T.C.
AKDENİZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ŞABLON EŞLEŞTİRME YÖNTEMİ İLE NESNE TAKİBİ VE YÜKSEK HIZLI FPGA GERÇEKLEMESİ

Hakan AKTAŞ

YÜKSEK LİSANS TEZİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

2015

T.C.
AKDENİZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ŞABLON EŞLEŞTİRME YÖNTEMİ İLE NESNE TAKİBİ VE YÜKSEK HIZLI FPGA GERÇEKLEMESİ

Hakan AKTAŞ

YÜKSEK LİSANS TEZİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez 2013.02.0121.013 proje numarası ile Akdeniz Üniversitesi Bilimsel Araştırma Projeleri Yönetim Birimi tarafından desteklenmiştir.

2015

T.C.
AKDENİZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ŞABLON EŞLEŞTİRME YÖNTEMİ İLE NESNE TAKİBİ VE YÜKSEK HIZLI FPGA GERÇEKLEMESİ

Hakan AKTAŞ

YÜKSEK LİSANS TEZİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİMDALI

Bu tez 22/06/2015 tarihinde aşağıdaki jüri tarafından oybirliği/oyçokluğu ile kabul edilmiştir.

Yrd.Doç.Dr. Refik SEVER(Danışman) ……

Yrd.Doç.Dr. Övünç POLAT ……

Doç.Dr. Habil KALKAN ……

ÖZET

ŞABLON EŞLEŞTİRME YÖNTEMİ İLE NESNE TAKİBİ VE YÜKSEK HIZLI FPGA GERÇEKLEMESİ

Hakan AKTAŞ

Yüksek lisans Tezi, Elektrik-Elektronik Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. Refik SEVER
Haziran 2015, 56 Sayfa

Görüntü işleme algoritmaları günümüzde birçok alanda kullanılmakta olup, bunlardan bir tanesi de nesne takibi uygulamalarıdır. Nesne takip sistemlerinin en yaygın kullanıldığı alanların başında askeri ve güvenlik uygulamaları gelmektedir. Günümüzde yaygın olarak kullanılan insansız hava araçlarının en temel işlevlerinden biri nesne takip uygulamalarıdır. Nesne takip sistemleri için önerilmiş bir çok yöntem olmakla birlikte bunlardan bir tanesi de Şablon Eşleştirme yöntemidir. Şablon Eşleştirme yönteminin doğruluk oranı çok fazla olmasına rağmen çok fazla işlem yükü gerektirmektedir. Söz konusu nesneyi takip etmek olunca, yapılan işlemin gerçek zamanlı olması gerekmekte; bu da yüksek hız ve performansı sağlayabilecek donanımların ve algoritmaların kullanılmasını zorunlu kılmaktadır.

Bu çalışmada işlem yoğunluğu yüksek Şablon Eşleştirme Algoritması için tur sayısını ve işlem yükünü azaltacak iki aşamalı yeni bir algoritma önerilmiştir. Eşleştirme için Farkların Mutlak Değerlerinin Toplamı metodu kullanılmıştır. Geliştirilen bu algoritma, Alan Programlanabilir Kapı Dizisi (FPGA) üzerinde gerçeklenmiştir. Algoritma, FPGA üzerindeki rastgele erişimli bellek öbekleri (BlokRAM) efektif bir şekilde kullanılarak daha da hızlandırılmıştır. Bu sayede, geliştirilen algoritma, İnsanız Hava Aracı (İHA – UAV) görüntülerinde gerçek zamanlı nesne takibi yapabilecek hale getirilmiştir.

ANAHTAR KELİMELER: Şablon Eşleştirme, Farkların Mutlak Değerlerinin Toplamı, Alan Programlanabilir Kapı Dizisi (FPGA), Paralel İşleme, Hafıza Adresleme, İnsansız Hava Aracı (İHA)

JÜRİ: Yrd. Doç. Dr. Refik SEVER (Danışman)
Yrd. Doç. Dr. Övünç POLAT
Doç. Dr. Habil KALKAN

ABSTRACT

OBJECT TRACKING WITH TEMPLATE MATCHING METHOD AND ITS HIGH SPEED FPGA IMPLEMENTATION

Hakan AKTAŞ

M.Sc. Thesis in Electrical-Electronics Engineering
Supervisor: Assist. Prof. Refik SEVER
June, 2015, 56 pages

Image processing algoritms are widely used in many application areas, such as object tracking systems. The object tracking systems are mainly used in security and defence applications. Unmanned Air Vehicles which are used widely nowadays, have the major functions of object tracking applications. There are many techniques for object tracking systems and one of them is Template Matching technique. Although the accuracy is very high in this technique, it has very big computationalcost. As the aim is to track the object, the system has to be operated in real time. To achieve a real time operation the algorithm and logic sources must be optimized.

In this study, to decrease the computational cost and number of cycles in Template Matching Algorithm, a novel two-stage algorithm is proposed. The Sum of Absolute Differences method is used for matching. The proposed algorithm is implemented on Field-Programmable-Gate-Array (FPGA). The algorithm is accelerated with the effective usage of Block RAMs distributed on FPGA. Thus, the proposed algorithm has become fast enough for real time object tracking applications on UAVs.

KEYWORDS: Template Matching, Sum of Absolute Differences, Field-Programmable-Gate-Array (FPGA), Parallel Processing, Memory Addressing, Unmanned Air Vehicle (UAV)

COMMITTEE: Assist. Prof. Refik SEVER (Supervisor)
Assist. Prof. Övünç POLAT
Assoc. Prof. Habil KALKAN

ÖNSÖZ

Yüksek Lisans dönemim boyunca yardımlarını ve desteklerini benden esirgemeyen, gelecekteki akademik çalışmalarıma yön veren ve her zaman örnek alacağım danışman hocam Refik SEVER'e teşekkürü bir borç bilirim.Yine tezime sağladığı katkılardan dolayı Yrd. Doç. Dr. Ümit Deniz ULUŞAR hocama ve Yrd. Doç. Dr. Behçet Uğur TÖREYİN hocama da ayrıca teşekkürlerimi sunarım.Ayrıca üç yıldır Akdeniz Üniversitesi Elektrik-Elektronik Mühendisliği bölümünde beraber çalıştığımız çok kıymetli hocalarıma da teşekkür ve saygılarımı sunarım.

Maddi manevi her zaman yanımda olan aileme sevgilerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler

Kısaltmalar

İHA	İnsansız Hava Aracı
SAD	Sum of Absolute Difference
FPGA	Field Programmable Gate Array
FFS	Full Frame Searching
ALU	Arithmetic Logic Unit
CPU	Central Processing Unit
GPU	Graphic Processing Unit
RAM	Random Access Memory
CLB	Configurable Logic Blocks

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

İnsansız hava araçları (İHA-UAV) günümüzde askeri ve güvenlik uygulamalarında yaygın bir şekilde kullanılmakta olup, daha çok arazi gözetleme, nesne takibi ve istihbarat işlemlerinde kullanılmaktadır. Nesne tespiti ve takibi literatürde kendine geniş bir yer bulmakta olup, önerilmiş birçok algoritma ve yöntem bulunmaktadır. Bu yöntemlerden bir tanesi de Şablon Eşleştirme yöntemidir. Bu yöntemde doğruluk oranı yüksek olmakla birlikte işlem yükü çok fazladır. Öyle ki hızlandırma tekniklerini kullanmadan günümüz donanımlarıyla gerçeklemek oldukça zordur. Söz konusu bir nesneyi takip etmek olunca işlemlerin gerçek zamanlı bir şekilde yapılması gerekmektedir. Bu da kullanılan donanım ve algoritmaların çok hızlı olmasını gerektirmektedir.

Bilgisayar görmesi uygulamalarında, doğadaki bir nesneyi gerçek zamanlı olarak takip etmek oldukça zordur (Kettnaker ve Zabih 1999). Nesne takip etme, hareket halindeki bir nesnenin arka arkaya gelen çerçevelerdeki yörüngesini tahmin etmeyi içermektedir. Nesneleri ifade eden özellikler, nesnelerin hareketleri ve geçici değişiklikler ya da takip edilen nesnenin tam olarak takılması gibi dinamik değişkenlerin takip etmede göz önünde bulundurulması gerekmektedir (Hanna 2011). Bu monograf, nesne takip sistemlerinin gelişmesini, metot ve sistemlerini ve ayrıca nesne takip sistem yapısını ve önerilen sistemlerde yeni trendleri ifade etmektedir.

Yılmaz ve arkadaşlarına göre, doğadaki görüntülerin 3 boyutlu olmasına rağmen elde edilen görüntülerin 2 boyutlu olması, görüntü üzerine gürültü binmesi, nesnelerin hareketlerinin kompleks yapıda olabilmesi, düzgün olmayan ya da parçalı nesnelerin bulunması (Xu vd 2004), kısmi ve tam nesne örtüşmeleri, karmaşık nesne şekillerinin bulunması, görüntü parlaklığının değişmesi, gerçek zamanlı uygulama ihtiyaçları ve hareket eden nesneden oluşan gölgeler nesne takibinin karmaşık bir yapıda olmasına sebep olmaktadır (Yilmaz vd 2006).

Hu ve arkadaşları 2004 yılında nesne takip sistemlerini alan-tabanlı takip, aktif çevrit-tabanlı takip, nitelik-tabanlı takip ve model-tabanlı takip sistemleri olarak dört ana başlık altında toplamışlardır. Şablon eşleştirme yöntemi, alan-tabanlı nesne takip sistemleri içerisinde en yaygın kullanılan yöntemdir. Yine, şablon eşleştirme algoritması alan-bazlı nesne takibi için en güvenilir ve algoritma olarak uygulanması en kolay yöntemdir. Tüm çerçeve taraması şablon eşleştirmenin en genel hali olup, tüm çerçeve taraması ile yapılan eşleştirmelerde yüksek doğrulukta sonuçlar elde edilmesine rağmen işlem yükü çok fazladır.

Literatürde Tüm Çerçeve Taramasını hızlandırmak için birçok yöntem ve algoritma önerilmiştir. Huang ve arkadaşları 2006'da bu hızlandırma algoritmalarını 6 başlık altında toplamışlardır. Bu başlıklar: Arama Noktalarını Azaltma, Eşleşme Kriterlerini Basitleştirme, Bit Genişliğini Azaltma, Tahminlere Göre Tarama, Hiyerarşik Tarama ve Hızlı-Tüm Tarama'dır.

Li ve akadaşları 1991'de Paralel Hiyerarşili Bir Boyutlu Arama (Parallel Hierarchical One Dimensional Search), ve Huang ve arkadaşları 2003'te Tahmin Edilebilir Çizgi Araması algoritmasını önererek, tüm çerçeve algoritmasını

hızlandırmak için yeni algoritma önermekle birlikte paralel işlemenin imkanlarından, veri akışından ve bellek öbeklerinin efektif bir şekilde kullanılmasından da faydalanmışlardır.

Şablon eşleştirmede en iyi eşleşme tüm çerçeve taraması ile elde edilmekte olup, tüm çerçeve taraması genelde donanımlar üzerinde gerçeklenmektedir. FPGA'nın paralel işlem yapabilme özelliği sayesinde son yıllarda birçok FPGA tabanlı tasarım önerilmiştir. Wong ve arkadaşları 2002'de SAD16 olarak adlandırılan 16X1-SAD yapısını gerçeklemişlerdir. Bu yapı toplama-ağacı yapısından ilham alınarak gerçeklenmiştir. Bu çalışmalarında önerdikleri 16X1-SAD yöntemini genişleterek 16X16-SAD gerçeklemesini nasıl yapacaklarından bahsetmişlerdir.

Roma ve arkadaşları 2003'te silindirik yapı ve zig-zag işlem dizisi üzerine kurulu yenilikçi bir proses şeması önermişlerdir. Bu silindirik yapı aktif ve pasif çerçeve bloklarından oluşmaktadır. Pasif çerçeve blokları okunup hafızaya yazdırılırken, aktif çerçeve blokları işlenmektedir. Böylece çakışık çerçeve bloklarının düzenli data akışı ve tekrar kolay kullanımı sağlanmış olmaktadır. Ancak bu tasarım donanım maliyetini önemli ölçüde artırmaktadır.

Olivares ve arkadaşları 2005'de çevrimiçi aritmetiğe (OLA-Online Arithmetic) dayalı özgün bir yapı önermişlerdir. OLA, her biti bir saat darbesinde işleyecek şekilde seri-bit modda çalışmaktadır. En büyük değerlikli bit (MSD - Most Significant Bit) ilk önce işlenerek, farkların mutlak değeri ve karşılaştırma işlemlerini kolaylaştırmaktadır. Yine bu kolaylaştırmanın hesap yüküne herhangi bir etkisi olmamaktadır. Çevrimiçi toplama-ağacı büyük bloğu işleme imkanı sunup; son bir çevrimiçi karşılaştırma devresi istenilen SAD değerine ulaşılır ise işlemlerin durdurulmasını sağlamaktadır. Böylece bu mimari ardışık düzen (pipeline) yapısını bir bit seviyesine yükseltmeyi başarmıştır.

Tüm çerçeve taraması (FFS – Full Frame Searching) ile yapılan şablon eşleştirme yöntemi, eşleşme için en doğru sonucu verse de işlem yükü çok fazla olmasından dolayı literatürde FFS'i hızlandırmak için önerilmiş birçok yöntem bulunmaktadır. Önerilen bu yöntemlerin birçoğu yazılım uygulamaları için uygun olmakla birlikte donanım uygulamaları için daha az yöntem önerilmiştir. Paralel işleme özelliğine sahip FPGA gibi donanımların kullanılması ile birlikte bu tür işlemler daha hızlı yapılabilir hale gelmiştir. Literatürde FFS algoritmasını hızlandırmak için birçok yöntem önerilmesine rağmen bunların hepsini doğrudan FPGA platformu üzerinde gerçeklemek büyük donanımsal maliyetlere sebep olmaktadır.

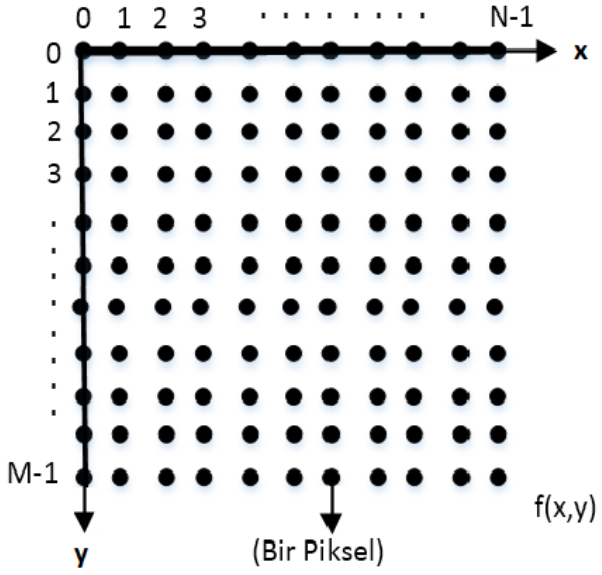
Bu tezde, şablon eşleştirme yöntemi kullanılarak alan-bazlı takip yapmak ve uygulamanın son halini FPGA üzerinde gerçek zamanlı olarak gerçekleyebilmek için iki aşamalı bir yöntem önerilmiştir. Önerilen yöntemin son hali FPGA donanımı üzerinde gerçekleneceği için yöntemin FPGA donanımsal maliyetinin düşük olmasına ve uygulanabilirliğinin kolay olmasına özen gösterilmiş ve yöntem buna göre önerilmiştir. Donanımsal maliyeti düşürmek ve bellek adreslerini bir döngü içerisinde üretebilmek için 2^n mimarisi kullanılmıştır. Böylece önerilen yöntem ve bellek adreslerini üretmek için gerekli tüm parametreler 2^n kümesinden seçilmiştir.

İşlem yükü yüksek olan şablon eşleştirme yönteminin, FPGA donanımları üzerinde düşük donanım maliyetleri ve gerçek zamanlı olarak gerçeklenmesi literatürde ve uygulamada kendine geniş bir yer bulmakta olup; bu tez çalışmasının 2. kısmında kısaca görüntü işlemeden, şablon eşleştirme yönteminden ve literatürde bulunan şablon eşleştirme hızlandırma yöntemlerinden bahsedilmiştir. 3. kısımda uygulamanın geliştirileceği Matlab ve FPGA platformlarından ve bu platformlar üzerinde gerçekleştirilecek görüntü işleme uygulamaları hakkında bilgiler verilmiş olup; seri ve paralel işlemenin birbirine göre farklarından kısaca bahsedilmiştir. 4. kısımda önerilen yöntemden ve bu yöntemin farklı çerçeveler üzerindeki test sonuçlarına çizelgeler üzerinden bahsedilmiştir. Yine 4. kısımda önerilen yöntemin FPGA donanımı üzerinde gerçeklenmesinden ve sonuçlarından bahsedilmiş olup; 5. kısımda çalışma ile varılan sonuç ve tartışmaya yer verilmiştir.

2. KURAMSAL BİLGİLER

2.1. Görüntü İşleme

Görüntü İşleme, görüntüyü dijital forma çeviren ve dijital hale gelen görüntü üzerinde bir takım işlemler yaparak görüntüyü istenilen forma sokan veya görüntü üzerinden istenilen gerekli bilgileri elde eden metotlardır. Bir diğer tanımla, giriş olarak gelen video görüntülerini ya da kayıtlı görüntüleri çıkışta işlenmiş görüntüye ya da görüntüyle alakalı karakteristiklere dönüştüren sinyal işleme uygulamalarıdır (Gonzalez ve Richard 2008). Görüntü işlemede görüntülere gerekli sinyal işleme metotları uygulanılırken görüntü iki boyutlu bir sinyal olarak ele alınır. M satır ve N sütuna sahip olan dijital bir görüntü f(x,y) formatında gösterilir. (x,y) koordinatlarındaki değerler bu şekilde ayrık değerlere dönüşürler. Notasyon olarak açıklık ve kolaylık sağlamak için sayı değerleri bu ayrık koordinatlar için kullanılır. Böylece orjin üzerindeki koordinatların değerleri (x,y) = (0,0) olarak gösterilir. Aynı şekilde aynı satır üzerindeki bir sonraki koordinat (x,y) = (0,1) olarak gösterilir. Şekil 2.1'de M satır N sütuna sahip bir görüntü koordinat halinde gösterilmektedir. f(x,y) her bir elemanı bir pikseli göstermek üzere; M satır ve N sütuna sahip bir görüntüde MxN adet piksel bulunmaktadır.



