIC 2005, Part II, LNCS 3645, 860–869.

Yılmaz, A., Li, X. ve Shah, M., (2004), "Object Contour Tracking Using Level Sets", Conference on Computer Vision, ACCV 2004, Jaju Islands.

Yılmaz, A., Li, X. ve Shah, M., (2004), "Contour Based Object Tracking With Occlusion Handling in Video Acquired Using Mobile Cameras", IEEE Trans. Pattern Anal. Mach.Intell., 26:1531-1536.

Yılmaz, A., Javed, O. ve Shah, M., (2006), "Object Tracking A survey: ACM computing surveys", 38(4).

Yılmaz, M., (2008), "Multiple Target Tracking Using Multiple Cameras", Yüksek Lisans Tezi, ODTÜ, Ankara

EKLER

Ek 1 sGMM.cpp

```
// GMM.cpp: implementation of the GMM class.
#include "stdafx.h"
#include "sGM.h"
#include "sGMM.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

// Construction/Destruction
const unsigned int TH = 10000, TM = 10000;
const int TT=10000;
const float T = (float)0.7;

sGMM::sGMM()
// : T(0.5)
{
	m_ptr = NULL;
}

void sGMM::Init(BYTE num, BYTE size, float *mean)
{
	m_num = num;
	m_ptr = new sGM*[num];
	ASSERT(m_ptr != NULL);
	if(m_ptr != NULL)
	{
		m_ptr[0] = new sGM;
		ASSERT(m_ptr[0] != NULL);
		m_ptr[0]->Init(size, mean);
		for(int i=0; i<size; i++)
		{
			mean[i] = -100;
		}
		for(int i=1; i<num; i++)
		{
			m_ptr[i] = new sGM;
			ASSERT(m_ptr[i] != NULL);
			m_ptr[i]->Init(size, mean);
		}
	}
}
sGMM::~sGMM()
{
	if(m_ptr != NULL)
	{
		for(int i=0; i<m_num; i++)
```

```
		{
			delete m_ptr[i];
		}
		delete []m_ptr;
	}
}

BOOL sGMM::Process(float *value)
{
	int i, match = -1;
	float dist, minDist = 1000000.0;
	for(i=0; i<m_num; i++)
	{
		if(m_ptr[i]->IsValid(value, dist) && dist < minDist)
		{
			minDist = dist;
			match = i;
		}
	}
	if(match == -1)
	{
		dist = (float)(m_ptr[m_num-1]->m_weight - 0.1) / (m_num - 1);
		m_ptr[m_num-1]->Init(value, (float)0.1);
		m_ptr[m_num-1]->match=true;
		for(i=0; i<m_num-1; i++)
		{
			m_ptr[i]->m_weight += dist;
		}
		return true;
	}
	else
	{
		//
		for(i=0; i<m_num; i++)
		{
			if(i == match)
			{
				m_ptr[i]->Update(value, minDist);
				m_ptr[i]->match=true;
			}
			else
			{
				m_ptr[i]->Update();
			}
		}
		// weight/variance
		sGM *temp;
		for(i=match; i>0; i--)
		{
			if(m_ptr[i]->m_weight/m_ptr[i]->m_variance > m_ptr[i-1]-
>m_weight/m_ptr[i-1]->m_variance)
```

```
		{
			temp = m_ptr[i];
			m_ptr[i] = m_ptr[i-1];
			m_ptr[i-1] = temp;
			match --;
		}
		else
		{
			break;
		}
	}
	//
	float wsum=0.0;
	for(i=0; i<m_num; i++)
	{
		wsum += m_ptr[i]->m_weight;
		if(!m_ptr[i]->m_background)//false->true
		{
			if(i == match)
			{
				if(m_ptr[i]->m_time == 0)
				{
					m_ptr[i]->m_time =GetTickCount();
				}
				else if(GetTickCount() - m_ptr[i]->m_time > TH)
				{
					m_ptr[i]->m_background = true;
					m_ptr[i]->m_time &= 0;
				}
			}
		}
		else//true->true
		{
			m_ptr[i]->m_time &= 0;
		}
		if(wsum > T)
		{
			break;
		}
	}
	for(i++; i<m_num; i++)
	{
		if(m_ptr[i]->m_background)//true->false
		{
			if(i != match)
			{
				if(m_ptr[i]->m_time == 0)
				{
					m_ptr[i]->m_time =GetTickCount();
				}
				else if(GetTickCount()- m_ptr[i]->m_time > TM)
```

```
                                        {
                                                m_ptr[i]->m_background = false;
                                                m_ptr[i]->m_time &= 0;
                                        }
                                }
                        }
                        else//false->false
                        {
                                m_ptr[i]->m_time &= 0;
                        }
                }
                return !m_ptr[match]->m_background;
        }
}
BOOL sGMM::Process(float *value,int frameNUM)
{
        int i, match = -1;
        float dist, minDist = 1000000.0;
        for(i=0; i<m_num; i++)
        {
                if(m_ptr[i]->IsValid(value, dist) && dist < minDist)
                {
                        minDist = dist;
                        match = i;
                }
        }
        if(match == -1)
        {
                dist = (float)(m_ptr[m_num-1]->m_weight - 0.1) / (m_num - 1);
                m_ptr[m_num-1]->Init(value, (float)0.1);
                m_ptr[m_num-1]->match=true;
           m_ptr[m_num-1]->m_background=true;
                m_ptr[m_num-1]->m_time=frameNUM;
                for(i=0; i<m_num-1; i++)
                {
                        m_ptr[i]->m_weight += dist;
                }
                return true;
        }
        else
        {
                //
                for(i=0; i<m_num; i++)
                {
                        if(i == match)
                        {
                                if(!m_ptr[i]->m_background)
                                {
                                        if(frameNUM-m_ptr[i]->m_time==0)
                                        {
                                                m_ptr[i]->match=true;
```

```
                                m_ptr[i]->m_time=frameNUM;

                        }
                        else if(frameNUM-m_ptr[i]->m_time>TT)
                        {
                                m_ptr[i]->Update(value, minDist);
                                m_ptr[i]->match=true;
                                m_ptr[i]->m_time=0;
                        }
                }
                else
                {
                        m_ptr[i]->Update(value, minDist);
                        m_ptr[i]->match=true;
                        m_ptr[i]->m_background=false;
                        m_ptr[i]->m_time=0;
                }
        }
        else
        {
                if(!m_ptr[i]->m_background)
                {
                        if(frameNUM-m_ptr[i]->m_time>TT)
                        {
                                m_ptr[i]->Update();
                        }
                }
                else
                {
                        m_ptr[i]->Update();
                        m_ptr[i]->m_background=false;
                        m_ptr[i]->m_time=0;
                }

        }
}
// weight/variance
sGM *temp;
for(i=match; i>0; i--)
{
        if(m_ptr[i]->m_weight/m_ptr[i]->m_variance > m_ptr[i-1]-
>m_weight/m_ptr[i-1]->m_variance)
        {
                temp = m_ptr[i];
                m_ptr[i] = m_ptr[i-1];
                m_ptr[i-1] = temp;
                match --;
        }
        else
        {
                break;
```

```
                        }
                }
                //
                float wsum=0.0;
                for(i=0; i<m_num; i++)
                {
                        wsum += m_ptr[i]->m_weight;
                                m_ptr[i]->m_time &= 0;
                                m_ptr[i]->m_background = true;
                        //
                        if(wsum > T)
                        {
                                break;
                        }
                }
                for(i++; i<m_num; i++)
                {
                        if(m_ptr[i]->m_background)//true->false
                        {
                                m_ptr[i]->m_time &= 0;

                        }
                        else//false->false
                        {
                                if(frameNUM - m_ptr[i]->m_time > TT)//TH)
                                {
                                        m_ptr[i]->m_background = true;
                                        m_ptr[i]->m_time &= 0;
                                }
                        }
                }
                return !m_ptr[match]->m_background;
        }
}

BOOL sGMM::Process_No_Update(float *value)
{
        int i, match = -1;
        float dist, minDist = 1000000.0;
        for(i=0; i<m_num; i++)
        {
                if(m_ptr[i]->IsValid(value, dist) && dist < minDist)
                {
                        minDist = dist;
                        match = i;
                }
        }
        if(match == -1)
        {
                return true;
        }
```

```
	else
	{
		return !m_ptr[match]->m_background;
	}
}

void sGMM::Init_Process(float *value)
{
	int i, match = -1;
	float dist, minDist = 1000000.0;
	for(i=0; i<m_num; i++)
	{
		if(m_ptr[i]->IsValid(value, dist) && dist < minDist)
		{
			minDist = dist;
			match = i;
		}
	}
	if(match == -1)
	{
		m_ptr[m_num-1]->Init(value, 1.0);
		return;
	}
	else
	{
		//
		m_ptr[match]->Init_Update(value, minDist);
		// weight/variance
		sGM *temp;
		for(i=match; i>0; i--)
		{
			if(m_ptr[i]->m_weight/m_ptr[i]->m_variance > m_ptr[i-1]-
>m_weight/m_ptr[i-1]->m_variance)
			{
				temp = m_ptr[i];
				m_ptr[i] = m_ptr[i-1];
				m_ptr[i-1] = temp;
			}
			else
			{
				break;
			}
		}
	}
}

void sGMM::Norm(void)
{
	int i;
	float sum = 0.0;
	float sum_fore=0.0;
```

```
for(i=0; i<m_num; i++)
{
	sum += m_ptr[i]->m_weight;
	sum_fore+=m_ptr[i]->m_wfore;
}
for(i=0; i<m_num; i++)
{
	m_ptr[i]->m_weight = m_ptr[i]->m_weight / sum;
	m_ptr[i]->m_wfore = m_ptr[i]->m_wfore / sum_fore;
}
//
float wsum=0.0;
for(i=0; i<m_num; i++)
{
	wsum += m_ptr[i]->m_weight;
	m_ptr[i]->m_background = true;
	if(wsum > T)
	{
		break;
	}
}
for(i++; i<m_num; i++)
{
	m_ptr[i]->m_background = false;	}}
```

Ek 2 Observation.cpp

```
#include "../include/defs.h"
#include "../include/utils.h"
#include "../include/observation.h"
#include "../include/dd_head.h"

////////////////////////////////////////////////////////////////////////////////////////
//  Converts a BGR image to HSV color space
//  @param bgr image to be converted
//  @return Returns bgr converted to a 3-channel, 32-bit HSV image
//  with S and V values in the range [0,1] and H value in the range [0,360]
////////////////////////////////////////////////////////////////////////////////////////
IplImage* bgr2hsv( IplImage* bgr )
{
        IplImage* bgr_flip, * bgr32f, * hsv;

        bgr32f = cvCreateImage( cvGetSize(bgr), IPL_DEPTH_32F, 3 );
        hsv             = cvCreateImage( cvGetSize(bgr), IPL_DEPTH_32F, 3 );

        bgr_flip = (IplImage*)cvClone(bgr);
        cvConvertImage(bgr,bgr_flip,1);

        cvConvertScale( bgr_flip, bgr32f, 1.0 / 255.0, 0 );
        cvCvtColor( bgr32f, hsv, CV_BGR2HSV );

        cvReleaseImage( &bgr_flip );
        cvReleaseImage( &bgr32f );

        return hsv;
}

////////////////////////////////////////////////////////////////////////////////////////
//  Calculates the histogram bin into which an HSV entry falls
//  @param h Hue
//  @param s Saturation
//  @param v Value
//  @return Returns the bin index corresponding to the HSV color defined by
//    \a h, \a s, and \a v.