Sekil 2.1. Dijital Görüntünün Koordinat Olarak Gösterimi

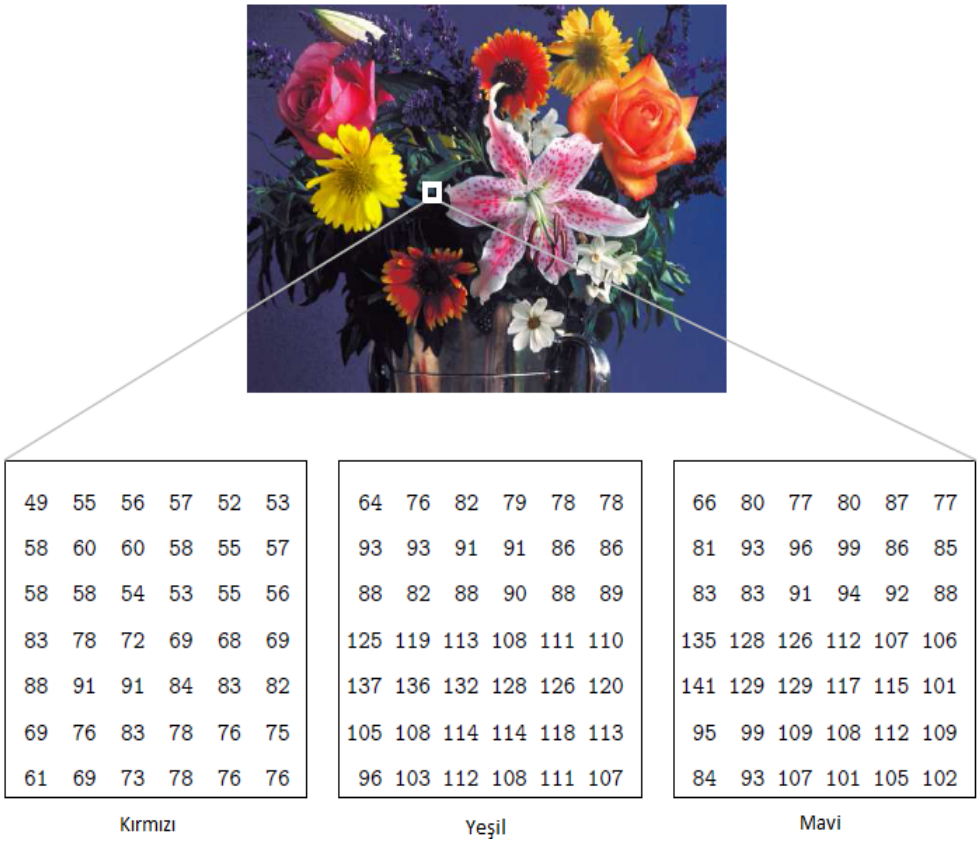
Daha önce tanımlanan bu notasyon, bize MxN boyutundaki görüntüyü aşağıdaki gibi ifade etmemizi sağlar:

$$f(x,y) = \begin{bmatrix} f(0,0) & ,f(0,1), & \dots & f(0,N-1) \\ f(1,0) & ,f(1,1), & \dots & f(1,N-1) \\ & . & & \\ & . & & \\ f(M-1,0) & ,f(M-1,1), & \dots & f(M-1,N-1) \end{bmatrix}$$

2.1.1. Dijital görüntü çeşitleri

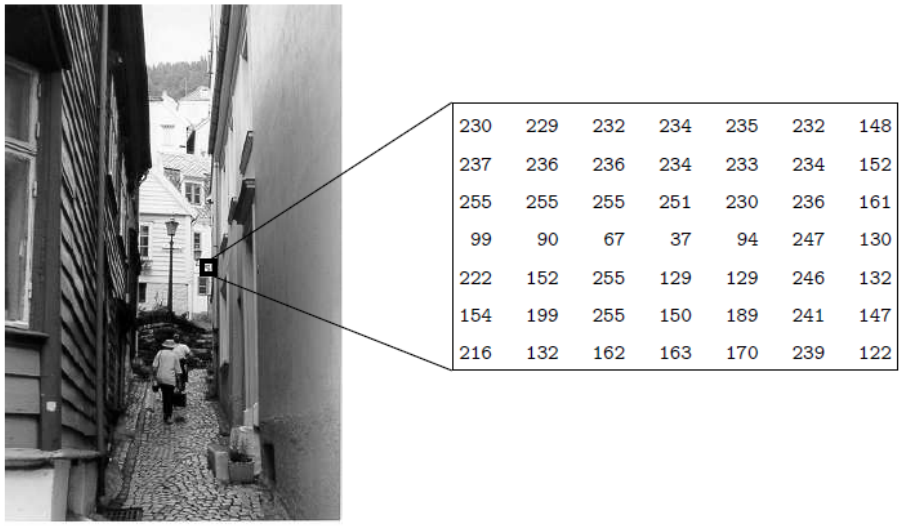
Dijital Görüntüler üç ana başlık altında toplanmaktadır. Bunlar :

RGB ya da Gerçek-Renk Görüntüler: Her bir pikseli üç elemanla ifade edecek 3 boyutlu dizilerden oluşmaktadır. Bu elemanlar sırasıyla Kırmızı, Yeşil ve Mavi (Red Green Blue) değerlerini ifade etmektedir (Chris Solomon 2011). Her bir eleman (R,G,B) 0-255 arasında bir değer alırsa resimdeki pikseller $255^3 = 16777216$ farklı renk ile ifade edilebilir demektir. Bu büyüklükteki bir renk çeşiti tüm görüntüler için yeterlidir. Bu sayede istenilen tüm renkler bu kombinasyona göre çok rahat bir şekilde ifade edilebilmektedir. Her bir pikseli ifade etmek için 24 bit gerekmekte olup, bu resimlere ayrıca 24-bitlik resimler de denilmektedir. Bu şekildeki bir görüntüde her bir piksel, R, G, B matrislerinin bir araya gelmesiyle oluşan yığınlarla ifade edilmektedir. Şekil 2.2'de örnek bir RGB görüntüsünü ve R, G, B değerlerini göstermektedir.



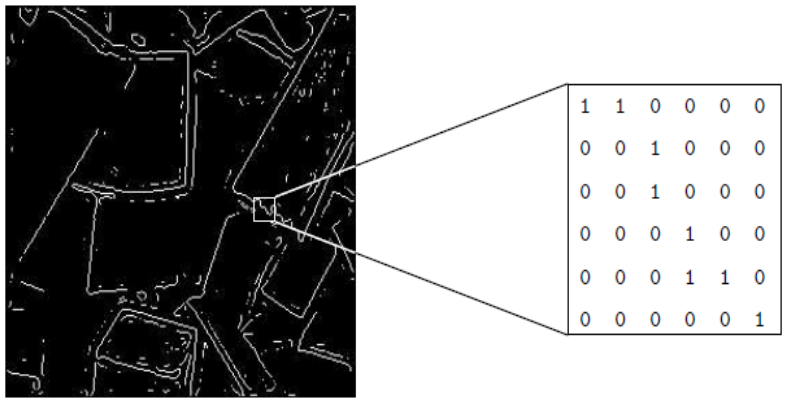
Sekil 2.2. RGB Görüntü

Gri Görüntü: Her bir piksel 0'dan (siyah) 1'e (beyaz) kadar grinin tonlarını içeren renklerden oluşmaktadır. Bu aralık, her bir pikselin 8 bit ya da 1 bayt ile ifade edildiğini göstermektedir. Bazı gri görüntülerde pikseller farklı sayıda bit değerleriyle gösterilse de yaygın kullanım 8 bitlik gösterimdir. Şekil 2.3'te örnek bir gri görüntü piksel değerleri gösterilmektedir.



Şekil 2.3. Gri Görüntü

İkili Görüntü: Her bir piksel siyah ya da beyazdır. Her bir piksel için sadece iki renk ihtimali olduğu için her bir piksel 1 bit ile ifade edilir. Düşük veri boyutları sayesinde bu tür görüntüler hafızada az yer kaplayacak şekilde kaydedilirler. Şekil 2.4'te örnek bir ikili görüntü gösterilmektedir. Şekilden de görüldüğü gibi resim sadece siyah ya da beyaz renkleri içermektedir.



Şekil 2.4. İkili Görüntü

2.1.2. Temel görüntü işleme metotları

Görüntü İşlemede birçok algoritma kullanılmaktadır. Kullanılan bu algoritmaları temel gruplar halinde ele alarak incelersek bunlar:

Görüntü İyileştirme: Temel olarak görüntü üzerinde bir takım iyileştirmeler yaparak insanların görüntüyü daha iyi yorumlayabilmesi ve görmek istedikleri şeyi daha iyi algılayarak bu sayede uygulanacak görüntü işleme algoritması için daha iyi bir girdi oluşturacak görüntü sağlamasıdır (Maini ve Aggarwal 2010). Görüntü iyileştirmenin temel amacı görüntünün özniteliklerini modifiye ederek, gözlemci için daha uygun bir

görüntü meydana getirmektir. Görüntü iyileştirme ile görüntünün köşeleri, kenarları daha netleştirilerek ya da kontrast veya aydınlık değeri daha keskinleştirilerek gözlemci için daha net ve anlaşılır bir görüntü elde edilir.

Görüntü Yenileme: Görüntü bazı önemli özelliklerini kaybettikten sonra ya da bozulduktan sonra tekrardan düzgün görüntü oluşturulabilmektedir. Görüntü sayısallaştırılırken ya da aktarım sırasında bozulmaya uğrayabilmektedir. Örneğin kamera ile görüntü çekerken, görüntüye kamera donanımından, görüntü sensörlerinden ya da görüntünün çekildiği çevreden bir miktar gürültü binebilmektedir. Görüntü yenileme metotları sayesinde görüntü üzerinde oluşan gürültü bazı yöntemlerle giderilmekte böylece orjinal görüntüye en yakın görüntü elde edilebilmektedir.

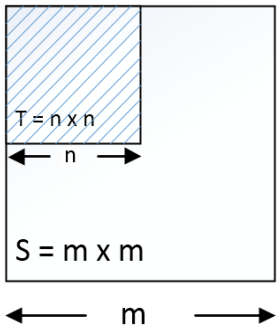
Görüntü Bölütleme: Görüntüdeki benzer özellikleri bir alan içerisine alarak görüntüyü anlamlı parçalara bölen ya da parçaları birbirinden ayıran işlemlerdir. En yaygın bölütleme metodu, tek renkli kamera görüntülerindeki parlaklık genliği özelliklerini ya da renkli görüntüler için renk özelliklerini kullanmaktır. Görüntüdeki köşe bilgileri ve doku bilgileri de yine bölütleme için önemli özelliklerdir (Pratt 2007). Görüntü Bölütleme ile bir görüntüdeki çizgiler, dairler ya da aratılmak istenilen belirgin bir şekil bulunabilir. Yine havadan çekilmiş fotoğraflardan, arabaları, evleri, yolları, ağaçları tespit etme işlemleri yapılabilmektedir.

Nesne Tanıma: Aratılmak istenilen nesneyi bir görüntüde ya da kameradan gelen görüntüler üzerinde bulma işlemidir. Görüntü içerisindeki her hangi bir nesne için, nesne üzerinden çıkarılabilecek ve nesne hakkındaki özellikleri tanımlayacak çok fazla nokta sayısı bulunmaktadır. Nesne üzerinden çıkarılan ve nesne hakkında nitelikli bilgileri veren bu noktalar, aratılmak istenen nesnenin birden fazla nesne ile aynı çerçevede bulunduğu görüntüde kullanılarak nesneyi tanımaya ve tespit etmeye yardımcı olmaktadır (Ferrari vd 2004).

2.2. Şablon Eşleştirme Yöntemi

Şablon, bulunmak istenen alt-görüntü (sub-image) olmak üzere şablon eşleştirmedeki temel amaç bu alt-görüntüyü aramanın yapılacağı resimlerde arattırarak şablonunun eşleşmelerini bulmaktır. Şablon eşleştirme algoritması hareket analizi, video takip etme, video sıkıştırma ve kodlama, tıbbi görüntüleme ve kriminal görüntülerin analizi gibi görüntü işlemenin birçok alanında yaygın olarak kullanılmaktadır. Şablon eşleştirme algoritmasında bulunmak istenilen örnek resim (Şablon), bir sonraki çerçevede şablon eşleştirme yöntemlerinden uygun olanı kullanılarak bulunmaktadır.

Şablon Eşleştirmede en temel metod Tüm Çerçeve Taraması Algoritması (Full-Frame Searching Algorithm – FFS) gerçeklemesidir (Mattoccia vd 2008). Şablon eşleştirme yöntemi: örnek çerçeve (T), eşleştirilmek istenilen çerçeve (S) üzerinde aranarak maksimum çapraz korelasyonun ya da minimum bozulmanın bulunduğu konumdaki eşleştirmedir. Aramanın yapılacağı çerçeve Şekil 2.5'teki gibi $S = mXm$, referans çerçeve $T = nXn$ ve $m > n$ olmak üzere T'nin S üzerindeki tüm piksellerde gezdirilmesi sonucu oluşan eşleştirmeye Tüm Çerçeve Tarama Algoritması denir.



Şekil 2.5. S ve T Çerçeveleri

2.2.1. Şablon eşleştirme teknikleri

Şablon eşleştirme algoritması literatürde alan-tabanlı (area-based) ve öznitelik-tabanlı (feature-based) algoritmalar olarak iki genel kategoride incelenmektedir (Brown vd 1992). Alan-Bazlı algoritmalardan en bilindik olanları: Farkın Tam Değerinin Ortalaması (Mean Absolute Difference - MAD), Normalize Edilmiş Çapraz-İlinti (Normalized Cross Correlation - NCC), Ardışıl Benzerlik Algılaması (Sequential Similarity Detection Algorithm - SSDA), Farkların Mutlak Değerlerinin Toplamı (Sum of Absolute Differences - SAD), Farkların Karelerinin Toplamı (Sum of Squared Differences - SSD) algoritmalarıdır. Öznitelik-tabanlı algoritmalarda ise şablondaki nesnelere ait kenar, köşe, şekil, doku ve çevrit bilgileri çıkartılarak bu bilgiler bir sonraki çerçeve üzerinde arattırılır (Lowe 2004).

Şekil 2.5'teki gibi T = nXn ve S = mXm; T'ye ait pikseller T(i, j) , S'e ait pikseller S(i, j), r = {0,1, …, m-n} ve c = {0,1, …, m-n} olmak üzere satırda ilerleme r ile, sütunda ilerleme c ile gösterilmek üzere SAD, MAD, SDD ve NCC Algoritmalarının genel denklemleri sırasıyla 1, 2, 3, 4'teki gibidir.

$$SAD(r,c) = \sum_{j=1}^{n}\sum_{i=1}^{n}|S(i+r,j+c) - T(i,j)| \quad (1)$$

$$MAD(r,c) = \frac{1}{nXn}\sum_{j=1}^{n}\sum_{i=1}^{n}|S(i+r,j+c) - T(i,j)| \quad (2)$$

$$SDD(r,c) = \sum_{j=1}^{n}\sum_{i=1}^{n}(S(i+r,j+c) - T(i,j))^2 \quad (3)$$

$$NCC(r,c) = \frac{\sum_{i=1}^{n}\sum_{i=1}^{n}[S(i+r,j+c) - \bar{S}\,(i,j)][T(i,j) - \bar{T}]}{\sqrt{\sum_{i=1}^{n}\sum_{i=1}^{n}[S(i+r,j+c) - \bar{S}\,(i,j)]^2 \sum_{i=1}^{n}\sum_{i=1}^{n}[T(i,j) - \bar{T}\,]^2}} \quad (4)$$

Denklem 1'e göre T (şablon) pikselleri her bir turda bu piksellere karşılık gelen S piksellerinden çıkartılır. Oluşan farkların mutlak değerleri birbirleri ile toplanarak her bir tur için SAD(r,c) değeri hesaplanır. SAD(r,c) değerinin en küçük olduğu turda T şablonu (r,c) değerleri için S çerçevesi üzerinde eşleşiyor denir.

Denklem 2'ye göre MAD(r,c) değerinin en küçük olduğu turda T şablonu (r,c) değerleri için S çerçevesi üzerinde eşleşiyor denir. MAD algoritması SAD ile benzerlik gösterip tek farkı, elde edilen toplam sonucunun nXn sayısına yani şablondaki piksel sayısına bölünerek sonucun normalize edilmesi işlemidir. Bu durumda denklem 2, denklem 1'e göre 5'teki gibi ifade edilebilir.

$$MAD(r,c) = \frac{1}{nXn} SAD(r,c) \tag{5}$$

SDD, SAD ve MAD algoritmaları denklem olarak birbirlerine benzemek üzere, işlem yükü olarak en belirgin farkları MAD'de her turda bölme işleminin bir defa yapılması, SDD tekniğinde ise çarpma işleminin her turda nXn kere yapılmasıdır. Bu açıdan bakıldığında SDD işlemi MAD ve SAD algoritmalarına göre çok daha fazla işlem yükü içermektedir. SDD algoritmasında SDD(r,c) değerinin en küçük olduğu turda T şablonu (r,c) değerleri için S şablonu üzerinde eşleşiyor denir.

Normalize Edilmiş Çapraz-İlinti Algoritmasının denklemi 4'teki gibi olup buradaki $\bar{S}$ve$\bar{T}$ifadelerinin denklemleri aşağıdaki gibidir.

$$\bar{S}(r,c) = \frac{1}{nXn} \sum_{j=1}^{n} \sum_{i=1}^{n} S(i+r, j+c) \tag{6}$$

$$\bar{T}(r,c) = \frac{1}{nXn} \sum_{j=1}^{n} \sum_{i=1}^{n} T(i,j) \tag{7}$$

NCC algoritmasında NCC(r,c) -1 ile 1 arasında değerler almaktadır. NCC(r,c) 1 değerini alınca tam eşleşme gerçekleşir, -1 değerini alır ise negatif eşleşme gerçekleşir. Eşleşmenin olması için 1'e yakın değerlerin bulunması gerekmektedir. 4'teki gibi ifade edilen NCC denklemi, donanım üzerinde daha efektif bir şekilde gerçeklemek için 8'deki formda yazılabilir (Sahani vd 2011).

$$NCC(r,c) = \frac{\sum_{i=1}^{n} \sum_{i=1}^{n} [S(i+r, j+c)T(i,j) - n^2 \bar{S}\,(i,j)\bar{T}]}{\sqrt{[\sum_{i=1}^{n} \sum_{i=1}^{n} [S^2(i+r, j+c) - n^2 \bar{S}^2(i,j)]][\sum_{i=1}^{n} \sum_{i=1}^{n} [T^2(i,j) - n^2 \bar{T}^2]]}} \tag{8}$$

Her metodun kendine göre farklı uygulama alanları, avantajları ve dezavantajları bulunmaktadır. SSD metodu, matematiksel ifadesi ve uygulanabilirliğinin çok iyi olmasından dolayı şablon eşleştirme dahil olmak üzere en bilindik mesafe ölçüm algoritmasıdır. Buna karşın, SSD metodu gürültü ve parlaklık değişimlerine karşı çok duyarlıdır (Brahim vd 2011). NCC metodu yüksek hızlı endüstriyel uygulamalarda kullanılmaktadır. Birden fazla şablonu aynı anda bulmak için çok uygun bir metottur. Fakat bu yöntem parlaklık değişiklikleri ve nesnenin yer değiştirmesi sonucu hızlıca hata verebilmektedir. Örneğin dış ortamdaki güneş ve bulut etkilerinin değişmesi NCC metodunun uygulanabilirliğini çok hızlı etkilemektedir. Ancak kayıtlı resimler üzerinden bir şablonu arattırıp bulmak için çok efektif bir metottur. NCC ve SDD'nin parlaklık ve karmaşık arka plandan dolayı çok çabuk ve kötü etkilenmesi sorunu SAD metodu tarafından çözülmektedir (Saravan vd 2013). SAD metodu nesne takibi ve dış uygulamalarda yaygın olarak kullanılmasına rağmen piksel koordinatlarının tam tespiti için bir miktar optimizasyon ve geliştirme gerektirmektedir.

2.2.2. Literatürde şablon eşleştirme ve hızlandırma teknikleri

Şablon eşleştirmede doğruluk seviyesi en yüksek metot Tüm Çerçeve Taramasıdır. Bu metodun uygulaması çok kolay olmasına karşın işlem yüklü çok fazladır. Bu yöntemde örnek çerçeve (T = nXn), eşleştirilmek istenilen çerçeve (S=mXm) olmak üzere, SAD, SDD, MAD ya da NCC algoritmalarından herhangi biri kullanılarak FFS uygulanmak istenilirse, işlenmesi gereken piksel sayısı (Number Of Pixel) 9'daki gibi hesaplanır.

$$NOP = (m - n + 1)^2 n^2 \quad (9)$$

Tek bir şablon eşleştirmesi için işlenmesi gereken piksel sayısı günümüz bilgisayarları ile çok rahat işlenmesine rağmen, gerçek zamanlı bir uygulamada SAD, SDD, MAD ya da NCC metotlarından herhangi birini kullanarak Tüm Çerçeve Taraması yapılmak istenilirse; işlenmesi gereken piksel sayısı, m ve n'e bağlı olarak karesi ile orantılı, kamera hızına bağlı olarak da doğru orantılı olarak artmaktadır. Bu da Tüm Çerçeve Taramasını gerçek zamanlı olarak gerçeklemeyi zorlaştırmakta; m, n ve kamera hızına göre bazen imkansız hale getirmektedir. Literatürde Tüm Çerçeve Taramasını hızlandırmak için birçok yöntem ve algoritma önerilmiştir. Huang ve arkadaşları 2006'da bu hızlandırma algoritmalarını 6 başlık altında toplamışlardır. Bu başlıklar: Arama Noktalarını Azaltma, Eşleşme Kriterlerini Basitleştirme, Bit Genişliğini Azaltma, Tahminlere Göre Tarama, Hiyerarşik Tarama ve Hızlı-Tüm Tarama'dır.

2.2.2.1. Arama noktalarını azaltma

Tüm Çerçeve Taramasında tüm pikselleri gezerek en doğru sonuca ulaşılsa da gezdirilen piksel sayısını azaltarak da doğru sonuca ulaşmak mümkün olabilmektedir. Aranan noktalar azaltılarak bu sayede yapılması gereken işlem sayısı önemli ölçüde azalmaktadır. 1981'den beri arama noktalarını azaltan bir çok algoritma önerilmiştir. Bunlar, İki-Boyutlu Logaritmik Aramalar (Two Dimensional Logarithmic Search) (Jain vd 1981), Üç Aşamalı Arama (Three Step Search) (Koga vd 1981), Birleşik Yönlü Arama (Conjugate Direction Search) (Srinivasan vd 1985), Modifiye Edilmiş Logaritmik Arama (Modified Logarithmic Search) (Kappagantula ve Rao 1985), Çapraz Arama (Cross Search) (Ghanbari 1990), Paralel Hiyerarşili Bir Boyutlu Arama (Parallel Hierarchical One Dimensional Search) (Chen vd 1991), Bir Boyutlu Tüm Arama (One Dimensional FullSearch) (Chen vd 1994), Üç Aşamalı Arama (Three step search) (Li vd 1994), Dört Aşamalı Arama (Four Step Search) (Po ve Ma 1996), Blok-Tabanlı Gradyan Düşmeli Arama (Block-Based Gradient Descent Search) (Liu ve Feig 1996), Merkez Eğimli Karo Arama (Centerbiased Diamond Search) (Zhu ve Ma 2000), Gelişmiş Karo Bölgesel Arama (Advanced Diamond Zonal Search) (Tourapis vd 2002), Minimum Sınırlı Alan Araması (Minimum Bounded Area Search) (Christopoulos ve Cornelis 2000), Tek-Boyutlu Gradyan Düşmeli Arama (One-Dimensional Gradient Descent Search) (Chen 2000), Çapraz Karo Arama (Cross Diamond Search) (Cheung ve Po 2002), Tahmin Edilebilir Çizgi Araması (Predictive Line Search) (Huang vd 2003) ve diğer birçok algoritma bu alanda önerilmiştir. Önerilen her yeni algoritma eskisine göre daha az işlem yükü gerektiren daha hızlı yakınsamaya sahip ve çözünürlüğü daha yüksek olan görüntülerde uygulanabilecek şekilde önerilmiştir. Huang ve arkadaşları 2006'da eşleştirme için gerekli işlem sayısını azaltmakla kalmayıp aynı zamanda veri

akışını düzenleyerek ve hafızaları daha etkin kullanarak paralel işleme sayesinde algoritmayı daha efektif hale getirmişlerdir.

2.2.2.2. Eşleşme kriterlerini basitleştirme

Tüm Çerçeve Taramasında, eşleştirme işlemi için şablon üzerindeki tüm pikseller işleme sokulmaktadır. İşlem yükünü azaltmak için ilk olarak alt-şablonlar kullanılması yöntemi önerilmiştir (Bierling 1988). Bu yöntemde bozunmayı tahmin etmek, eşleşmeyi bulmak için yatayda ve düşeyde her iki pikselden sadece bir tanesi işleme sokulur. Böylece işlem yükü dört kat azalmış olur. Bu yöntemde Alçak Geçiren filtreler kullanılarak bozunumların önüne geçilmektedir. Bozunumların önüne geçmek ve alçak geçiren filtreden kurtulmak için daha sonradan dört tane alt-şablonun farklı arama bölgelerine uygulanması önermiştir (Liu ve Zaccarin 1993). Daha sonra ise adaptif piksel-seyreltme şeması önerilmiştir (Wang vd 2000). Bu yöntemde eşleştirme için tüm şablon kullanılması yerine eşleştirmeyi sağlayacak şablon üzerindeki belirgin özelliklerin kullanılması amaçlanmıştır.

2.2.2.3. Bit genişliğini azaltma

Uygulamada her piksel 8-bit ile ifade edilmektedir. Luo ve arkadaşları 2002'de her pikseli 1-bit ile ifade ederek geleneksel hareket tahmini (Motion Estimaion-ME) yöntemini uygulamışlardır. He ve arkadaşları 2000'de doğrudan piksellerin bit genişliğini kısaltmışlardır. Pikseller 4-bitten daha fazla ifade edildiği sürece resim kalitesinde ciddi kayıplar yaşanmamaktadır. Piksel kısaltması donanımın kompleksliğini ciddi manada azaltmakta ve donanım üzerinde harcanan gücü azaltmaktadır. Sabit uzunluktaki kısaltma işlemi donanım boyutunu ve harcanan gücü azaltmasına rağmen çok fazla nitelik kaybına sebep olmaktadır. En düşük değerlikli bitleri sıfır yaparak donanım üzerideki alandan bir kazanç olmasa da nitelik kaybı olmadan donanım üzerinde harcanan güçten büyük ölçüde tasarruf edilmektedir.

2.2.2.4. Tahminlere göre tarama

Nesnelerin hızlı hareket ettiği video görüntülerinde yukarıda bahsedilen hızlandırma teknikleri uygulanır ise tahmini bölgedeki sürekli bozunmadan dolayı bu tekniklerin başarı oranı düşük olmaktadır. Arama bölgelerinde yapılacak örnek seyreltmeye dayalı algoritmalar genel olarak düşük başarı ile sonuçlanmaktadır. Hareketi tahmin eden algoritmalarda konumsal ya da geçici bloklardaki hareket bilgilerini faydalı bir şekilde kullanarak tarama yapılacak alan ve işlem yükü önemli ölçüde azalmaktadır (Chalidabhongse ve Kuo 1997)

2.2.2.5. Hiyerarşik tarama

Çoklu çözünürlü (Multiresolution) aynı zamanda piramid yapı (Pyramid Structure) olarak bilenen yapılar, görüntü işleme uygulamaları için çok güçlü sayısal konfigürasyonlardır. FFS'in işlemsel yükünü azaltmak için yaygın bir şekilde piramid yapılar kullanılmaktadır. Çoklu çözünürlü yapılarda ilk turda tahmin kabaca yapılarak, sonraki turlarda tahminin hassasiyeti artırılmaktadır. Genellikle iki ya da üç seviyeli Hiyerarşik Tarama yöntemi uygulanmaktadır (Lee v Lee 2004). Tahmin hassasiyetinin sağlandığı seviyelerde tarama aralığı orjinal tarama aralığından çok daha küçüktür. Çok

daha yüksek seviyelere çıkılarak tarama işlemleri yapılabilmesine rağmen bilgi içeren detaylar kaybolacağı için hata olasılığı da artmaktadır. Çok büyük çerçevelerde alanlarda çoklu çözünürlü tekniği blok eşleştirme algoritmaları için en faydalı tekniklerden bir tanesidir.

2.2.2.6. Hızlı-tüm tarama

Hızlı-Tüm Tarama şu şekilde gerçekleşmektedir. İlk turda aday bloğun optimum olup olmadığı kontrol edilir. Daha sonra, sadece potansiyel aday bloklar daha detaylı bozunum hesaplaması için kullanılırlar. Bu sayede uygun olmayan bloklar elenerek gereksiz işlemlerden kurtulmuş olunur. Örneğin Başarılı Eleme Algoritması (Successive Elimination Algorithm-SEA) (Li ve Salari 1995) aday bloklar ile taramanın yapıldığı blok arasındaki piksellerin farklarının mutlak değerlerinin toplamlarını daha önceden belirlenen SAD_{min} değeri ile karşılaştırarak gereksiz bloklardan kurtulmaktadır. Eğer ki bulunan SAD değeri SAD_{min} değerinden daha küçük ise SAD_{min} değeri güncellenerek taramaya devam edilir. Taramanın yapıldığı bloktaki piksellerin toplamı sadece bir kere yapılarak aday bloklarla yapılacak çıkarma işleminde kullanılır.

3. MATERYAL ve METOT

3.1. Gerçek Zamanlı Görüntü İşleme

Gerçek zamanlı sistem, belirli bir zaman içinde meydana gelen olaylara cevap verebilen sistemlerdir. Aksi takdirde sistem başarısız olarak değerlendirilir (Dougherty ve Laplante 1995). Görüntü işleme tarafında ise; gerçek zamanlı bir görüntüleme sistemi düzenli olarak görüntüyü yakalayan, bu görüntüleri veri elde etmek için analiz eden ve bu verileri bazı olayları kontrol etmek için kullanan sistemlerdir. Tüm bu işlemler önceden tanımlanan kesin bir zaman içinde meydana gelmelidir. Gerçek zamanlı görüntü işleme sistemleri ile ilgili örnekler oldukça fazladır. Görüntü işleme algoritmaları makine görüntüleme sistemlerinde denetleme veya işlem kontrolü için de kullanılmaktadır. Görüntü aktarma sistemlerinde ise son videoda oluşacak kalite kaybından kaçınmak için art arda gelen resimler doğru bir sırayla ve minimum gecikme ile iletilmelidir.

Gerçek zamanlı sistemler, zor gerçek zamanlı (Hard Real Time) ve kolay gerçek zamanlı (Soft Real Time) olmak üzere 2 kategoriye ayrılmışlardır. Zor gerçek zamanlı bir sistem, çıkışın gereken zamanda üretilmediği durumda tüm sistemin başarısız olarak değerlendirildiği sistemdir. Taşıma bandındaki nesneleri sınıflandırmak için yapılan görüntüleme sistemi buna bir örnektir. Nesnenin sınıfına göre hangi banttan gideceğinin kararının verilmesi, nesne ayrım noktasına gelmeden önce yapılmalıdır. Eğer karar bu zamana kadar yapılamaz ise sistem başarısız olur. Bunun yanı sıra, kolay gerçek zamanlı sistemde ise çıkış son ana kadar karar verilmese de tüm sistem başarısız olarak değerlendirilmez ama sistem performansı düşer. İnternet üzerinden video aktarımı ise buna bir örnektir. Eğer bir sonraki resim gecikir veya zamanında çözülemez ise aktarılan videonun kalitesi düşer. Bunun gibi bir sistem kolay gerçek zamanlı sistemdir, çünkü çıkışın gerekli zamanda üretilememesine rağmen çıkış hala üretilebilir ve tüm sistem başarısız olmaz.

Sinyal işleme perspektifinden, gerçek zamanlı görüntü işleme bir sonraki örneğin gelmesinden önce eldeki örneğin işlenip tamamlanması demektir. Video işlemede bu, her pikselin toplam işlemin piksel örnekleme zamanından önce tamamlanması demektir. Elbetteki bu, tek bir pikselin tüm işlemlerinin tamamlanması bir sonraki pikselin gelmesinden önce tamamlanacaktır demek değildir. Çünkü bir çok görüntü işleme işlemi her bir çıkış pikselinden veriye ihtiyaç duymaktadır. Ancak bu, geçici olarak tutulan piksellerin sonradan kullanılmasından dolayı, ortalama işlem oranına bir sınır getirmektedir (Kehtarnavaz ve Gamadia 2006).

Görüntüleme sistemini senkron hale getirmek, istenilen tepki zamanını elde etmeyi garantilemek için bir yaklaşımdır. Girişler düzenli ve belirli bir zamanda oluşursa bu sisteme uygundur. Fakat, olaylar rastgele ortaya çıkarsa; özellikle olaylar arasındaki minimum süre, her olayın işlem zamanı için gereken süreden daha az ise senkronize sistemler güvenli bir şekilde kullanılamaz.

Olaylar arasındaki süre, gereken tepki zamanından önemli bir derecede küçük olabilir. Bununla ilgili genel bir örnek ise konveyör temelli denetleme sistemleridir. Konveyördeki nesneler arasındaki zaman, denetim ve ayrım için olan toplam zamandan küçük olabilir. Bu sorunu çözmek için iki yöntem vardır. Birincisi, gerçekleşecek tüm

işlemleri, ardışık nesnelerin ayrım noktasına ulaşana kadar geçen zaman süresince durdurmak. Bu etkili bir şekilde daha sıkı gerçek zamanlı sınırlama sağlar. Bu sınırlama yönteminin başarılı olmadığı durumda, orijinal zaman kısıtlamasını değiştirmek için dağınık veya paralel işleme yöntemini kullanmak alternatif bir yöntemdir. Ancak uygulamayı birkaç işlemci üstünde yapmak gerekir. Bu yöntem zaman kısıtlamasını istenilen şekilde olmasını ve üretilen işin istenilen hız oranında gerçekleştirilmesini sağlayabilir.

3.1.1. Seri görüntü işleme

Geleneksel görüntü işleme platformları seri bilgisayar mimarisine dayanmaktadır. Bu temel formda mimari, tüm işlemleri aritmetik ve lojik işlemlere bölerek ALU (Arithmetic Logic Unit) tarafından seri bir şekilde işlenmesini sağlamaktadır. CPU (Central Processing Unit)'nun geri kalan kısmı ise ALU'yu besleyecek verileri sağlayacak şekilde dizayn edilir. Algoritma her saat darbesinde ALU ve CPU tarafından yapılacak işlemleri kontrol edecek şekilde komut dizinlerine dönüştürülür. Böylece bir CPU'nun temel görevi, bu komutları hafızadan alıp getirmek, komutu yapılması gereken işe dönüştürmek için çözmek ve komutu çalıştırmak olacaktır. Bilgisayar mimarisinde yapılan geliştirmelerin bir çoğu bu hafıza ve ALU arasındaki bağlantıdan geçirilen veri miktarını artırmaya yönelik olmuştur (Backus 1978).

En aşikar yaklaşım saat hızını artırarak komutları daha hızlı işleyebilmektir. Bu alanda son yıllarda çok büyük gelişmeler elde edilmiş olup saat hızları Ghz mertebelerine çıkmıştır. Bu gelişimin en büyük sebebi, yarı iletken teknolojisindeki boyutta ve besleme gerilimlerinde elde edilen gelişmelerdir. Bir diğer yaklaşım ise ALU veri genişliğini büyütmektir. Böylece her saat darbesinde daha fazla data işlenebilmektedir. Bu kelime genişliği daha büyük olan datalar için ciddi bir avantaj sağlamaktadır (örneğin gerçek sayılar). Bu sayede her bir data daha az saat darbesinde yüklenip işlenebilmektedir. Ancak görüntü işlemedeki datalar gibi kelime uzunluğu küçük olan datalar için hafıza bant genişliği limit faktör olmadıkça performans artışı çok iyi denilemez. Data yolu eğer kelime uzunluğundan daha küçükse, ALU'yu vektör işlemcisi olarak dizayn etmek de mümkündür.

Birden fazla datayı tek bir işlemci kelimesine ekleyerek, ALU'nun bu dataları eş zamanlı olarak işlemesine imkan sağlanmaktadır (Örneğin Intel MMX komutlarının kullanılması) (Peleg vd 1997). Son zamanlarda çoklu çekirdekli yapılar yaygın hale gelmiştir. Bu sayede bir uygulama içinde yer alan birden çok içerik farklı işlemci çekirdeklerinde paralel olarak çalışabiliyor (Geer 2005). Bu durumda eğer uygulama birden çok içeriği destekleyecek şekilde geliştirilebilirse görüntü işleme uygulamalarında bazı iyileşmeler sağlanabilir. Ancak yeterli dikkat gösterilmezse bellek bant genişliği görüntüye erişimde karşımıza engel olarak çıkabilir.

Son gelişmelerden bir tanesi de özellikle grafikle ilgili işlemlere tahsis edilmiş ve son teknoloji video oyunlarında öncelikle kullanılan bir işlemci olan GPU (Graphic-Processing-Unit, Grafik-İşleme-Birimi) ile ilgili olmuştur. GPU tarafından gerçekleştirilen esas işlem sahne üstündeki üçgen yamaları meydana getiren köşe verilerini alarak bunlara karşılık gelen çıkış piksellerini üretmek ve bir çerçeve

arabelleğinde tutmaktır. Gerçekleştirilen işlemler arasında doku kaplama, piksel gölgeleme, z-arabelleğe alma, karıştırma ve örtüşme önleme yer almaktadır. İlk olarak geliştirilen GPU'larda bu aşamaların her biri için ayrılmış ardışık veri blokları mevcuttu. Ancak bu durumda daha geniş uygulamalar için kullanım kısıtlanıyordu. Daha sonra geliştirilen cihazlar programlanabilir hale getirilmiştir. Bu sayede görüntü işlemede (Cope vd 2005) veya başka hesap yükü ağır işlemlerde (Manocha, 2005) kullanılabilecek duruma gelmişlerdir. Ardışık veri bloklarının hafif donanımlı çok içerikli kullanım ile harmanlanmasıyla birlikte işlem hızı daha da artırılmıştır (NVIDIA, 2006).