////////////////////////////////////////////////////////////////////////////////////////
int histo_bin( float h, float s, float v )
{
        int hd, sd, vd;

        /* if S or V is less than its threshold, return a "colorless" bin */
        vd = MIN( (int)(v * NV / V_MAX), NV-1 );
        if( s < S_THRESH  ||  v < V_THRESH )
                return NH * NS + vd;

        /* otherwise determine "colorful" bin */
        hd = MIN( (int)(h * NH / H_MAX), NH-1 );
```

```
        sd = MIN( (int)(s * NS / S_MAX), NS-1 );

        return sd * NH + hd;
}

////////////////////////////////////////////////////////////////////////////////////////
//
//  Calculates a cumulative histogram as defined above for a given array of images
//  @param img an array of images over which to compute a cumulative histogram;
    each must have been converted to HSV colorspace using bgr2hsv()
//  @param n the number of images in imgs
//  @return Returns an un-normalized HSV histogram for \a imgs
////////////////////////////////////////////////////////////////////////////////////////
histogram* calc_histogram( IplImage** imgs, int n )
{
        IplImage* img;
        IplImage* h, * s, * v;

        histogram* histo;

        float* hist;
        int i, r, c, bin;

        histo = (histogram*)malloc( sizeof(histogram) );
        histo->n = NH*NS + NV;
        hist = histo->histo;
        memset( hist, 0, histo->n * sizeof(float) );

        for( i = 0; i < n; i++ )
        {
                /* extract individual HSV planes from image */
                img = imgs[i];

                h = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
                s = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
                v = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
                cvCvtPixToPlane( img, h, s, v, NULL );

                /* increment appropriate histogram bin for each pixel */
                for( r = 0; r < img->height; r++ )
                        for( c = 0; c < img->width; c++ )
                        {
                                bin = histo_bin( pixval32f( h, r, c ),
                                                 pixval32f( s, r, c ),
                                                 pixval32f( v, r, c ) );
                                hist[bin] += 1;
                        }

                cvReleaseImage( &h );
                cvReleaseImage( &s );
                cvReleaseImage( &v );
```

```
        }

        return histo;
}

////////////////////////////////////////////////////////////////////////////////
//  Normalizes a histogram so all bins sum to 1.0
//  @param histo a histogram
////////////////////////////////////////////////////////////////////////////////
void normalize_histogram( histogram* histo )
{
        float* hist;
        float sum = 0, inv_sum;

        int i, n;

        hist    = histo->histo;
        n               = histo->n;

        /* compute sum of all bins and multiply each bin by the sum's inverse */
        for( i = 0; i < n; i++ )
                sum += hist[i];

        inv_sum = (float)(1.0 / sum);

        for( i = 0; i < n; i++ )
                hist[i] *= inv_sum;
}

////////////////////////////////////////////////////////////////////////////////
//
//  Computes squared distance metric
//      based on the Battacharyya similarity coefficient between histograms.
//  @param h1 first  histogram; should be normalized
//  @param h2 second histogram; should be normalized
//  @return Returns a squared distance
//  based on the Battacharyya similarity coefficient between \a h1 and \a h2
////////////////////////////////////////////////////////////////////////////////
float histo_dist_sq( histogram* h1, histogram* h2 )
{
        float* hist1, * hist2;
        float sum = 0;
        int i, n;

        n               = h1->n;
        hist1   = h1->histo;
        hist2   = h2->histo;

        /*
        According the the Battacharyya similarity coefficient,
```

```
        D = sqrt{ 1 - sum_1^n{ sqrt{ h_1(i) * h_2(i) } } }
        */

        for( i = 0; i < n; i++ )
                sum += (float)(sqrt( hist1[i]*hist2[i] ));

        return (float)(1.0 - sum);
}

////////////////////////////////////////////////////////////////////////////////////////
//
//  Computes the likelihood of there being a player at a given location in an image
//  @param img image that has been converted to HSV color space using bgr2hsv()
//  @param r row location of center of window around which to compute likelihood
//  @param c col location of center of window around which to compute likelihood
//  @param w width of region over which to compute likelihood
//  @param h height of region over which to compute likelihood
//  @param ref_histo reference histogram for a player; must have been normalized with
normalize_histogram()
//  @return Returns the likelihood of there being a player at location (\a r, \a c) in \a img
////////////////////////////////////////////////////////////////////////////////////////

float likelihood_color( IplImage* img, int r, int c, int w, int h, histogram** ref_histo )
{
        IplImage* tmp;
        histogram* histo;
        float d_sq = 0;

        for ( int i = 0; i < 2; i++ )
        {
                /* extract region around (r,c) and compute and normalize its histogram */
                cvSetImageROI( img, cvRect( c - w / 2, r -  ( h / 2 ) * ( 1 - i ), w, h / 2 ) );
                tmp = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 3 );
                cvCopy( img, tmp, NULL );
                cvResetImageROI( img );

                /* compute histogram */
                histo = calc_histogram( &tmp, 1 );
                cvReleaseImage( &tmp );
                normalize_histogram( histo );

                /* compute likelihood as e^{\lambda D^2(h, h^*)} */
                d_sq += histo_dist_sq( histo, ref_histo[i] );
                free( histo );
        }

        return (float)(exp( -LAMBDA * 4 * d_sq / 2 ));
}

float likelihood_decomposed( IplImage** features, int r, int c, int w, int h, model&
observation, int chain_index, int nFeature, FILE* info )
```

```
{
	int i, j, k;
	float d_sq = 0;

	// set ROI
	IplImage* f[MAX_FEATURES], *tmp;
	for ( i = 0; i < nFeature; i++ )
	{
		cvSetImageROI( features[i], cvRect( c - w / 2, r -  h / 2, w, h ) );
		tmp = cvCreateImage( cvGetSize(features[i]), IPL_DEPTH_32F, 1 );
		cvCopy( features[i], tmp, NULL );
		cvResetImageROI( features[i] );

		f[i] = cvCreateImage( cvSize(MW,MH), IPL_DEPTH_32F, 1 );
		cvResize(tmp, f[i] );

		cvReleaseImage(&tmp);
	}

	// matching
	int width		= MW;
	int height		= MH;

	double val;
	float idx_w;
	int idx_f, idx_i, count;

	int num[MAX_FEATURES];
	double template_flat[MAX_FEATURES][MW*MH], temp_template_flat[MW*MH];
	double test_flat[MAX_FEATURES][MW*MH], temp_test_flat[MW*MH];

	for ( i = 0; i < nFeature; i++ )
	{
		num[i] = 0;

		for ( j = 0; j < MW*MH; j++ )
		{
			template_flat[i][j] = 0;
			test_flat[i][j]		= 0;
		}
	}

	for ( i = 0; i < observation.chains[chain_index].num; i++ )
	{
		idx_f = observation.chains[chain_index].pc_f[i];
		idx_i = observation.chains[chain_index].pc_i[i];
		idx_w = observation.chains[chain_index].pc_w[i];

		// template features
		count = 0;
		for ( j = 0; j < height; j++ )
```

```
                    for ( k = 0; k < width; k++ )
                    {
                        val = pixval32f(observation.feature[idx_f][idx_i], j, k);
                        temp_template_flat[count++] = val;
                    }

                // test features
                count = 0;
                for ( j = 0; j < height; j++ )
                    for ( k = 0; k < width; k++ )
                    {
                        val = pixval32f(f[idx_f], j, k);
                        temp_test_flat[count++] = val;
                    }

                num[idx_f]++;
                for ( j = 0; j < width*height; j++ )
                {