3.1.2. Paralel görüntü işleme

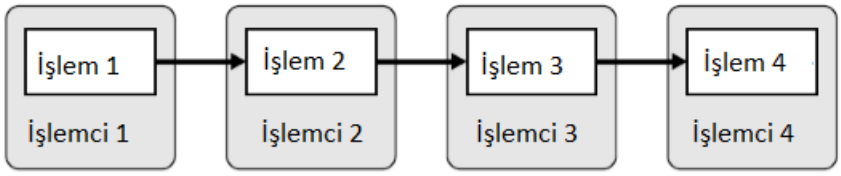
Temel olarak algoritmada yer alan her bir adım ayrı bir işlemcide gerçekleştirilerek tam paralel bir uygulama sağlanabilmektedir. Ancak algoritma çoğu yerde algoritma içindeki her adım önceki adımdan gelen verilere bağlı olacak şekilde ardışık ilerliyorsa yanıt süresini azaltmak adına çok az gelişme kaydedilebilir. Paralel uygulamada daha pratik çözüm geliştirmek adına algoritmada paralel olarak uygulanacak belli sayıda adım olmalıdır. Bu uygulamaya Amdahl kanunu (Amdahl 1967) denilir. Diyelim ki s algoritmanın seri halde çalışacak parçası (program önişlemleri ve diğer ardışık bileşenler) ve p algoritmanın N tane işlemci üzerinden paralel halde çalışacak parçası olsun. Bu durumda elde edilebilecek en iyi işlem hızı aşağıdaki gibidir:

$$Hızlandırma \leq \frac{s+p}{s+\frac{p}{N}} = \frac{N}{1+(N-1)s} \tag{10}$$

Bu denklem ancak paralellik sonucunda ek işletim yükü (örneğin haberleşme veya diğer program önişlemleri) söz konusu değilse doğru olacaktır. Bu işlem hızı paralel işlemcilerin işlem hızından mutlaka daha düşük olup algoritmanın seri olarak gerçekleştirilecek parçası sayesinde sınırlandırılacaktır:

$$Limit_{n\to\infty}\ Hızlandırma = \frac{1}{s} \tag{11}$$

Bu durumda işlem hızında kayda değer bir artış sağlamak için algoritmanın paralel olarak gerçekleştirilecek parçası da önem kazanmaktadır. Görüntü işlemenin de özellikle işlem piramidinin düşük ve orta seviyelerinde paralel olması bu durumda avantaj sağlamaktadır. Bu paralellik kendini birkaç şekilde göstermektedir. Görüntü işleme algoritmaları görüntü üstünde yapılan bir dizi işlemden meydana gelmektedir. Bu işlemler zamanda paralellik sağlar. Böylesi bir yapı Şekil 3.1'de gösterildiği gibi her işlem için ayrı bir işlemci kullanılarak sağlanabilir. Bu ardışık düzenli veri bloğu mimarisidir. Bu mimari verilerin işlemden geçerken her aşamadan geçirildiği bir üretim hattına benzetilebilir. Her işlemci kendine ayrılan işlemleri gerçekleştirir ve elde edilen sonucu sonraki aşamaya geçirir. Her bir işlemci kendinden önce gelen işlemci işlemi bitirene kadar beklemek zorunda olduğundan toplam işlem süresi (ya da yanıt süresi) azalmayacaktır. Fakat birinci işlemci ikinci işlemi gerçekleştirirken ikinci işlemci birinci işlemden gelen ilk işlemi gerçekleştirebileceğinden veri işlem hacminde artış sağlanabilir.



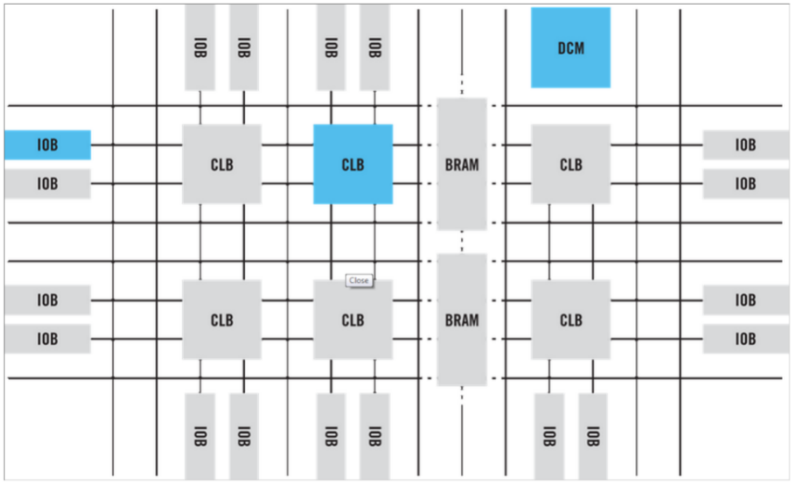
Şekil 3.1. Ardışık Düzenli Veri Bloğu Mimarisi

Görüntü işlenirken daha görüntünün tümü işlenmeden çok önce herhangi bir işlemden gelen veriler başka bir işlemin çıktısı olabilir. Verinin bir işlemin girdisi olduğu an ile bundan gelecek çıktının elde edilmesi arasındaki zaman farkına gecikme süresi denilir. Bu gecikme süresi her işlemde küçük bir yerel komşuluktan alınan giriş piksel değerleri kullanıldığında her çıkış için az sayıda giriş piksel değeri gerekeceğinden en aza indirgenir. Çıkış piksel değerini hesaplamak için bütün görüntünün işlenmesini gerektiren işlemlerde gecikme süresi daha fazla olur. İşlemlerin eş zamanlı yapılabilmesi her işlemin gecikme süresi düşük olduğunda üst işlemci işlemi bitirmeden art işlemci işleme başlayabileceğinden performansta önemli iyileşmelere sebep olabilmektedir. Bu çeşit bir avantaj ancak çok işlemcili sistemlerde sağlanabilir. Bunun yanında her işlem için ayrı bir bellek dizisinin kullanılması sayesinde kullanıcının bütün görüntü işlenmeden önce sonuçları elde edebildiği yazılım tabanlı sistemlerde de işe yarayabilir (McLaughlin 2000). Tabi ki tek çekirdekli bir yazılım sisteminde önbellekte yer olmadığında yavaş bir harici bellekten verilerin alınması beklenirken başka bir bellek dizisine geçiş yapılarak bazı iyileştirmeler sağlansa da toplam yanıt süresi azaltılamaz. Ancak donanım sistemlerinde toplam yanıt süresi her aşamanın gecikme süresi ile bütün görüntünün giriş olarak alınması için geçen sürenin toplamıdır. Her bir gecikme süresi görüntüyü yüklemek için gereken süreden kısaysa veri işlem hattındaki işlemci sayısına göre işlem hızlandırma önemli hale gelebilir.

3.2. FPGA Nedir?

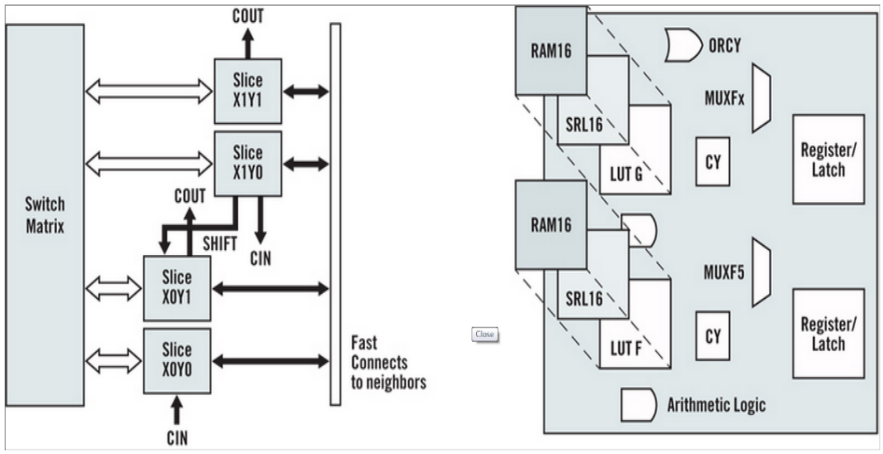
Alan Programlanabilir Kapı Dizileri (FPGAs), matris halindeki konfigüre edilebilir lojik blokların (CLBs) programlanabilir bir bağlantı ile birbirlerine bağlanan yarı iletken aygıtlardır. FPGA'lar üretildikten sonra istenilen uygulamaya ya da fonksiyona göre tekrardan programlanabilirler. Bu özellik FPGA'ları sadece spesifik bir işi yapan Uygulama Özellikli Entegre devrelerden (ASICs) ayırmaktadır. Modern FPGA'lar kompleks hesaplamaları ve dijital uygulamaları gerçeklemek için çok sayıda lojik kapılara ve RAM lere sahiptirler. Şekil 3.2'deki gibi gösterilen FPGA tasarımları çok hızlı ve çift yönlü I/O lara sahip olduklarından doğru zamanlamayı doğru data için yapmanın zorluk seviyesi oldukça yüksektir. Katman planlayıcısı (floor planning) sayesinde bu zamanlama sorunu ortadan kaldırılmaktadır.

Bazı FPGA'lar ise dijital özelliklerinn yanında analog özellikleri de barındırmaktadır. En bilindik analog özellik, her çıkış pinindeki programlanabilir yetişme hızı (slew rate) ve sürüş gücü (drive strength)'dür. Bir diğer sık kullanılan analog özellik ise diferansiyel sinyal kanallarının giriş pinlerinde birbirlerine bağlı olmasını sağlayan diferansiyel karşılaştırıcılardır.



Şekil 3.2. FPGA Blok Yapısı

Şekil 3.3'te gösterildiği gibi FPGA içindeki en temel yapılar Konfigüre Edilebilir Lojik Bloklar (Configurable Logic Blocks - CLB)'dır. CLB sayıları ve özellikleri FPGA türüne göre değişiklik göstermektedir. Ancak her CLB'de 4 ya da 6 girişi olan konfigüre edilebilir anahtar matrisler, bazı seçici devreleri (MUX ve benzeri) ve flip floplar bulunmaktadır. Anahtar matrisler çok esnek yapılarda olup; ardışıl devreler (combinational logic), kaydırma saklayıcıları (shift register) ya da RAM yapılarını çok rahat gerçekleyebilecek şekilde konfigüre edilebilmektedirler.



Şekil 3.3. CLB Blok Yapısı

3.2.1. FPGA ve görüntü işleme

FPGA'lar her fonksiyon için ayrı donanım gerektiren bir uygulama üzerine bir lojik kullandığından dolayı paraleldirler. Bu sayede bir yandan donanım tabanlı tasarımın hızını kazanırken diğer yandan da yeniden programlanabilme esnekliğiyle daha düşük maliyet sağlanabilmektedir. Bu şekliyle FPGA'lar, görüntü işleme için özellikle görüntülerde paralel işlem yapma özelliğinin önem kazandığı düşük ve orta seviyeli işlemlerde daha uygundur.

Ardışık veri bloklu mimaride ardışık veri bloğunun içinde yer alan her görüntü işleme işlemi için bir donanım parçası ayrılmıştır. Veri senkronize bir sistemde bir işlemin çıkışından bir sonraki işlemin girişine aktarılır. Sistem senkronize değilse veri akışında ve erişim modellerinde varyasyonlar sağlamak için işlemler arasına uygun arabellekler yerleştirilebilir.

Görüntü işlemede lojik paralellik FPGA uygulamaları için uygun olup birçok görüntü işleme algoritması bu yöntemle önemli ölçüde hızlandırılabilir. Bu yöntemde iç döngüler paylaştırılarak işlemler sırayla yapılmaktansa paralel donanım kullanılmaktadır. Duraksız işlemle görüntü verisi tek bir fonksiyon bloğuyla seri olarak beslenmektedir. Bu durum donanım uygulaması için özellikle görüntünün doğal akışının sağlandığı bir kamera veya ekrana doğrudan arabağlantı yapıldığında uygun olmaktadır. Tüm işlemler akış işleme kullanılarak yapılabiliyorsa bütün algoritmanın tek akışlı ardışık veri bloğu olarak kullanılması sonucunda çok verimli bir uygulama elde edilmektedir. Akış işleme kullanıldığında gerekli veri işlem hacmini sağlamak için ardışık veri bloğu gerekmektedir.

Paralellik kullanımı gömülü görüntü sistemlerinde oldukça işe yaramaktadır. Birden çok işlem birbirine paralel yapılarak çalışma hızı önemli ölçüde azaltılabilmektedir. Bir kameradan saniyede 30 çerçeve hızla alınan bir VGA çözünürlüklü video akışı saniyede yaklaşık 10 milyon piksel üretmektedir (ancak çalışma hızı boşluk zamanlarına bağlı olarak genellikle daha iyidir). Ciddi bir görüntü işlemede her piksel için çok sayıda işlem gerçekleştirilir. Bu durumda alışıla gelmiş seri işlemcilerin çalışma frekansı çok daha yüksek olmalıdır. Sistemin dinamik güç tüketimi doğrudan çalışma frekansına bağlıdır. Bu yüzden daha düşük çalışma hızı çok daha düşük güçlü tasarıma sebep olmaktadır.

Bütün algoritma bir FPGA üstünde uygulanabilirse sonuçta elde edilen sistemin küçük bir biçim katsayısı olacaktır. Sadece iki veya üç mikroçip kullanarak tasarımlar yapılabilir. Bu sayede bütün görüntü işleme sistemi sensör içine gömülü hale getirilebilir. Bu şekilde sistemin beynini kendi içinde barındıran akıllı sensörler ve akıllı kameralar geliştirilebilmektedir (Mosqueron vd 2007). Sonuçta görüntü işleme mekanizması çok yönlü bir sensör olarak birçok uygulama içinde gömülü olarak kullanılabilmektedir.

3.3. Matlab İle Görüntü İşleme

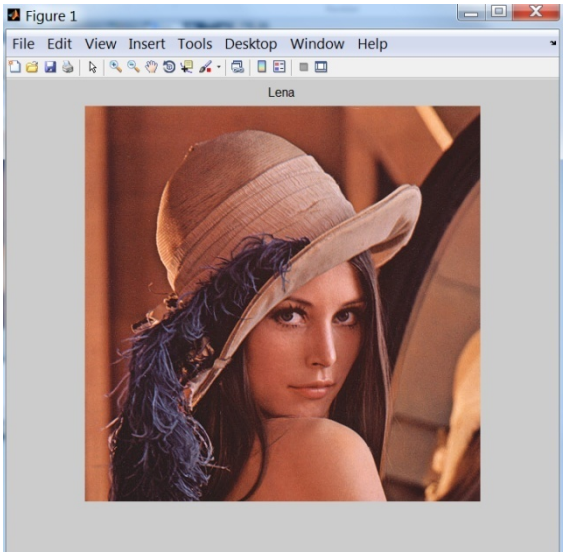
Matlab çok geniş kütüphanesi ve uygulama örnekleri sayesinde görüntü işleme uygulamaları için çok uygun bir platform niteliğindedir. Matlab kendi kütüphanesinde bulunan hazır fonksiyonlar sayesinde algoritmayı hızlıca geliştirme ve testlerini yapma aşamasında kullanıcılara büyük kolaylıklar sağlamaktadır. Öyle ki matlabın kütüphanesi literatürdeki ilerlemeye bağlı olarak sürekli olarak güncellenmekte buda kullanıcının işini kolaylaştırmaktadır. Günümüzde gerçek zamanlı uygulamalarda kullanılan kameraların yüksek hızda ve yüksek çözünürlükte olmasından dolayı matlab kodları ile çalışan gerçek zamanlı uygulamalar yapmak çok kolay olmamakla birlikte daha çok algoritma geliştirme ve analizler için kullanılmaktadır.

3.3.1. Temel matlab fonksiyonları ve uygulamaları

Matlab kodları ile algoritma geliştirmek için bazı temel fonksiyonlara ve matlab arayüzüne kısaca değinmekte fayda vardır. Matlab ile algoritma geliştirilirken ilk olarak algoritmanın uygulanacağı çerçeve, okuma fonksiyonu ile okunur. Bu işlem şu şekilde olmaktadır:

```
t = imread('Leana.bmp','bmp');
imshow(t)
title('Lena')
```

Kodun bulunduğu klasördeki template.jpg dosyası Matlab'ın imread fonksiyonu ile okutularak t isimle matriste bu resmin piksel değerleri saklanmaktadır. Şekil 3.4'teki Lena.bmp görüntüsü 512x512 boyutunda renkli bir görüntü olup t matrisininin boyutu 512x512x3 'tür.

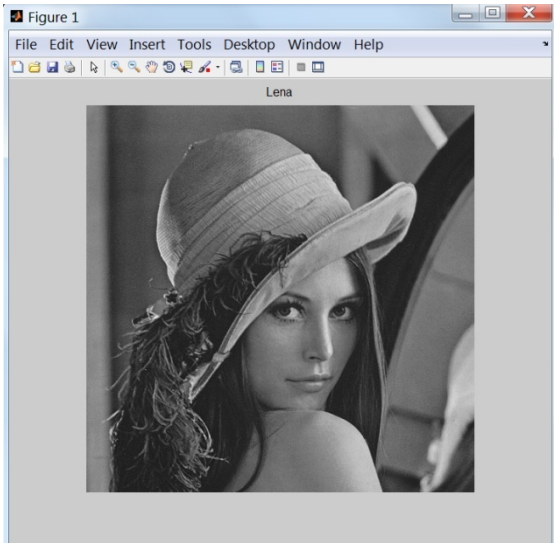


Şekil 3.4. Örnek Resim('Lena.bmp')

Kameradan gelen görüntüler renkli ya da gri formatta görüntü olabileceği gibi görüntü işleme algoritmalarında resimdeki renk değerleri kullanılmayacaksa resim gri formatta görüntüye dönüştürülür. Bu işlem Matlab kodları ile şu şekilde yapılmaktadır:

```
t1 = imread('Lena.bmp','bmp');
t2 = rgb2gray(t1)
imshow(t2)
title('Lena')
```

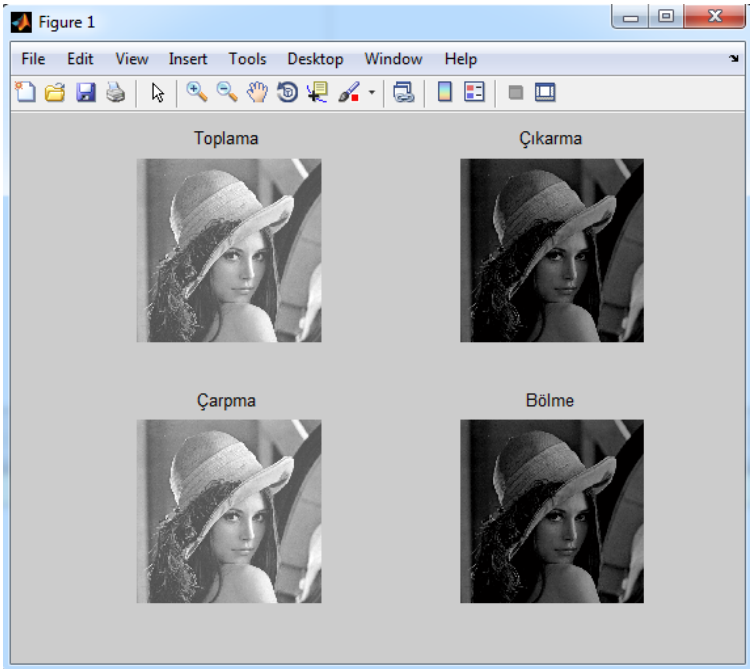
Şekil 3.5'ten görüldüğü gibi resim gri formatta görüntüye dönüşmüştür. Renkli resimin tutulduğu t1 matrisinin boyutu 512x512x3 iken gri formattaki görüntünün tutulduğu t2 matrisinin boyutu ise 512x512 olacaktır. Her bir piksel 8 bit ile ifade edilmek üzere 0 siyahı ve 255 beyazı ifade etmektedir. Bu durumda her bir pikselin alabileceği değer 0 ile 255 arasında değişmektedir.