                    template_flat[idx_f][j] += temp_template_flat[j];
                    test_flat[idx_f][j]             += temp_test_flat[j];
                }
            }

            count = 0;
            for ( i = 0; i < nFeature; i++ )
            {
                if ( num[i] > 0 )
                {
                    count++;

                    for ( j = 0; j < width*height; j++ )
                    {
                        temp_template_flat[j] = template_flat[i][j] / float(num[i]);
                        temp_test_flat[j]      = test_flat[i][j] / float(num[i]);
                    }

                    dd_distdd;
                    float temp = dd.dd2D(temp_template_flat, temp_test_flat, width,
height) / float( width * height );
                    d_sq +=  temp;

                    if ( info )
                        fprintf(info, "%d (%f) ",i,temp);
                }
            }

            d_sq =  d_sq / float(count);

            for ( i = 0; i < nFeature; i++ )
                cvReleaseImage(&f[i]);
```

```
        return (float)(exp( -LAMBDA * d_sq ));
}

void extract_feature(IplImage* image, IplImage** f, int update, int nFeature)
{
        int j, k;
        int count;
        double val;

        int width               = image->width;
        int height              = image->height;
        int linear_size = height * width;

        IplImage*h, *s, *v, *e;

        // 1. feature extraction
        h = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
        s = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
        v = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
        e = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );

        // (1) hue, (2) saturation, (3) value
        cvCvtPixToPlane( image, h, s, v, NULL );

        // normalization
        cvConvertScale( h, h, 1.0f/360.0f, 0 );

        // (4) edge
        double* i_linear                = (double*)malloc(linear_size * sizeof(double)); // Input
        double* e_linear                = (double*)malloc(linear_size * sizeof(double)); //
output
        double* c_linear                = (double*)malloc(linear_size * sizeof(double)); //
output

        try
        {
                // input
                count = 0;
                for ( j = 0; j < width; j++ )
                        for ( k = 0; k < height; k++ )
                        {
                                val =  pixval32f( v, k, j );
                                i_linear[count++] = val;
                        }

                mwArray im(height, width, mxDOUBLE_CLASS, mxREAL);
                im.SetData(i_linear, linear_size);

                // output
                mwArray M;                      // Maximum moment of phase congruency
covariance. This is used as a indicator of edge strength.
```

```
                mwArray m;                  // Minimum moment of phase congruency
covariance. This is used as a indicator of corner strength.
                mwArray or0;        // Orientation image in integer degrees 0-180, positive
anticlockwise. 0 corresponds to a vertical edge, 90 is horizontal.
                mwArray featType;   // *Not correctly implemented at this stage*
                mwArray PC;                 // Cell array of phase congruency images (values
between 0 and 1)  for each orientation
                mwArray EO;                 //  A 2D cell array of complex valued convolution
results

                // compute edge and corner phase congruency in an image
                phasecong2(6, M, m, or0, featType, PC, EO, im);

                M.GetData(e_linear, linear_size);
        }
        catch (const mwException& e)
        {
                AfxMessageBox(e.what());
                return;
        }

        // edge strength image
        count = 0;
        for ( j = 0; j < width; j++ )
                for ( k = 0; k < height; k++ )
                {
                        setpix32f(e, k, j, e_linear[count++]);
                        val = pixval32f(e, k, j);
                }

        cvDilate(v,v);
        cvDilate(e,e);
        cvDilate(e,e);

        free(i_linear);
        free(e_linear);
        free(c_linear);

        // 2. description
        if ( nFeature == 2 )
        {
                f[0]    = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
                f[1]    = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );

                cvCopy( v, f[0] );
                cvCopy( e, f[1] );
        }
        else
        {
                f[0]    = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
                f[1]    = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
```

```
            f[2]    = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );
            f[3]    = cvCreateImage( cvSize(width,height), IPL_DEPTH_32F, 1 );

            cvCopy( h, f[0] );
            cvCopy( s, f[1] );
            cvCopy( v, f[2] );
            cvCopy( e, f[3] );
        }

        cvReleaseImage(&h);
        cvReleaseImage(&s);
        cvReleaseImage(&v);
        cvReleaseImage(&e);
}

void make_observation_model(IplImage** features, int r, int c, int w, int h,int queue, int
update, int nFeature, model& observation)
{
        int i, j, k, l;
        int count;
        double val;

        // set ROI
        IplImage* f[MAX_FEATURES], *tmp;
        for ( i = 0; i < nFeature; i++ )
        {
            cvSetImageROI( features[i], cvRect( c - w / 2, r -  h / 2, w, h ) );
            tmp = cvCreateImage( cvGetSize(features[i]), IPL_DEPTH_32F, 1 );
            cvCopy( features[i], tmp, NULL );
            cvResetImageROI( features[i] );

            f[i] = cvCreateImage( cvSize(MW,MH), IPL_DEPTH_32F, 1 );
            cvResize(tmp, f[i] );

            cvReleaseImage(&tmp);
        }

        // 3. SPCA
        int width               = MW;
        int height              = MH;
        int linear_size = height * width;

        for ( l = 0; l < nFeature; l++ )
        {
            if ( observation.feature[l][queue] )
                cvReleaseImage(&observation.feature[l][queue]);
            observation.feature[l][queue] = f[l];
        }

        if ( update == 1 )  // initial
        {
```

```
                for ( l = 0; l < MAX_CHAINS; l++ )
                {
                        int type = rand() % nFeature;

                        observation.chains[l].pc_f[0] = type; // choose type of feature
                        observation.chains[l].pc_i[0] = 0;     // choose index in the feature
                        observation.chains[l].pc_w[0]         = 1.0f;

                        observation.chains[l].num             = 1;
                }
        }
        else if ( update % MAX_UPDATE == 0 ) // update
        {
                int max_index;
                if ( update < MAX_OSBS )
                        max_index =  queue + 1;
                else
                        max_index = MAX_OSBS;

                // making data matrix
                CvMat* data   = cvCreateMat(width * height, max_index * nFeature,
CV_64FC1);

                int row, col;
                for ( i = 0; i < max_index; i++ )
                {
                        for ( l = 0; l < nFeature; l++ )
                        {
                                for ( j = 0; j < height; j++ )
                                        for ( k = 0; k < width; k++ )
                                        {
                                                row = j * width + k;
                                                col = i + max_index * l;

                                                val = pixval32f(observation.feature[l][i], j, k);

                                                data->data.db[row * data->width + col ]      = val;
                                        }
                        }
                }

                // normalization
                CvMat* norm = cvCreateMat(data->height, 1, CV_64FC1);