Şekil 3.5. Gri Formatta Örnek Resim

Görüntü iki boyutlu matrisler halinde ifade edildikten sonra, bu matrisler üzerinde istenilen aritmetik işlemler yapılmaktadır. Örneğin Şekil 3.5 'teki resmin her bir pikseline 50 değerini eklersek, piksel değerleri 255'e daha çok yakalaşacağı için resmin parlaklığını artırmış oluruz. Tersine her pikselden 50 değerini çıkarırsak parlaklığı azaltmış oluruz. Benzer şekilde çarpma ve bölme işlemleri de sırasıyla parlaklığı artıracak ve azaltacaktır. Bu işlemler şu şekilde yapılmakta olup işlemlerin sonuçları Şekil 3.6'daki gibidir:

```
t1 = imread('lena.bmp','bmp');
t2 = rgb2gray(t1);
t3 = t2+50;
t4 = t2-50;
t5 = t2*2;
t6 = t2/2;

figure
subplot(2,2,1);
imshow(t3)
title('Toplama')
subplot(2,2,2);
imshow(t4)
title('Çıkarma')

subplot(2,2,3);
imshow(t3)
title('Çarpma')

subplot(2,2,4);
imshow(t4)
title('Bölme')
```



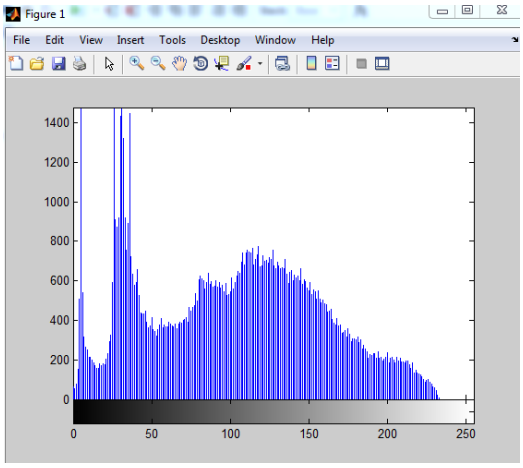
Şekil 3.6. Çerçeveler Üzerinde Aritmetik İşlemler

Bir diğer sık kullanılan fonksiyon ise eşik değeri hesaplaması ve köşe tespitidir. Bu tezdeki şablon eşleştirme yönteminde her ne kadar köse tespiti yapılmış görüntüler kullanımlasada, şablon eşleştirmede yaygın olarak kullanılmaktadır (Choi vd 2006).

Eşik değeri hesaplamasında gri formatındaki görüntünün histogram değerine bakılarak uygun eşik değeri ile görüntü siyah beyaz görüntü haline dönüştürülür. Bu işlem matlab kodları ile şu şekilde yapılmaktadır:

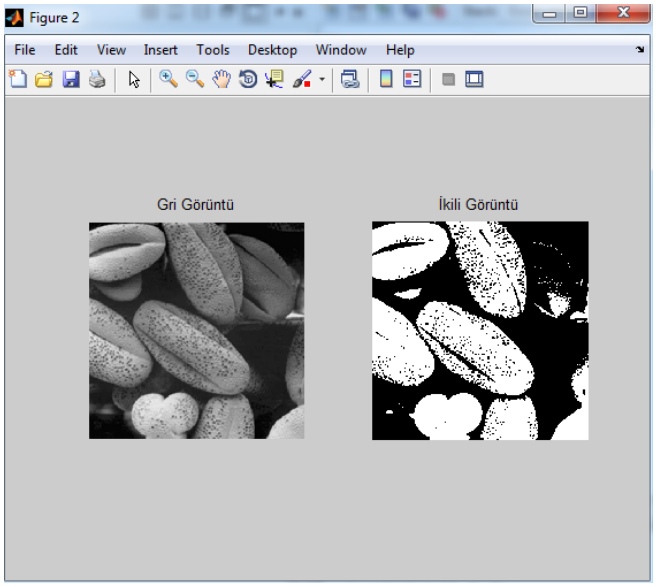
```
t = imread('Tohum.png','png');
t1 = rgb2gray(t);
imhist(t1)

figure
subplot(1,3,2);
imshow(t1)
title('Gri Görüntü')

subplot(1,3,3);
imshow(t1>100)
title('İkili Görüntü')
```

Şekil 3.7'deki histogram değerine bakılırsa tohum pikselleri yaklaşık olarak 75 ile 150 arasındadır. 100'den büyük olan değerleri 1 (beyaz) diğer piksel değerlerini 0 (siyah) yaparak görüntü siyah beyaz görüntü haline dönüştürülmüş olur.



Şekil 3.7. Histogram Eğrisi

Gri formatındaki görüntü ve bu görüntünün eşik değeri 100 ile oluşturulmuş ikili görüntüsü Şekil 3.8'deki gibidir.



Şekil 3.8. Gri Görüntüden Elde Edilmiş Siyah-Beyaz Görüntü

3.4. FFS İçin Önerilen İki Aşamalı Yöntem

Tüm çerçeve taraması işleminde şablon (T = nXn), eşleştirilmek istenilen çerçeve (S = mXm) ve $m > n$ olmak üzere T'nin S üzerindeki tüm piksellerde gezdirilmesi sonucu yapılması gereken eşleştirme sayısı: $(m-n+1)\mathrm{X}(m-n+1)$ 'dir. Her bir eşleştirmede işlenmesi gereken piksel sayısı, $n\mathrm{X}n$ olmak üzere; Tüm Çerçeve Eşleştirme Algoritması için işlenmesi gereken piksel sayısı denklem 9'daki gibi ifade edilmiştir. Denklem 9'a göre işlenmesi gereken piksel sayısı m ve n değerlerine göre değişmek üzere farklı görüntü boyutları ve şablon boyutları için işlenmesi gereken piksel sayısı Çizelge 3.1'de örnek olarak verilmiştir.

Çizelge 3.1. Farklı m ve n Değerleri İçin İşlenmesi Gereken Piksel Sayısı

S = mXm	T = nXn	İşlenmesi Gereken Piksel Sayısı
1024X1024	32X32	1.009.714.176
	64X64	3.782.742.016
	128X128	13.182.713.856
512X512	32X32	236.913.664
	64X64	825.757.696
	128X128	2.428.518.400
256X256	32X32	51.840.000
	64X64	152.571.904
	128X128	272.646.144

Şablon eşleştirme birçok uygulamada kullanılmak üzere eşleştirilmek istenilen şablonun boyutu, uygulama alanı ve ihtiyaçlara göre farklılık göstermektedir. Ancak yukarıdaki tablodan da görüldüğü gibi şablonun boyutu büyüdükçe işlenmesi gereken piksel sayısıda yaklaşık olarak şablon boyutundaki artış kadarartmaktadır. Kayıtlı resimler üzerinde şablon eşleştirme yöntemi ile eşleştirme işlemi yapmak günümüz bilgisayarları ile mümkün olsada, gerçek zamanlı bir uygulama yapılmak istenildiğinde FFS için hızlandırma yöntemleri ve paralel işleme metotları kullanılmaktadır (Kawanishi vd 2004).

Hızlandırma teknikleri detaylı bir şekilde anlatılmış olup, bizim önerdiğimiz hızlandırma tekniği Bölüm 2.2.2.1'deki arama noktalarını azaltma alt başlığındaki gruba dahil olmaktadır. Önerilen algoritmaya göre: Toplam işlem yükünü ve süresini azaltabilmek için, algoritma iki aşamalı hale getirilmiştir. İlk aşamada, birer piksel kaydırmak yerine, çift sayıda (2^k, k∈ $\mathbb{Z}$) piksel kaydırılarak kayma miktarı kabaca hesaplanmaktadır. İkinci aşamada ise, 2^k cinsinden bulunan kabaca kayma miktarı üzerinde, birer piksel kaydırma yapılarak kayma miktarı 1 piksel hassasiyet ile hesaplanmaktadır. Algoritma bu şekilde geliştirilerek birinci aşamada işlenmesi gereken piksel sayısı 9 'a göre şu şekilde hesaplanır :

$$\left(\frac{m-n}{2^k} + 1\right) \mathrm{X} \left(\frac{m-n}{2^k} + 1\right) \mathrm{X} n^2 \tag{12}$$

Denklem 12'de görüldüğü gibi işlenmesi gereken piksel sayısı k'ya bağlı olarak önemli ölçüde azalmaktadır. k sayısı artınca işlem sayısı azalmakla birlikte eşleşme doğruluğu da (r, c sayısının bulunması) azalmaktadır. Doğruluğu daha da artırmak için ilk hesaplamada bulunan SAD(r,c) değeri için satır ve sütunda birer piksel ilerleme yapılarak ikinci bir eşleştirme yapılır. Yapılan bu eşleştirme için $r_2 = r - 2^k + 1$, $c_2 = c - 2^k + 1$, r'= {0,1, …,2^{k+1} - 2} ve c' = {0,1,…,2^{k+1}- 2} olmak üzere genişletilmiş denklem şu şekildedir:

$$SAD(r', c') = \sum_{j=1}^{n} \sum_{i=1}^{n} \left| S(i + r_2 + r', j + c' + c_2) - T(i, j) \right| \tag{13}$$

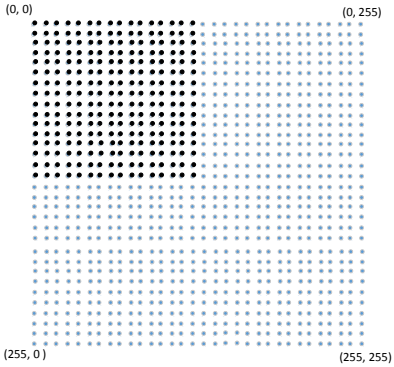
İkinci eşleştirme için işlenmesi gereken piksel sayısı şu şekildedir:

$$(2^{k+1} - 1)^2 \mathrm{X} n^2 \tag{14}$$

Denklem 12 ve 14'ten iki aşamalı eşleştirme için işlenmesi gereken toplam piksel sayısı 15'deki gibidir:

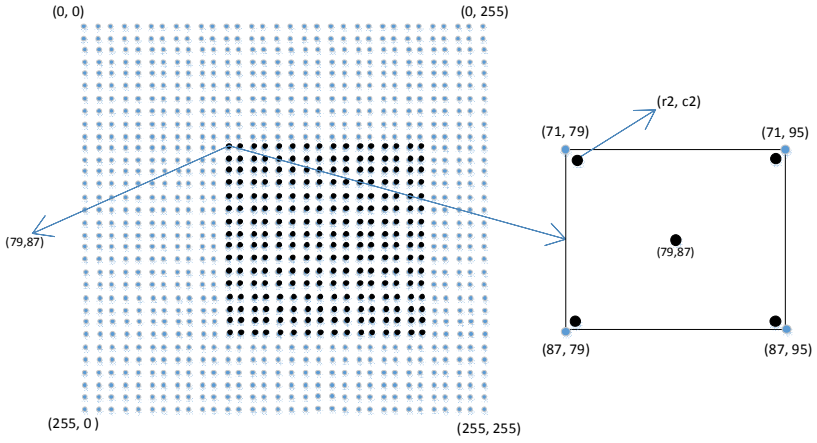
$$n^2 \left((2^{k+1} - 1)^2 + \left(\frac{m-n}{2^k} + 1 \right)^2 \right) \tag{15}$$

Görüldüğü gibi işlenmesi gereken piksel sayısı k'ya ve seçilen m, n değerlerine bağlı olarak oldukça azalmaktadır. Yukarıdaki bilgiler eşliğinde önerilen yöntem şu şekilde çalışmaktadır. Şablon T = 128X128, aramanın yapılacağı resim S = 256X256 ve k=3 için ilk turda 2^k = 8 piksel satır ve sütunda ilerleme yapılacaktır. Şekil 3.9'da her iki nokta arası 8 piksel olup, siyah noktalar T'yi mavi noktalar ise S'i ifade etmektedir. İlk turda satır ve sütunda 8 piksel ilerleme yapılarak ilk eşleşme bulunacaktır.



Şekil 3.9. Temsili S ve T Çerçeveleri

Şekil 3.10'da ilk turdaki eşleşmenin olduğu temsili bir nokta (79,87) gösterilmiştir.



Şekil 3.10. Temsili İlk Tur Eşleşmesi

Önerilen algoritmanın ilk turu için işlenmesi gereken piksel sayısı denklem 12'ye göre 4.734.976'tür. İlk turdaki eşleşme (79,87) noktasında olur ise denklem 13 için gerekli parametreler şu şekilde hesaplanır:

SAD(r,c) = (79,87) ==> (r,c) = (79,87),
$r_2 = r - 2^k + 1 = 79 - 8 - 1 = 72$,
$c_2 = c - 2^k + 1 = 87 - 8 - 1 = 80$,
r'= {0,1, …,2^{k+1} - 2} = r'= {0,1, …,14},
c' = {0,1,…,2^{k+1} - 2} = c' = {0,1,…,14},

Denklem 13 için gerekli parametreler bu şekilde hesaplandıktan sonra ikinci turdaki eşleşme 1 piksel hassasiyeti ile gerçeklenmiş olur. Önerilen algoritmanın ikinci aşamasında işlenmesi gereken piksel sayısı denklem 14'e göre 3.686.400'dür. Bu durumda toplam işlenmesi gereken piksel sayısı 8.421.376'dır. Çizelge 3.1'deki FFS algoritması ile karşılaştırılacak olursa önerilen yöntem sayesinde işlenmesi gereken piksel sayısı 32,3 kat azalmıştır. Önerilen yöntem için Çizelge 3.1'in genişletilmiş hali aşağıdaki gibidir.

Çizelge 3.2. FFS ile k=3 İçin Önerilen Algoritmanın Karşılaştırması

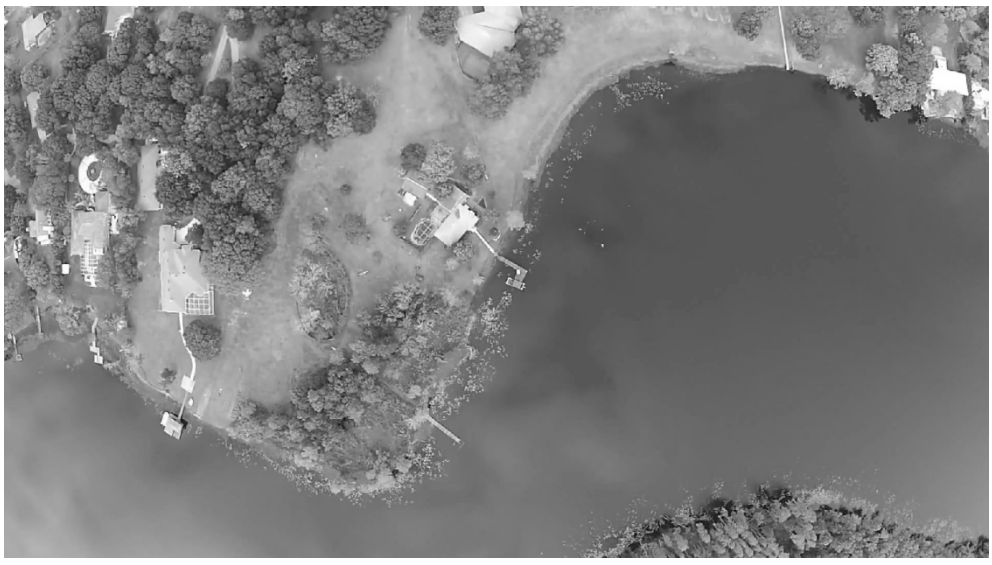
S = mXm	T = nXn	FFS ile İşlenmesi Gereken Piksel Sayısı	Önerilen Algoritman ile İşlenmesi Gereken Piksel Sayısı
1024X1024	32X32	1.009.714.176	16.230.400
	64X64	3.782.742.016	60.891.136
	128X128	13.182.713.856	212.893.696
512X512	32X32	236.913.664	4.040.704
	64X64	825.757.696	14.229.504
	128X128	2.428.518.400	43.024.384
256X256	32X32	51.840.000	1.091.584
	64X64	152.571.904	3.481.600
	128X128	272.646.144	8.421.376

4. BULGULAR ve TARTIŞMA

4.1. Önerilen Yöntemin Matlab ile Gerçeklenmesi ve Test Sonuçları

Tüm çerçeve tarama algoritması Bölüm 2'de bahsedilen SAD, MAD, SDD ve NCC algoritmaları kullanılarak gerçekleştirilmektedir. Bu tezde tüm çerçeve tarama algoritması ve önerilen yöntemin testleri İHA görüntüleri için denenmiş olup, parlaklık ve karmaşık arka plandan fazla etkilenmemesinden dolayı uygulama ve testlerde SAD algoritması kullanılmıştır. Önerilen yöntem genel bir yöntem olup farklı uygulama alanları ve görüntüler için MAD, SDD ve NCC içinde kullanılabilir. Şablon eşleştirme yöntemi İHA görüntülerinde kullanılarak eşleşme yapılmak istenildiğinde n. çerçeveden bir tane T şablon üretilir ve bu şablon (n+1), (n+2), ... kendisinden sonra gelen çerçevelerde eşleştirme için kullanılır.

Şekil 4.1'deki gibi 1920X1080 çözünürlükteki bir İHA görüntüsünün n. çerçevesinden üretilmiş Şekil 4.2'deki T = 128x128'lik görüntüsü bizim eşleştirmelerde kullanacağımız şablonu göstermektedir. Eşleştirme işlemi İHA görüntüsünün 1920X1080 çözünürlükteki (n+1). çerçevesinin tamamı üzerinde değilde, bu (n+1). çerçeveden üretilmiş S = 256X256 görüntüsü üzerinde SAD algoritması kullanılarak yapılacaktır.