                for ( j = 0; j < data->width; j++ )
                {
                        for ( i = 0; i < data->height; i++ )
                                norm->data.db[i] = data->data.db[i * data->width + j];

                        float val = cvNorm(norm);
                        for ( i = 0; i < data->height; i++ )
```

```
                                data->data.db[i * data->width + j] = data->data.db[i * data->width + j] / val;
                        }
                        cvReleaseMat(&norm);

                        // make covariance matrix
                        CvMat* S = cvCreateMat(data->width, data->width, CV_64FC1);
                        cvMulTransposed(data,S,1);

                        count = 0;
                        double* S_linear = (double*)malloc(data->width*data->width*sizeof(double));
// Input
                        for ( i = 0; i < data->width; i++ )
                                for ( j = 0; j < data->width; j++ )
                                        S_linear[count++] = S->data.db[j * data->width + i];

                        // SPCA
                        int maxiter                     = 3000;                 // Maximum Number of Iterations
                        int info                = 100;          // Control amount of output

                        double* X_linear = (double*)malloc(data->width*data->width*sizeof(double));
                        double* x_linear = (double*)malloc(data->width*1*sizeof(double));
                        int* k_linear = (int*)malloc(1*sizeof(int));

                        try
                        {
                                // input
                                mwArray covmtx(data->width, data->width, mxDOUBLE_CLASS, mxREAL);
                                covmtx.SetData(S_linear, data->width * data->width);

                                // Penalty factor controlling sparsity (large rho => sparse sol)
                                double n_rho[1];
                                n_rho[0] = RHO;
                                mwArray rho(1, 1, mxDOUBLE_CLASS, mxREAL);
                                rho.SetData(n_rho, 1*1);

                                // Target precision
                                double n_gapchange[1];
                                n_gapchange[0] = GAP;
                                mwArray gapchange(1, 1, mxDOUBLE_CLASS, mxREAL);
                                gapchange.SetData(n_gapchange, 1*1);

                                // Maximum Number of Iterations
                                int n_maxiter[1];
                                n_maxiter[0] = maxiter;
                                mwArray maxiter(1, 1, mxINT16_CLASS, mxREAL);
                                maxiter.SetData(n_maxiter, 1*1);
```

```
                    // Control amount of output
                    int n_info[1];
                    n_info[0] = info;
                    mwArray info(1, 1, mxINT16_CLASS, mxREAL);
                    info.SetData(n_info, 1*1);

                    int n_algo[1];
                    n_algo[0] = 3;
                    mwArray algo(1, 1, mxINT16_CLASS, mxREAL);
                    algo.SetData(n_algo, 1*1);

                    // output
                    mwArray U;                  // symmetric matrix that solves the above
SDP
                    mwArray X;                  // dual variable, solves the dual SDP
                    mwArray x;                  // largest eigenvector of U
                    mwArray F;                  // Average gradient
                    mwArray k;                  // number of iterations run
                    mwArray dualitygap; // vector of duality gaps at designated iterations
                    mwArray cputime;    // vector of cumulative cpu times at designated
iterations
                    mwArray perceigs;   // vector of percentage of eigenvalues used (in
partial eigenvalue decomposition) at designated iterations

                    // run SPCA
                    DSPCA(8, U, X, x, F, k, dualitygap, cputime, perceigs, covmtx ,rho
,gapchange ,maxiter, info, algo);

                    x.GetData(x_linear, data->width * 1);
                    X.GetData(X_linear, data->width*data->width);
                    k.GetData(k_linear, 1);

                }
                catch (const mwException& e)
                {
                    AfxMessageBox(e.what());
                    return;
                }

                // output
                CvMat* X = cvCreateMat(data->width, data->width, CV_64FC1);

                count = 0;
                for ( i = 0; i < data->width; i++ )
                    for ( j = 0; j < data->width; j++ )
                        X->data.db[j * data->width + i] = X_linear[count++];

                CvMat* evects = cvCreateMat(data->width, data->width, CV_64FC1);
                CvMat* evals = cvCreateMat(data->width, 1, CV_64FC1);
                cvEigenVV( X, evects, evals);
```

```
                cvTranspose(evects, evects);

                int idx, idx_f, idx_i;
                for ( i = 0; i < MAX_CHAINS; i++ )
                {
                        idx = 0;

                        for ( j = 0; j < data->width; j++ )
                        {
                                idx_f = j / max_index;
                                idx_i = j % max_index;
                                val = fabsf(evects->data.db[j * data->width + i]);

                                if ( val > THRES || idx_i == 0 )
                                //if ( idx_f == i )
                                {
                                        observation.chains[i].pc_f[idx] = idx_f;
                                        observation.chains[i].pc_i[idx] = idx_i;

                                        observation.chains[i].pc_w[idx]       = val;

                                        idx++;
                                }
                        }

                        observation.chains[i].num = idx;
                }

                free(S_linear);
                free(x_linear);
                free(X_linear);
                free(k_linear);

                cvReleaseMat(&data);
                cvReleaseMat(&S);
        }
        else
        {
                // nothing
        }
}
```

Ek 3 Gta.cpp

```
/*
  Functions for object tracking with Wang-Landau Monte Carlo
*/

#include "stdafx.h"
#include "Track.h"


#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////
// Definitions
//////////////////////////////////////////////////////////////////

/* start frame */
#define START_FRAME 0

/* maximum number of frames for exporting */
#define MAX_FRAMES 1000

/* default base name and extension of exported frames */
#define EXPORT_BASE "\\frame\\frame_"
#define EXPORT_EXTN ".jpg"
#define EXPORT_VIDEO "\\frame\\output.avi"

//////////////////////////////////////////////////////////////////
// Structures
//////////////////////////////////////////////////////////////////

typedef struct params
{
	CvPoint		center[MAX_OBJECTS];

	int		width[MAX_OBJECTS];
	int		height[MAX_OBJECTS];

	int		ang[MAX_OBJECTS];
	int		frame[MAX_OBJECTS];

	IplImage*	objects[MAX_OBJECTS];

	char* win_name;

	IplImage* orig_img;
	IplImage* cur_img;
```

```
        int n;
} params;

//////////////////////////////////////////////////////////////////
// Function Prototypes
//////////////////////////////////////////////////////////////////
histogram** compute_ref_histos( IplImage*, CvBox2D);
histogram** compute_dyn_histos( IplImage** , IplImage**);

int export_frame( IplImage*, int, std::string folderpath );
int get_regions_from_file( char* , int**, CvBox2D** );
void get_object_image( IplImage* img, IplImage** tmp, int r, int c, int w, int h);
void get_param_from_file(CString folderpath);

int nChannel;
int nFeature;               /* number of features */
int nIter;                      /* number of samples */