Şekil 4.1. İHAGörüntüsü n. Çerçeve

Şekil 4.2. n. Çerçeveden Üretilmiş Şablon, T = 128X128

Matlab kodları ile önerilen yöntem kullanılmadan Şekil 4.2'deki T, Şekil 4.3'ten üretilmiş Şekil 4.4'teki S ile FFS algoritması (satır ve sütunda 1 piksel ilerleme yapılarak) kullanılarak eşleştirme yapılırsa eşleştirme sonuçları şu şekilde olmuştur:

SAD(r,c) = SAD(51,76) = 73.444

Şekil 4.3. İHA Görüntüsü (n+1). Çerçeve

Şekil 4.4. (n+1). Çerçeveden Üretilmiş S=256X256 Çerçevesi

Yapılan eşleştirmeye göre T çerçevesi S çerçevesinin üzerinde (51,76) koordinatlarında eşleşmiştir. Eşleşmenin olduğu noktadaki farkların mutlak değerlerinin toplamı 77.844 olarak bulunmuştur. Eşleşme noktasındaki SAD değerinde 128X128 = 16.384 tane çıkarma işlemi yapılmış olup, her iki pikselin ortalama farkı 73.444/16.384 = 4,4 olarak hesaplanmıştır. Şekil 4.2'deki T (n+2). çerçeveden üretilen S ile yine önerilen yöntem kullanılmadan FFS algoritması kullanılarak yapılan eşleştirme işleminin sonucu şu şekilde olmuştur:

SAD(r,c) = SAD(52,81) = 89.959

Önerilen algoritma kullanılmadan yapılan testlerin sonuçları Çizelge 4.1'de detaylı bir şekilde verilmiştir.

Çizelge 4.1. Önerilen Algoritma Kullanılmadan Yapılan Test Sonuçları

	T = 128X128 ve S= 256X256 Boyutlarındaki Çerçeveler için		
	S Üzerindeki Eşleşme (r,c)	SAD(r,c) Değeri	İki Piksel Arası Ortalama Fark
(n+1). Çerçeve	(51,76)	73.444	4,4
(n+2). Çerçeve	(52,81)	89.959	5,4
(n+3). Çerçeve	(53,87)	100.269	6,1
(n+4). Çerçeve	(54,92)	116.993	7,1

T çerçevesi ilk başta n. çerçeveden üretilmiş olup, sonradan hiç güncellenmemiştir. Tabloya bakacak olursak İHA'nin hareketlerini piksel kaymalarından tahmin edebiliriz. Tabloyu daha da genişletecek olursak (n+12). çerçeveden üretilen S ile n. çerçeveden üretilen T kullanılır ise eşleşme olmayacaktır (Kayıtlı çerçeveler üzerinden yapılan tesler sonucu). Bunun sebebi yeni hesaplanan (r,c) değerindeki r+128 ya da c+128 değeri 256 değerinden büyük olacağı için T'yi bu çerçevede bulmak mümkün olmayacaktır. Bu yüzden sistem gerçek zamanlı hale getirilmek istenilirse, uygun tur değerlerlerinde T çerçevesini güncellemek gerekecektir.

FFS algoritmasının satırda ve sütunda sadece 1 piksel kaydırma işlemi ile eşleştirme sonuçları Çizelge 4.1'de detaylı bir şekilde verilmiş olup, bir sonraki aşamada aynı çerçeveler için önerilen algoritmanın testleri yapılmıştır. Önerilen algoritma iki aşamalı olup, ilk aşamada satır ve sütunda 8 piksel ilerleme yapılmıştır. İlk aşamanın sonuçları da tabloda sütun olarak gösterilmiştir. İkinci aşamada ise 1 piksel hassasiyetle eşleştirme işlemi yapılmıştır.

Çizelge 4.2'ye bakılırsa önerilen yöntem için tüm testlerde Çizelge 4.1 ile aynı değerler elde edilmiştir. Testler sadece İHA görüntüleri için yapılmış olup, farklı görüntüler için de ayrıca testler yapılması gerekebilmektedir.

Çizelge 4.2. k=3 için Önerilen Algoritma ile Yapılan Test Sonuçları

	T = 128X128 ve S= 256X256 Boyutlarındaki Çerçeveler için		
	İlk Aşamada S Üzerindeki Eşleşme (r,c)	İkinci Aşamada S Üzerindeki Eşleşme (r,c)	Son Aşamada Hesaplanan SAD(r,c) Değeri
(n+1). Çerçeve	(48,72)	(51,76)	73.444
(n+2). Çerçeve	(48,80)	(52,81)	89.959
(n+3). Çerçeve	(56,88)	(53,87)	100.269
(n+4). Çerçeve	(56,96)	(54,92)	116.993

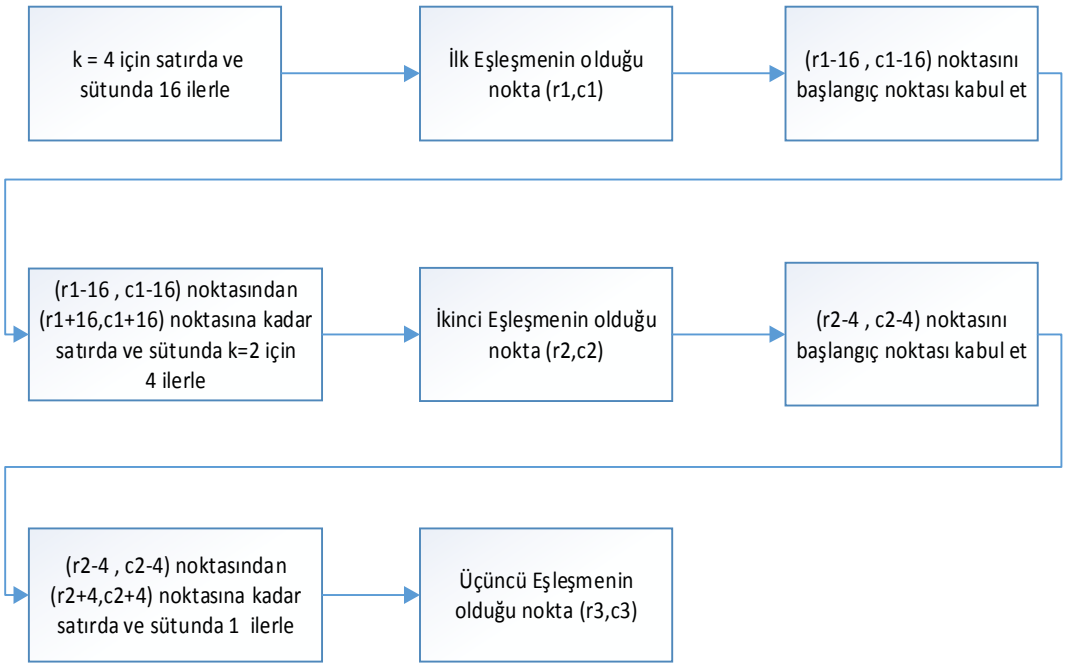
4.1.1. Önerilen yöntemin genişletilmesi ve sonuçları

Önerilen yöntem için testlerin hepsi Çizelge 4.2'deki k = 3 değeri için yapılmış olup farklı k değerleri için ayrıca testler yapmak gerekmektedir. k = 3 değeri için satır ve sütunda 8 piksel ilerleme yapılırken, k = 1, k = 2 değerleri için satır ve sütunda daha az ilerleme yapılacağı için yöntemin doğru çalışacağı şüphe gerektirmemektedir. Ayrıca k'yı belirlerken S = mXm ve T = nXn için (m-n) sayısı, 2^k sayısına kalansız bölünebilmeli ki T çerçevesi S üzerinde tam olarak gezdirilsin. Nitekim seçilen tüm parametreler (S ve T'nin boyutları, satırda ve sütunda ilerleme sayıları) 2^k kümsesinden seçildiği için n sayısı her seferinde 2^k sayısına kalansız bölünmektedir. Çizelge 4.1'de S = 256X256 ve T = 128X128 olduğu için k sayısı 1,2,3,4... sayılarını alabilmektedir. Ancak k sayısı büyüdükçe hassasiyet de azalacaktır. Çizelge 4.3'te k = 4 için önerilen algoritmanın test sonuçları verilmiştir:

Çizelge 4.3. k = 4 İçin Önerilen Algoritma ile Yapılan Test Sonuçları

	T = 128X128 ve S= 256X256 Boyutlarındaki Çerçeveler için		
	İlk Aşamada S Üzerindeki Eşleşme (r,c)	İkinci Aşamada S Üzerindeki Eşleşme (r,c)	Son Aşamada Hesaplanan SAD(r,c) Değeri
(n+1). Çerçeve	(48,80)	(51,76)	73.444
(n+2). Çerçeve	(48,80)	(52,81)	89.959
(n+3). Çerçeve	(48,80)	(53,77)	100.269
(n+4). Çerçeve	(48,96)	(54,92)	116.993

Çizelgeden görüldüğü gibi ikinci ve üçüncü sütundaki değerlerin hepsi Çizelge 4.2 ve 4.1 ile aynı değerlerdedir. Testlerde kullanılan S ve T çerçeveleri için k = 4 sayısı halen uygun bir sayı olarak gözükmektedir. Ancak k = 4 iken denklem 13'e göre işlenmesi gereken toplam piksel sayısı 17.072.128 olarak hesaplanmıştır. Her ne kadar k = 4 değeri için doğruluk halen yeterli olsa da bu seferde işlenmesi gereken piksel sayısı k = 3 için 8.421.376 iken bu sayı daha da artmıştır. Ayrıca k = 4 seçilerek algoritmanın hassasiyeti azaltılmıştır. n değeri, m değerinin yarısı kadar olup; yine n değeri, 2^k değerinin de 8 katı olduğu için doğru eşleşmeler elde edilmiştir. Ancak n değeri daha küçük bir değer olsaydı k'yı bu kadar büyük seçmek mümkün olmayacaktı. Tüm bu veriler ışığında S = 256X256 ve T = 128X128 değerleri için k = 3 sayısı en uygun değer olarak gözükmektedir. Hassasiyetten ödün vermeden seçilecek daha büyük k değerlerinde işlenmesi gereken piksel sayısını azaltmak için algoritma 3 aşamalı hale getirilebilir. k = 4 değeri için tam eşleşme olmasına rağmen işlenmesi gereken piksel sayısı k = 3'e göre daha fazla çıkmıştır. Seçilen tüm parametrelerin 2^n kümsesinden olması önerilen algoritmanın gerçeklenmesinde ve sonradan geliştirilmesinde bize çok büyük kolaylıklar sağlamış olup; farklı S, T ve k değerleri için önerilen yöntemin 2 aşamalıdan 3 aşamalı bir yönteme dönüşmesini mümkün kılmaktadır. Yine önerilen algoritma 3 aşamalı hale getirildiğinde, genişletilmiş algoritmanın son halinin FPGA üzerinde gerçeklemesi, donanım ve zaman maliyeti açısından düşük maliyetli bir uygulama olacaktır. Algoritma bu şekilde 3 aşamalı hale getirilirse bu işlem Şekil 4.5'teki gibi olacaktır.



Şekil 4.5. Önerilen Yöntemin Genişletilmesi

Algoritma bu şekilde genişletildikten sonra bulunan eşleme sonuçları Çizelge 4.4'teki gibi olmuştur. Çizelgeden görüldüğü gibi 3. sütun ile Çizelge 4.3'ün 2. sütunu birebir aynıdır. Bu da bize genişletilmiş yöntemin T = 128X128 ve S = 256X256 boyutlarındaki çerçeveler için kullanılabilir olduğunu göstermektedir.

Çizelge 4.4. Önerilen Yöntemin Genişletilmesi Sonucu Test Sonuçları

	T = 128X128 ve S= 256X256 Boyutlarındaki Çerçeveler için		
	İlk Aşamada S Üzerindeki Eşleşme (r,c)	İkinci Aşamada S Üzerindeki Eşleşme (r,c)	Üçüncü Aşamada S Üzerindeki Eşleşme (r,c)
(n+1). Çerçeve	(48,80)	(52,76)	(51,76)
(n+2). Çerçeve	(48,80)	(52,80)	(52,81)
(n+3). Çerçeve	(48,80)	(52,78)	(52,77)
(n+4). Çerçeve	(48,96)	(52,92)	(54,92)

Şekil 4.5'teki genişletilmiş yöntemde $k_1 = 4$ ve $k_2 = 2$ olmak üzere denklem 13'e göre işlenmesi gereken piksel sayısı 16'daki gibi olacaktır:

$$n^2\left((\frac{2*2^{k1}}{2^{k2}})^2 + \left(\frac{m-n}{2^{k1}} + 1\right)^2 + (2*2^{k2}-1)^2 \right) \quad (16)$$

Denklem 16'ya göre işlenmesi gereken piksel sayıları Çizelge 4.5'te verilmiştir. $k_1 = 4$ sayısı büyük bir sayı olduğu için sadece T = 128x128 değerleri için işlenmesi gereken

piksel sayıları tabloya eklenmiştir. Daha küçük boyuttaki T değerleri için farklı testler yapılması gerekmektedir.

Çizelge 4.5. Genişetilmiş 3 Aşamalı Yöntem için İşlenmesi Gereken Piksel Sayısı

S = mXm	T = nXn	Önerilen İki Aşamalı Yöntem için İşlenmesi Gereken Piksel Sayısı	Genişletilmiş Üç Aşamalı Yöntem için İşlenmesi Gereken Piksel Sayısı
1024X1024	128X128	212.893.696	55.083.008
512X512	128X128	43.024.384	12.091.392
256X256	128X128	8.421.376	3.178.496

Çizelge 4.5'den görüldüğü gibi yöntemin 3 aşamalı hale dönüştürülmesi ile birlikte tam eşleşme olmakla birlikte işlenmesi gereken piksel sayısı da azalmıştır. Bu ve bunun gibi örnekler farklı m ve n değerleri için çoğaltılabilmekle birlikte; hepsi için farklı testler yapmak gerekmektedir. Ancak çizelgelerdeki sonuçlardan önerilen yöntemin geliştirmeye açık olduğu düşünülmektedir.

4.2. Önerilen Yöntemin FPGA ile Hızlandırılması ve Sonuçları

Tüm çerçeve taraması uygulaması işlem yükü çok yüksek bir uygulama olup, bu tezde Bölüm 3.3'te önerilen yöntem FPGA bellek öbekleri (Block Ram) efektif bir şekilde kullanılarak, yöntemin daha da hızlı hale getirilmesi amaçlanmıştır. Günümüz modern FPGA'ları 550Mhz saat hızı seviyesinde çıkabilmekte ve bu hızda işlem yapabilmektedirler. FFS algoritması FPGA kullanılarak gerçeklenmek istenirse FPGA'nın birim zamanda ne kadar data işleyebileceğini çok iyi hesaplayıp, sistemi buna göre tasarlamak gerekmektedir. Örneğin Çizelge 3.1'deki veriler için kullanılacak kameranın 50fps olduğu ve her bir saat darbesinde bir piksel işlendiği düşünülürse ihtiyaç olan FPGA saat hızları Çizelge 4.6'daki gibi olacaktır.

Çizelge 4.6. Farklı m ve n Değerleri İçin Gerekli Olan FPGA Saat Hızları

S = mXm	T = nXn	FFS ile İşlenmesi Gereken Piksel Sayısı	FPGA Saat Hızı
1024X1024	32X32	1.009.714.176	50.485.708.800
	64X64	3.782.742.016	189.137.100.800
	128X128	13.182.713.856	659.135.692.800
512X512	32X32	236.913.664	11.845.683.200
	64X64	825.757.696	41.287.884.800
	128X128	2.428.518.400	121.425.920.000
256X256	32X32	51.840.000	2.592.000.000
	64X64	152.571.904	7.628.595.200
	128X128	272.646.144	13.632.307.200

Çizelgeden de görüldüğü gibi günümüz modern FPGA'larıyla bile FFS algoritmasını, hızlandırma tekniklerini kullanmadan gerçeklemek neredeyse imkansız gibi durmaktadır. Ancak ve ancak m ve n değerleri daha küçük seçildiği takdirde FPGA ile gerçek zamanlı bir sistem geliştirilebilmektedir. Önerilen yöntem FPGA ile

gerçeklenmek istenilirse ihtiyaç olan FPGA saat hızları ise Çizelge 4.7'deki gibi olacaktır.

Çizelge 4.7. Önerilen Algoritma İçin Gerekli Olan FPGA Saat Hızı

S = mXm	T = nXn	Önerilen Algoritman ile İşlenmesi Gereken Piksel Sayısı	FPGA Saat Hızı
1024X1024	32X32	16.230.400	811.520.000
	64X64	60.891.136	3.044.556.800
	128X128	212.893.696	10.644.684.800
512X512	32X32	4.040.704	202.035.200
	64X64	14.229.504	711.475.200
	128X128	43.024.384	2.151.219.200
256X256	32X32	1.091.584	54.579.200
	64X64	3.481.600	174.080.000
	128X128	8.421.376	421.068.800

Önerilen yöntem ile işlenmesi gereken piksel sayısı her ne kadar çok azalmış olsa da önerilen yöntemi FPGA ile gerçek zamanlı gerçeklemek istediğimizde FPGA saat hızı Çizelge 4.7'ye göre her zaman yeterli olmamaktadır. Çizelge 4.7'ye göre yalnızca S = 512X512 iken T = 32X32, S = 256X256 iken T = 32X32 ve S = 256X256 iken T = 32X32 değerleri için FPGA gerçek zamanlı bir sistem olarak kullanılabilmektedir. Tüm bu hesaplamalar FPGA her saat darbesinde bir piksel işlediği varsayılarak yapılmıştır. Ancak biz her saat darbesinde birden fazla piksel işleyebilirsek ihtiyaç olan FPGA saat hızını daha da azaltmış oluruz. Her saat darbesinde birden fazla piksel işlemek için FPGA'nın bellek öbeklerinin efektif bir şekilde kullanılması amaçlanmıştır. Bu tez boyunca FPGA bellek öbeklerinin efektif bir şekilde kullanılması ve gerçek zamanlı işlemlerin hepsi Şekil 4.6'daki S = 256x256 iken T = 128X128 örnekleri için yapılmıştır.

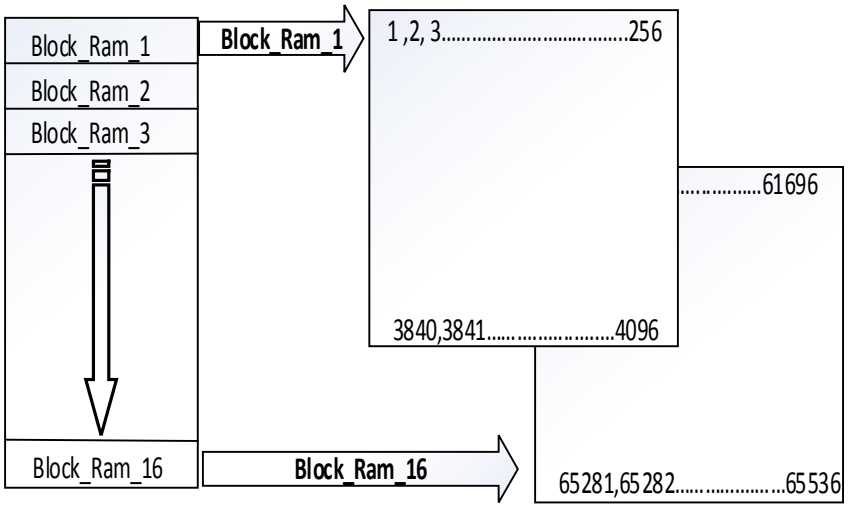
Şekil 4.6. S = 256x256 Çerçevesi(solda), T = 128x128 Çerçevesi(sağda)

FPGA'nın iç yapısında18K'lık bellek öbekleri bulunmaktadır. Bunların bir araya gelmesiyle de 36K'lık (8bit + İşaret biti = 9 X 1024 = 36K) bellek öbekleri FPGA'nın iç yapısında bulunmaktadır. Çizelge 4.8'de Virtex-5 FPGA ailesinde bulunan bellek öbeklerinin sayıları tablo olarak verilmiştir.

Çizelge 4.8. Virtex-5 FPGA Ailesi Bellek Öbekleri Sayısı

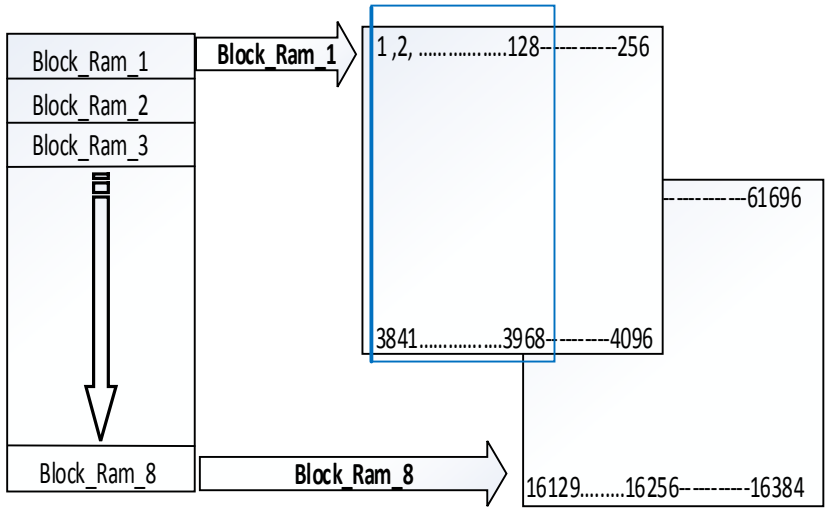
FPGA Türü	Konfigüre Edilebilir Lojik Bloklar		Bellek Öbekleri	
	Dizi (SatırxSütun)	Virtex-5 Slices	18 Kb	36 Kb
XC5VLX20T	60 x 26	3120	52	26
XC5VLX30T	80 x 30	4800	72	36
XC5VLX50T	120 x 30	7200	120	60
XC5VLX85T	120 x 54	12960	216	108
XC5VLX110T	160 x 54	17280	296	148
XC5VLX155T	160 x 76	24320	424	212
XC5VLX220T	160 x 108	34560	424	212
XC5VLX330T	240 x 108	51840	648	324

Günümüz modern FPGA'larında (Virtex 7, Kintex 7, Spartan 6) bu sayı çok daha fazla gelişmiş olup, bu bellek öbeklerini kullanarak çok farklı veri işlemleri yapılabilmektedir. Şekil 4.6 a'da S = 256x256 = 65536 piksel demektir. Her piksel 8 bit ise bu da 65536 B = 64 KB demektir. En düşük sayıda bellek öbeğiyle birim zamanda en fazla pikseli okuyabilmek için S çerçevesi 36K'lık bellek öbeklerini tam dolduracak biçimde yazılmıştır. S = 256x256 = 65536B olmak üzere toplam 16 adet bellek öbeğine (16x32Kb = 524288b = 65536B) yerleştirilmiştir. Bu durumda her bir bellek öbeğinde toplam 4096 adet S pikseli bulunmaktadır. S çerçevesinin bellek öbeklerine yerleştirilmesi Şekil 4.7'de temsili olarak gösterilmiştir.



Şekil 4.7. S Çerçevesinin Bellek Öbeklerine (Block RAMs) Yerleştirilmesi

T Çerçevesinin bellek öbeklerine yerleştirilmiş hali ise Şekil 4.8'deki gibidir. T çerçevesi 128x128 = 16384B olmak üzere toplam 8 adet bellek öbeğine yazılmıştır. Bu durumda her bir bellek öbeğinde 16384B/8 = 2048B = 2048 adet piksel bulunmaktadır. Her bir bellek öbeği 4096B olduğu için bellek öbeklerinin sadece yarısı kullanılmıştır. Bu tezdeki uygulamada T çerçevesinin kullanıldığı bellek öbeklerinin yarısı boş bırakılmış olup bu boş alanlar uygulamada başka işlemler içinde kullanılabilir.



Şekil 4.8. T Çerçevesinin Bellek Öbeklerine (Block RAMs)Yerleştirilmesi

T çerçevesinin bellek öbeklerine yazdırılması, t1 birinci bellek öbeğini ifade etmek üzere Algoritma 1'deki gibi olmaktadır (Aktaş vd 2015):

Algoritma 1: T Bellek Öbeklerinin Doldurulması

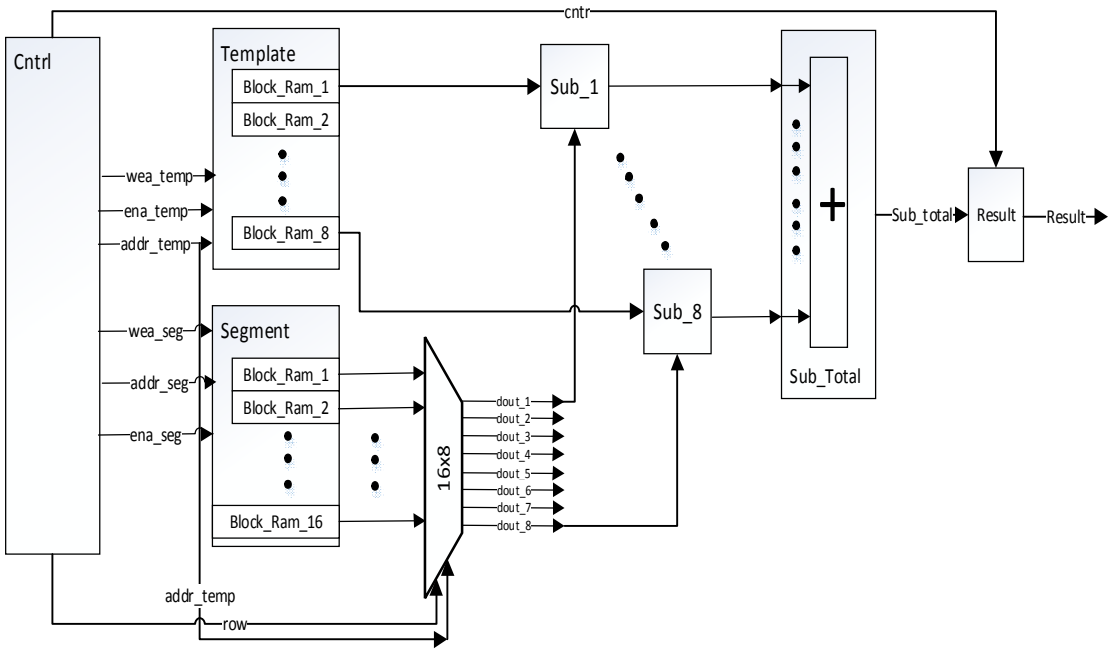
```
a ← 1
for j ← 1 to 16 do
for i ← 1 to 128 do
        t1(i+(j-1)*256) ← T(a)
a ← a + 1
end for
end for
```

S ve T çerçeveleri bellek öbeklerine bu şekilde yerleştirilerek, her saat darbesinde 8 adet piksel okuma yapılır. Böylece Bölüm 3'te önerilen yöntem 8 kat daha hızlandırılmıştır. Okuma işlemi yapılırken T bellek öbekleri için üretilecek adresler her tur için sabit olmak üzere, S 'in yazıldığı bellek öbekler için üretilecek adresler T satırda ve sütunda yer değiştirdikçe sürekli değişecektir. T ve S'yi bellek öbeklerine bu şekilde yazmamızın bir diğer sebebi ise okuma zamanında bir döngü içerisinde S adreslerini oluşturabilmektir. Nitekim önerilen yöntemin FPGA kodları ile gerçeklenmesi sürecinde, zaman maliyeti en yüksek kısım bellek öbekleri adreslerinin üretilmesi olmuştur. Bellek öbeklerinin adresleri bir döngü içerisinde üretilemediği takdirde adreslerin üretilmesi kaotik bir sürece girmekte ve nitekim gerçeklenmesi imkansız olmaktadır. İki aşamalı yöntemin ilk aşaması olan 8 artırmalı tüm tarama için üretilen S adresleri Algoritma 2'deki gibi olmaktadır (Aktaş vd 2015):

Algoritma 2 : S Bellek Öbekleri için Adreslerin Üretilmesi

```
for x ← 1to272do
for j ← 1to16do
for i ← 1to 128 do
   addr_temp ← (i+(j-1).256)
   aadr_seg  ← addr_temp + (column) + (row).256
if j ← 16 & i ← 128  then
column ← column + 8
end if
if column ← 128 then
        column ← 0
        row ← row + 8
end if
end for
end for
end for
```

S çerçevesi toplam 16 adet bellek öbeğine yazılmış olup üretilen addr_seg adresi her bir bellek öbeği için tur sayısına göre farklı bir adres ifade etmektedir. Şekil 4.4'ten görüldüğü gibi Segment modülü içerisindeki bellek öbekleri sürekli aktif olup sürekli çıkış vermektedirler. Ancak SAD algoritmasının gerçeklemesi için T piksellerinin uygun S pikselleri ile işleme sokulması gerekmektedir. Bunu gerçeklemek için 16x8 lik bir çoğullayıcı kullanılmıştır. Bellek öbeklerinin bu şekilde kullanılması sonucu yapılacak modül tabanlı tasarımı Şekil 4.9'daki gibidir.



Şekil 4.9. Algoritmanın FPGA Gerçeklenmesinin Modül Tabanlı Gösterimi

Çizelge 4.7'deki önerilen algoritmayı gerçeklemek için gerekli olan FPGA saat hızı, bellek öbeklerinin efektif bir şekilde kullanılması ve her saat darbesinde 8 adet piksel okunması sonucu Çizelge 4.9'daki gibi geliştirilmiştir.

Çizelge 4.9. Önerilen AlgoritmanınFPGA ile Hızlandırılmasından Sonra Gerekli olan FPGA Saat hızı

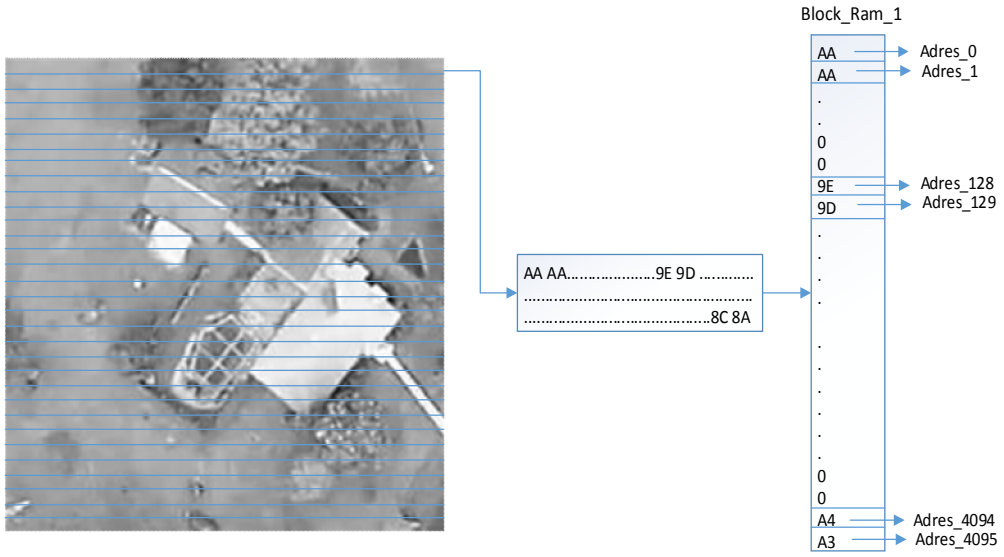
S = mXm	T = nXn	Önerilen Algoritma ile İşlenmesi Gereken Piksel Sayısı	FPGA Saat Hızı
1024X1024	32X32	16,230,400	101,440,000Hz
	64X64	60,891,136	380,569,600Hz
	128X128	212,893,696	1,330,585,600Hz
512X512	32X32	4,040,704	25,254,400Hz
	64X64	14,229,504	88,934,400Hz
	128X128	43,024,384	268,902,400Hz
256X256	32X32	1,091,584	6,822,400Hz
	64X64	3,481,600	21,760,000Hz
	128X128	8,421,376	52,633,600Hz

Çizelge 4.9'dan görüldüğü gibi S = 1024X1204 iken T = 128X128 ve S = 1024X1204 iken T = 64X64 değerleri hariç diğer tüm değerler için algoritma gerçek zamanlı olarak FPGA üzerinde gerçeklenebilir hale gelmiştir.

4.2.1. Farklı sayılarda bellek öbeklerinin kullanılması

Çizelge 4.9'daki S = 1024X1024 iken T = 128X128, S = 1024X1024 iken T = 64X64 ve S = 512X512 iken T = 128X128 değerlerinde ihtiyaç duyulan FPGA saat hızları halen yetersiz olarak görülmektedir. Bu tezdeki çalışmada T çerçevesi sadece 8 adet bellek öbeğine yazılmış olup, istenildiği takdirde bu sayı 2'nin katları ile artırılabilmektedir. S = 256X256 iken T = 128X128 örneğinde T çerçevesi 8 adet bellek öbeğine yazılmış olup; her saat darbesinde 8 farklı piksel okunması sağlanmıştır. Eğer ki T bellek öbeği 16 farklı bellek öbeğine yazılır ise her saat darbesinde 16 adet piksel okuma işlemi yapılabilmektedir. Bu durumda, 256/128 = 2 olduğu için S bellek öbekleri de 16X2 = 32 adet bellek öbeğine yazılması gerekmektedir. S = 256X256 = 65536B olup toplam 32 adet bellek öbeğine yazılacağı için her bellek öbeğine 65536B/32 = 2048 adet S pikseli yerleştirilecektir. Her bir bellek öbeği 4096B olduğu için S pikselleri her bir bellek öbeğinin yarısını dolduracaktır. Şekil 4.10'da S çerçevesinin 32 adet bellek öbeğine yazdırılması gösterilmekte olup bellek öbeğinin ilk 128 adresi pikseller ile doldurulup sonra gelen 128 adres boş bırakılmıştır. Bu işlem Bellek öbeği doldurulana kadar yapılmıştır. Yine aynı şekilde T = 128X128 = 16384B olup toplam 16 adet bellek öbeğine yazılacağı için her bellek öbeğinde 16384B/16 = 1024 adet T pikseli yerleştirilecektir. Her bir bellek öbeği 4096B olduğu için T pikselleri her bir bellek öbeğinin 1/4'ünü dolduracaktır. T pikselleri de bellek öbeklerine bu şekilde yerleştirildikten sonra Algoritma 1 ve 2'nin bu yeni yapıya göre düzenlemesi gerekmektedir. Bu sayede her saat darbesinde 16 adet piksel okuyup ilgili pikselleri birbirleri ile çıkarma işlemine sokmak mümkün olacaktır. Dikkat edilecek olursa herşey 2^n mimarisine göre tasarlanmıştır. S ve T'nin boyutları, kullanılacak bellek öbeği

sayısı, satırda ve sütunda ilerleme sayısı tüm bu değerler 2^n sayılarına göre seçildikten sonra bellek öbeklerini okumak için üretilmesi gereken adresler çok daha rahat ve bir döngü içerisinde üretilebilmektedir.



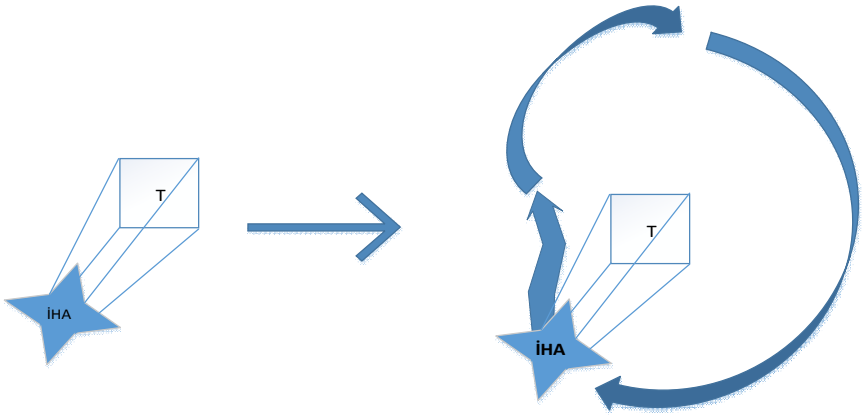
Şekil 4.10. S Çerçevesinin 32 adet Bellek Öbeğine Yazdırılması

Farklı boyutlarda S ve T kullanılması ile birlikte ihtiyaç duyulan bellek öbekleri sayısı da değişecektir. Örneğin S = 512X512 çerçevesi kullanılır ise 512X512 = 261144B demektir. Her bir bellek öbeği 4096B olacağı için bu da S çerçevesini bellek öbeklerine yerleştirmek için 261144/4096 = 64 adet bellek öbeğine ihtiyaç vardır. T = 128X128 olduğu durumda yine bir döngü içerisinde adresleri üretebilmek için (512X512)/(128X128) = 4 adet bellek öbeğine ihtiyaç vardır. Bu durumda S = 512X512 ve T = 128X128 çerçeveleri için 64 adet bellek öbeği S çerçevesi için, 4 adet bellek öbeği de T çerçevesi kullanıldığı takdirde her saat darbesinde 4 adet piksel okunarak işlemler 4 kat hızlandırılmış olur. Eğer ki işlemleri 8 kat hızlandırmak istersek S çerçevesi 128 adet bellek öbeğine, T çerçeveleri de 8 adet bellek öbeğine yerleştirilmesi gerekmektedir ki bu çok fazla bellek öbeğinin kullanılması demektir. Farklı boyutlardaki S ve T çerçeveleri ile yapılacak hızlandırma işlemleri için gerekli olan bellek öbeklerinin sayısı ayrıca hesaplanması gerekmekte olup, büyük boyutlardaki çerçeveler için fazla sayıda bellek öbeği gerekeceği gerçeğini de göz ardı etmemek gerekmektedir.