float nVar;
float sMin;
float sMax;

int num_particles;          /* number of particles */

//////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////

CTrack::CTrack()
{
        show_all            = FALSE;
        export_img          = TRUE;
        export_vid          = TRUE;

        interaction         = TRUE;
}

CTrack::~CTrack()
{

}

void CTrack::RunTracking(CString filename, CString filepath, CString folderpath)
{
        // parameter setting
    get_param_from_file(folderpath);

        CvCapture* video;
        histogram** ref_histos, ** dyn_histos=0;
        IplImage* frame, * hsv_frame, * frames[MAX_FRAMES],
*cur_features[MAX_FEATURES], *prev_features[MAX_FEATURES];
```

```
        particle* particles = NULL;
        particle* new_samples, * old_samples;
        particle spca_sample;
        float ratio;

        model observation[MAX_OBJECTS];

        CvScalar* color;
        CvBox2D* regions;
        int* detectfr;

        int num_objects   = 0;
        int total_objects = 0;

        int i, j, k, l, w, h, n, o, p;

        i = 0;
        n = 0;
        p = 0;

        // open a video file
        vid_file        = (LPSTR)(LPCSTR)filepath;
        video           = cvCaptureFromFile( vid_file );

        if( ! video )
                AfxMessageBox("couldn't open video file");

        // open a info file
   total_objects = get_regions_from_file( (LPSTR)(LPCSTR)(filepath+".txt"), &detectfr,
&regions );
        color = (CvScalar*)malloc(total_objects*sizeof(CvScalar));

        // create particles
        particles       = (particle*)malloc( num_particles * sizeof( particle ) );
        old_samples = (particle*)malloc( MAX_CHAINS * sizeof( particle ) );
        new_samples = (particle*)malloc( MAX_CHAINS * sizeof( particle ) );

        float max_weight;
        int select_sample = 0;
        IplImage* ref_img[MAX_OBJECTS][2];
        IplImage* dyn_img[MAX_OBJECTS][2];
        int img_queue[MAX_OBJECTS];
        int img_update[MAX_OBJECTS];

        // save the best configuration at each frame
        FILE *info1;
   info1 = fopen((std::string(folderpath)+std::string("\\frame\\best_configuration.txt")).c_str(),
"w");

        // matlab library initialization
```

```
        if ( !mclInitializeApplication(NULL,0))
        {
                AfxMessageBox("could not initialize application properly");
                return;
        }
if (!mclmcrInitialize() )
{
   AfxMessageBox("could not initialize application properly");
   return;
}
        if ( !phaseInitialize() )
        {
                AfxMessageBox("could not initialize application properly");
                return;
        }
        if ( !DSPCAInitialize())
        {
                AfxMessageBox("could not initialize application properly");
                return;
        }

        // do tracking
        img_update[0] = 0;
bool flag = true;
        while( frame = cvQueryFrame( video ) )
        {

  if(flag)
   {
      for(int i=0; i<total_objects; i++)
      {
         regions[i].center.y = frame->height-regions[i].center.y;
      }
      flag = false;
   }

                // seek the start frame
                if ( p < START_FRAME )
                {
                        p++;
                        continue;
                }

                // covert RGB into HSV
                hsv_frame = bgr2hsv( frame );
                frames[i] = (IplImage *)cvClone( frame );

                // extract feature
                extract_feature(hsv_frame, cur_features, img_update[0], nFeature);

                w = frame->width;
```

```
                h = frame->height;

                // detect the object
                while ( n < total_objects && detectfr[n] == i )
                {
                        /* compute reference histograms and distribute particles */
                        ref_histos = compute_ref_histos( hsv_frame, regions[n]);

                        // get initial image
                        get_object_image( hsv_frame,
                                                ref_img[n],
                                                cvRound( regions[n].center.y ),
cvRound( regions[n].center.x ),
                                                cvRound( regions[n].size.width ),
cvRound( regions[n].size.height ));

                        dyn_img[n][0] = ref_img[n][0];
                        dyn_img[n][1] = ref_img[n][1];

                        // initialize image queue
                        img_update[n]           = 1;
                        img_queue[n]            = 0;

                        for( j = 0; j < nFeature; j++ )
                                for( l = 0; l < MAX_OSBS; l++ )
                                        observation[n].feature[j][l] = NULL;

                        for( j = 0; j < nFeature; j++ )
                                prev_features[j] = (IplImage *)cvClone( cur_features[j] );

                        // initialize particles
                        init_distribution( regions[n], particles, n, num_particles );

                        // display bounding box
                        color[n] = CV_RGB(255,255,255);
                        display_object( frame, regions[n], CV_RGB(255,0,0));

                        n++;
                        num_objects = n;
                }

                int accepted_num        = 0;
                float acceptance_rate = 0.0f;

                if ( num_objects > 0 )
                {
                        ////////////////////
                        /* initialize state */
                        ////////////////////

                        ////////////////
```

```
                        ////////////////
                        /* IMC tracker */
                        ////////////////
                        ////////////////

                        // make observation models
                        for( j = 0; j < num_objects; j++ )
                        {
                                float s = particles[select_sample].s[j];
                                make_observation_model(prev_features,
                                                                        int(
particles[select_sample].y[j] + 0.5f ), int( particles[select_sample].x[j] + 0.5f ),
                                                                        int(
float(particles[select_sample].width[j]) * s + 0.5f ), int(
float(particles[select_sample].height[j]) * s + 0.5f ),
                                                                        img_queue[j],
img_update[j], nFeature, observation[j]);
                        }

                        //////////////
                        /* sampling */
                        //////////////

                        // move all targets according to the second order autoregressive motion
model
                        for ( j = 0; j < MAX_CHAINS; j++ )
                        {
                                int motion_index        = j%MAX_MOTIONS;
                                new_samples[j]                  = transition(
particles[select_sample], num_objects, w, h, motion_index, nVar, sMin, sMax);
                        }

                        for( j = 0; j < num_objects; j++ )
                        {
                                for ( l = 0; l < MAX_CHAINS; l++ )
                                {
                                        int observation_index = l/MAX_MOTIONS;
                                        float s = new_samples[l].s[j];
                                        new_samples[l].w[j]   = likelihood_decomposed(
cur_features,

                int( new_samples[l].y[j] + 0.5f ), int( new_samples[l].x[j] + 0.5f  ),

                int( float(new_samples[l].width[j]) * s + 0.5f ), int(
float(new_samples[l].height[j]) * s + 0.5f ),

                observation[j], observation_index, nFeature, NULL);
                                        old_samples[l] = new_samples[l];
                                }
                        }
```

```
k = 0;
max_weight = 0;

float rnd, prob;
float factor     = 0.5f / float(nIter);
float alpha              = 1.0f;

for( j = 0; j < nIter; j++ )
{
        int m = rand() % num_objects; // randomly choose one target

        for ( l = 0; l < MAX_CHAINS; l++ )
        {
                int motion_index              = l%MAX_MOTIONS;
                int observation_index = l/MAX_MOTIONS;

                // proposal step
                new_samples[l] =
sample_from_Proposal(old_samples[l], m, w, h, motion_index, nVar, sMin, sMax);

                // acceptance step
                float s = new_samples[l].s[m];
                new_samples[l].w[m] = likelihood_decomposed(
cur_features,

        int( new_samples[l].y[m] + 0.5f ), int( new_samples[l].x[m] +
0.5f  ),

        int( float(new_samples[l].width[m]) * s + 0.5f ), int(
float(new_samples[l].height[m]) * s + 0.5f ),

        observation[m], observation_index, nFeature, NULL);

                ratio = acceptance_ratio(&new_samples[l],
&old_samples[l], m);

                // update a sample
                if ( ratio >= 1 )
                {
                        update_sample(&old_samples[l],
&new_samples[l], m);
                        accepted_num++;
                }
                else
                {
                        if (  float(rand()) / float(RAND_MAX) < ratio )
                        {
                                update_sample(&old_samples[l],
&new_samples[l], m);
                                accepted_num++;
                        }
```

```
                }

                // save the particle and do MAP estimation
                if ( k < num_particles )
                {
                        particles[k++] = old_samples[l];

                        float weight = 0;
                        for ( o = 0; o < num_objects; o++ )
                                weight += particles[k-1].w[o];

                        if ( weight > max_weight )
                        {
                                select_sample = k-1;
                                max_weight = weight;
                        }
                }
                else
                {
                        l = MAX_CHAINS;
                        j = nIter;
                }

        }

        if ( interaction )
        {
                // interaction
                rnd = float(rand()) / float(RAND_MAX);
                if ( rnd <= alpha )
                {
                        // decrease an alpha value
                        alpha = alpha - factor;

                        for ( l = 0; l < MAX_CHAINS; l++ )
                        {
                                float temp = 0.0f;
                                for ( o = 0; o < MAX_CHAINS; o++ )
                                        temp += old_samples[o].w[m];

                                prob = 0.0f;
                                rnd = float(rand()) / float(RAND_MAX);

                                for ( o = 0; o < MAX_CHAINS; o++ )
                                {
                                        prob += old_samples[o].w[m] /
temp;

                                        if ( rnd < prob )
                                        {
                                                new_samples[l] =
old_samples[o];
```

```
                                                o = MAX_CHAINS;
                                            }
                                        }
                                    }

                                    for ( l = 0; l < MAX_CHAINS; l++ )
                                        old_samples[l] = new_samples[l];
                                }
                            }
                        }
                        spca_sample = particles[select_sample];
                }

                // get dynamic image
                for ( int l = 0; l < num_objects; l++ )
                {
                        img_update[l]++;
                        img_queue[l]++;
                        if ( img_queue[l] == MAX_OSBS )
                                img_queue[l] = 1;

                        get_object_image( hsv_frame,
                                                dyn_img[l],
                                                particles[select_sample].y[l],
                                                particles[select_sample].x[l] ,
                                                particles[select_sample].width[l] *
particles[select_sample].s[l],
                                                particles[select_sample].height[l]*
particles[select_sample].s[l]);

                        if ( dyn_histos )
                                free(dyn_histos);
                        dyn_histos = compute_dyn_histos(dyn_img[l], ref_img[l]);
                }

                for ( j = 0; j < nFeature; j++ )
                        cvReleaseImage(&prev_features[j]);
                for( j = 0; j < nFeature; j++ )
                                prev_features[j] = (IplImage *)cvClone( cur_features[j] );
                for ( j = 0; j < nFeature; j++ )
                        cvReleaseImage(&cur_features[j]);

                /* display most likely particle */
                display_particle( frames[i], spca_sample, num_objects, color, info1, 0 );

                /* display all particle */
                if( show_all )
                        for( j = 1; j < num_particles; j++ )
                                display_particle( frames[i], particles[j], num_objects, color,
info1, 0 );
```

```
                cvNamedWindow( "Video", 1 );
                cvShowImage( "Video", frames[i] );

                cvWaitKey( 5 );

                cvReleaseImage( &hsv_frame );

                i++;

                int c = cvWaitKey(10);

                if ( i == MAX_FRAMES || c == 27)
                        break;
        }

        fclose(info1);

        if ( particles )
                free( particles );
        if ( old_samples )
                free( old_samples );

        /* export video frames, if export requested */
        CvVideoWriter* videoWriter;
        double dblFrameRate = cvGetCaptureProperty(video, CV_CAP_PROP_FPS);
        double codec = cvGetCaptureProperty(video, CV_CAP_PROP_FOURCC);

        cvReleaseCapture( &video );

        if( export_vid )
      videoWriter       =
cvCreateVideoWriter((std::string(folderpath)+std::string(EXPORT_VIDEO)).c_str(),
(int)codec, 20, cvSize(frames[0]->width, frames[0]->height),1);

        /* export image frames, if export requested */
        for( j = 0; j < i; j++ )
        {
                if( export_img )
        export_frame( frames[j], j+1, std::string(folderpath) );

                if( export_vid )
                        int nret = cvWriteFrame( videoWriter, frames[j]);

                cvReleaseImage( &frames[j] );
        }

        if( export_vid )
                cvReleaseVideoWriter(&videoWriter);

        AfxMessageBox("finished");
        cvDestroyWindow("Video");
```

```
}

BOOL CTrack::PeekAndPump()
{
	MSG msg;

	while (::PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
	{
		if (!AfxGetApp ()->PumpMessage ())
		{
			::PostQuitMessage (0);
			return FALSE;
		}
	}

	LONG lIdle = 0;
	while (AfxGetApp ()->OnIdle (lIdle++));

	return TRUE;
}

////////////////////////////////////////////////////////////////////////////////
//  Computes a reference histogram for each of the object regions defined by the user
//
//  @param frame; video frame in which to compute histograms
//              should have been converted to hsv using bgr2hsv in observation.h
//  @param regions; regions of a frame over which histograms should be computed
//  @param export if TRUE, object region images are exported
//
//  @return Returns a normalized histograms
//                corresponding to regions of a frame specified in a regions.
////////////////////////////////////////////////////////////////////////////////

histogram** compute_ref_histos( IplImage* frame, CvBox2D region )
{
	IplImage* tmp;
	histogram** histos = (histogram**)malloc( sizeof( histogram ) * 2 );

	/* extract region from frame and compute its histogram */
	for ( int i = 0; i < 2; i++ )
	{
		cvSetImageROI( frame, cvRect( (int)(region.center.x - region.size.width/2.0f +
0.5f),
                                      (int)(region.center.y -
region.size.height/2.0f * ( 1 - i ) + 0.5f),
                                       region.size.width,
region.size.height /2 ));

		tmp = cvCreateImage( cvGetSize( frame ), IPL_DEPTH_32F, 3 );
		cvCopy( frame, tmp, NULL );
		cvResetImageROI( frame );
```

```
                histos[i] = calc_histogram( &tmp, 1 );
                normalize_histogram( histos[i] );
                cvReleaseImage( &tmp );
        }


        return histos;
}

////////////////////////////////////////////////////////////////////////////////////////
//  Mix a reference histogram with a dynamic histogram
//
//  @return Returns a normalized histograms
////////////////////////////////////////////////////////////////////////////////////////

histogram** compute_dyn_histos( IplImage** dyn_img, IplImage** ref_img)
{
        IplImage* tmp[2];
        histogram** histos = (histogram**)malloc( sizeof( histogram ) * 2 );

        /* extract region from frame and compute its histogram */
        for ( int i = 0; i < 2; i++ )
        {
                tmp[0]          = dyn_img[i];
                tmp[1]          = ref_img[i];

                histos[i]       = calc_histogram( tmp, 2 );

                normalize_histogram( histos[i] );
        }

        return histos;
}

////////////////////////////////////////////////////////////////////////////////////////
//  Exports a frame whose name and format are determined by EXPORT_BASE and
EXPORT_EXTN, defined above.
//
//  @param frame frame to be exported
//  @param i frame number
////////////////////////////////////////////////////////////////////////////////////////
int export_frame( IplImage* frame, int i, std::string folderpath )
{
        char* name;
        char num[5];

        name = (char*)malloc(strlen(EXPORT_BASE)+strlen(EXPORT_EXTN)+4);

        _snprintf( num, 5, "%04d", i );
        strcpy( name, EXPORT_BASE );
```

```
        strcat( name, num );
        strcat( name, EXPORT_EXTN );

   int nRet = cvSaveImage( (folderpath+std::string(name)).c_str(), frame );

        return nRet;
}

////////////////////////////////////////////////////////////////////////////////////
//
// Get object regions from file
////////////////////////////////////////////////////////////////////////////////////
int get_regions_from_file( char* filename, int** detectfr, CvBox2D** regions )
{
        params p;
        int * f;
        CvBox2D* r;

        int i;

        FILE *file;
        file = fopen (filename, "rb");

        p.n      = 0;

        while ( TRUE )
        {
                if (fscanf(file, "%d %d %d %d %d %d",
                                &p.center[p.n].x, &p.center[p.n].y, &p.width[p.n], &p.height[p.n],
&p.ang[p.n], &p.frame[p.n]) != 6)
                {
                        break;
                }

                p.n++;
        }

        /* extract regions defined by user; store as an array of rectangles */
        if( p.n == 0 )
        {
                *regions = NULL;
                return 0;
        }

        r = (CvBox2D*)malloc( p.n * sizeof( CvBox2D ) );
        f = (int*)malloc( p.n * sizeof( int ) );

        for( i = 0; i < p.n; i++ )
        {
                r[i].center.x     = (float)p.center[i].x;
                r[i].center.y     = (float)p.center[i].y;
```

```
                r[i].size.width = (float)p.width[i];
                r[i].size.height= (float)p.height[i];

                r[i].angle              = (float)(p.ang[i]);

                f[i]                    = p.frame[i];
        }

        *regions        = r;
        *detectfr       = f;

        return p.n;
}

////////////////////////////////////////////////////////////////////////////////////////
//
// Get the image of the object regions defined by the user
////////////////////////////////////////////////////////////////////////////////////////
void get_object_image( IplImage* img, IplImage** tmp, int r, int c, int w, int h)
{
        /* extract region around (r,c) */
        for ( int i = 0; i < 2; i++ )
        {
                cvSetImageROI( img, cvRect( c-w/2, r-h/2*(1-i), w, h/2 ) );

                tmp[i] = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 3 );
                cvCopy( img, tmp[i], NULL );
                cvResetImageROI( img );
        }
}

void get_param_from_file(CString folderpath)
{
        FILE *file;
   file = fopen ((std::string(folderpath)+std::string("\\param.txt")).c_str(), "rb");


        fscanf(file, "%d", &nChannel);
        fscanf(file, "%d", &nIter);
        fscanf(file, "%f", &nVar);
        fscanf(file, "%f", &sMin);
        fscanf(file, "%f", &sMax);
        fclose(file);

        if ( nChannel == 0 )
                nFeature = 2;
        else
                nFeature = 4;

        num_particles = nIter * MAX_CHAINS;}
```

ÖZGEÇMİŞ

Doğum Tarihi	27.06.1983	
Doğum Yeri	İzmit	
Lise	1997-2001	60.Yıl Anadolu Lisesi
Lisans	2002-2006	Kocaeli Üniversitesi Mühendislik Fakültesi Elektrik Mühendisliği Bölümü
Yüksek Lisans	2008-2011	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektrik Mühendisliği Bölümü, Kontol ve Otomasyon Prg.

Çalıştığı Kurumlar

	2010-2011	DFKİ (German Research Center For Artificial Intelligence)