4.3. Gerçek Zamanlı Sistem Tasarımı

Bu tezdeki amaç İHA'lardaki görüntülerden gerçek zamanlı nesne takibini yapmak olup; nesne takibi için şablon eşleştirme yöntemi kullanılmıştır. Söz konusu İHA'lar, gözetlemenin yapıldığı bölgeden uçarken aynı anda kameradan gelen görüntüyü gerekli istihbarat birimlerine iletmektedir. Bu birimlere gelen görüntülerden bir bölge ya da nesne takip edilmek istenirse İHA'nın dairesel bir hareketle, kameranın ise çok hassas bir şekilde sürekli bu noktayı görecek şekilde döndürülmesi gerekmektedir. Kameranın bu noktaya doğru dönmesini sağlamak için ilk şablonun

oluşturulduğu görüntünün kendisinden sonra gelen görüntüler de arattırılarak n. ve (n+1). çerçeve arasındaki piksel kaymalarını bulup, bu kayma değerlerine göre kamerayı uygun noktaya doğru döndürmek gerekmektedir. Şekil 4.11'de bu işlem temsili olarak gösterilmiştir.



Şekil 4.11. İHA ile Nesne Takibi

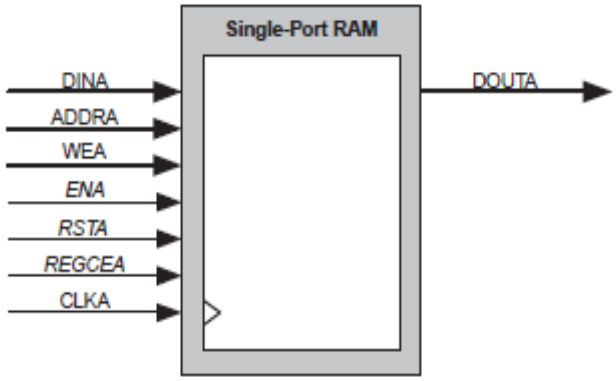
Şekil 4.1'deki n. çerçeve ilk T görüntüsünün oluşturulduğu çerçeve olmak üzere bundan sonra gelen (n+1)., (n+2)., (n+3). ve (n+4). çerçeveler Çizelge 4.1'deki gibi olmak üzere, Çizelge 4.1'de birici sütuna baktığımızda T çerçevesinin S çerçevesi üzerinde ne kadar kaydığını görebiliriz. Örneğin T, (n+1). çerçevede (51,76) noktasında eşleşmiş iken (n+2). çerçevede (52,81) noktasında eşleşmiştir. Yani satırda 1 piksel sütunda ise 5 piksellik kayma mevcuttur. Bu bize T'nin aşağıya doğru 5 piksel, sağa doğru ise 1 piksel kaydığını göstermektedir. Eğer ki kamera (n+3). çerçeveden önce 5 piksel aşağı ve 1 piksel sağa çok hassas bir şekilde döndürülürse bu şekilde bir sonraki (n+3). çerçevede T (53-1,87-5) yani (52,82) noktasında eşleşecektir. (n+3). çerçevede T(52,82) noktasında eşleşince; bu sefer 52-51=1 piksel satırda kayma, 82-76=5 piksel sütunda kayma vardır denilir. (n+4). çerçeveden önce ve diğer çerçevelerde de kameranın konumu bu şekilde güncellenerek, bu işlem gerçek zamanlı bir sisteme dönüştürülmüş olur. Kameranın konumunun bu şekilde güncellenmesi sonucu Çizelge 4.1'in yeni hali Çizelge 4.10'daki gibi olacaktır. Bu şekilde İHA sürekli hareket etse bile T çerçevesi sürekli aynı noktada sabitlenmiş olup, bu çerçevenin takibi gerçekleştirilmiş olacaktır.

Çizelge 4.10. Kamera Konumunun Değişmesi Sonucu Yeni Eşleşme (r,c)

	T = 128X128 ve S= 256X256 Boyutlarındaki Çerçeveler için		
	S Üzerindeki Eşleşme (r,c)	Kamera Konum Değişimi	Değişim Sonrası T'nin Konumu (r,c)
(n+1). Çerçeve	(51,76)	(0,0)	(51,76)
(n+2). Çerçeve	(52,81)	(1,5)	(51,76)
(n+3). Çerçeve	(52,82)	(1,6)	(51,76)
(n+4). Çerçeve	(52,81)	(1,5)	(51,76)

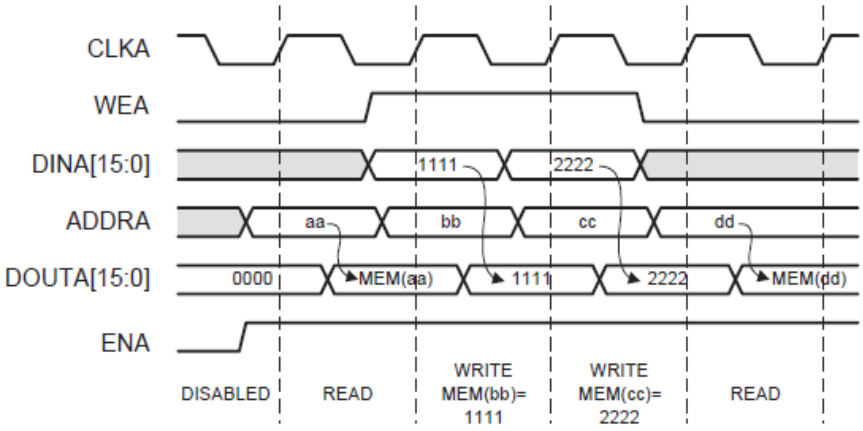
4.3.1. S ve T çerçevelerinin bellek öbeklerine yazdırılması

FPGA üzerinde bulunan bellek öbekleri, single ve dual port RAM olarak kullanılabilmekte olup; bu tezdeki çalışmada bellek öbekleri Single-Port RAM yapısında kullanılmıştır. Şekil 4.12'de Single-Port RAM yapısının giriş çıkış sinyalleri gösterilmektedir.



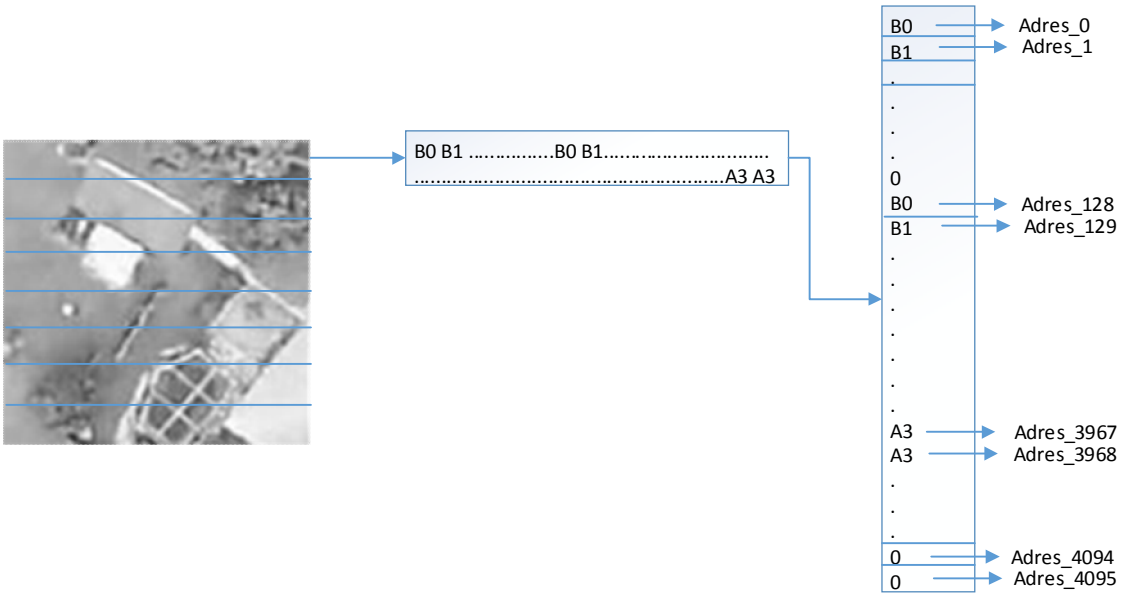
Şekil 4.12. Single-Port RAM Giriş Çıkış Sinyalleri

Giriş çıkış sinyalleri Şekil 4.12'deki gibi olan Single-Port RAM'e yazma işlemi yapılmak istenilirse, uygulanması gereken sinyaller Şekil 4.13'teki gibi olacaktır:



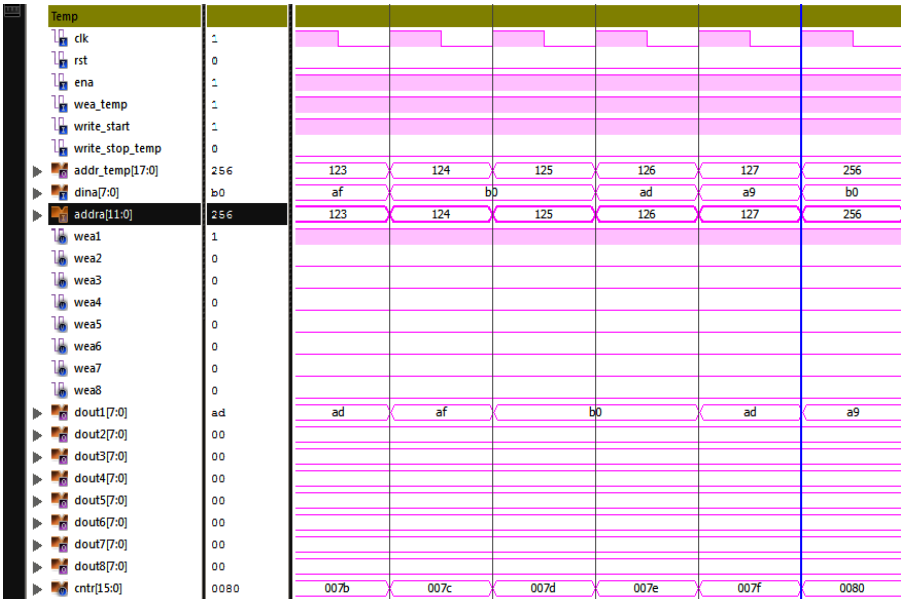
Şekil 4.13. Yazdırma İşlemi

Şekil 4.1'deki gibi ilk görüntüden T = 128X128 çerçevesi oluşturulduktan sonra ilk olarak bu çerçeve FPGA bellek öbeklerine yazdırılacaktır. T çerçevesi Şekil 4.14'daki gibi olmak üzere, her saat darbesinde 8 piksel okumak için T çerçevesi 8 adet bellek öbeğine yerleştirilecektir. Bu durumda her bir bellek öbeğinde 2048 adet T pikseli bulunacaktır. Her bir bellek öbeği 4096B olduğu için, bellek öbeklerinin yarısı boş bırakılacaktır. Bu işlem Şekil 4.14'te gösterildiği gibi olacaktır. Bu durumda T çerçevesinin ilk 128 pikseli bellek öbeğinin ilk 0-127 adreslerine yazdırılacak. Bellek öbeğinin sonraki 128-255 adresleri boş bırakılacak, bu işlem bellek öbeği dolana kadar devam edecektir.



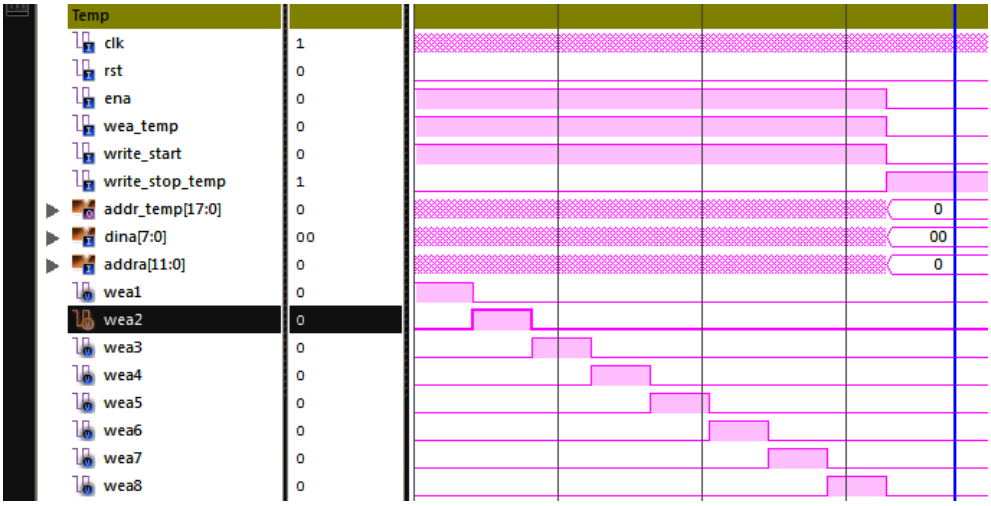
Şekil 4.14. T = 128X128 Çerçevesinin Bellek Öbeklerine Yerleştirilmesi

Şekil 4.14'teki gibi T'ye ait bellek öbeklerinin doldurulması işlemi verilog kodları ile gerçeklendikten sonra, kodların simülasyon çıktısı Şekil 4.15'teki gibidir. 8 bitlik dina datası T çerçevesinden gelen pikselleri göstermek üzere, ena ve write_start sinyallerinin 1' çekilmesi ile bellek öbeklerine yazdırma işlemleri başlamaktadır. Daha önce bahsedilen 128 adet pikseli yazıp diğer 128 adet adresi boş bırakma işlemi Şekil 4.15'te gösterildiği gibi olacaktır. Şekilden de görüleceği gibi en son 126. ve 127. adresler için dina'dan gelen veriler bu adreslere yazılmış olup; sonrasında adres 256'ya çekilip yazma işlemine buradan devam edilmiştir.



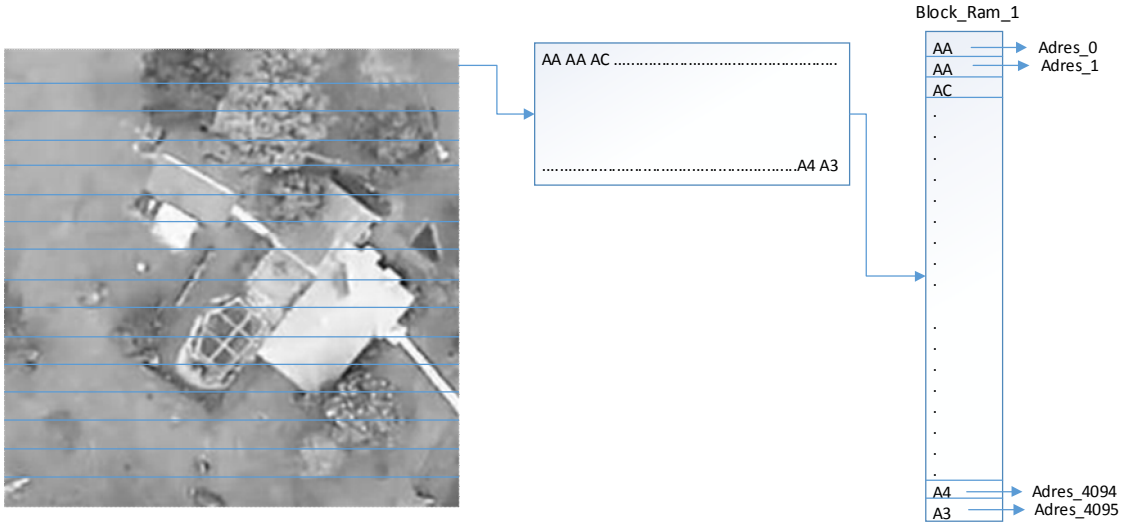
Şekil 4.15. T'ye Ait Tek Bir Bellek Öbeğinin Yazdırılması

Template modülünün içinde 8 adet bellek öbeği bulunup bunların hepsi yazma işlemi için sırası ile aktif edilmektedir. İlk bellek öbeği wea1 sinyali ile aktif edilip, T'ye ait ilk pikseller bu bellek öbeğine yazdırılmaktadır. İlk bellek öbeği doldurulunca daha sonra wea2 sinyalinin 1'e çekilmesi ile 2. bellek öbeği doldurulmaktadır. Bu şekilde tüm bellek öbeklerinin doldurulması işlemi ise Şekil 4.16'daki gibi olacaktır.



Şekil 4.16. T'ye ait Tüm Bellek Öbeklerinin Sırası ile Doldurulması

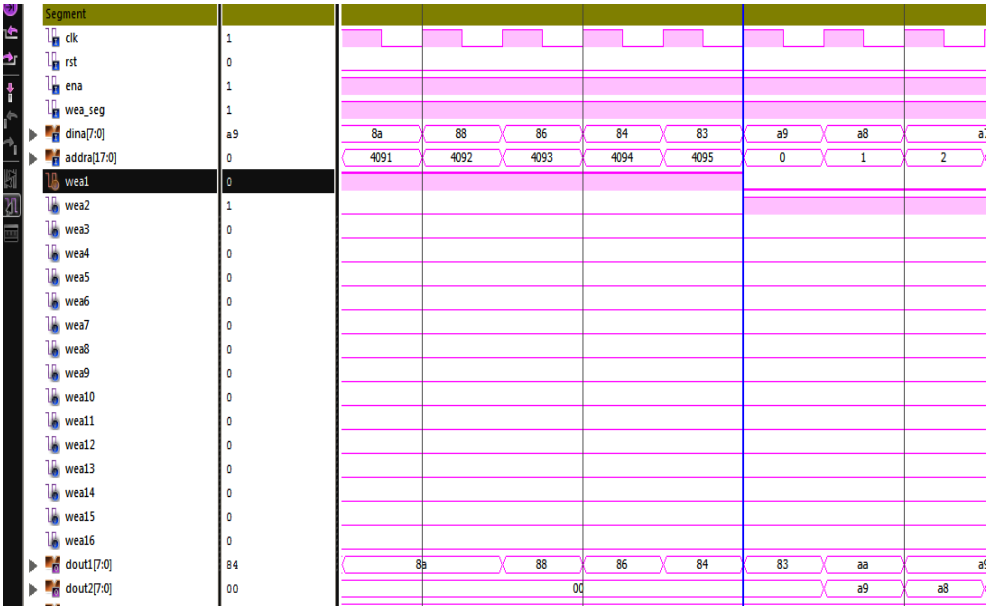
T bellek öbekleri Şekil 4.14, Şekil 4.15 ve Şekil 4.16'da detaylı bir şekilde anlatıldığı gibi doldurulduktan sonra bir sonraki aşamada (n+1). çerçeveden gelen S = 256X256 çerçevesi Şekil 4.17'deki gibi bellek öbeklerine yazdırılacaktır.



Şekil 4.17. S = 256X256 Çerçevesinin Bellek Öbeklerine Yerleştirilmesi

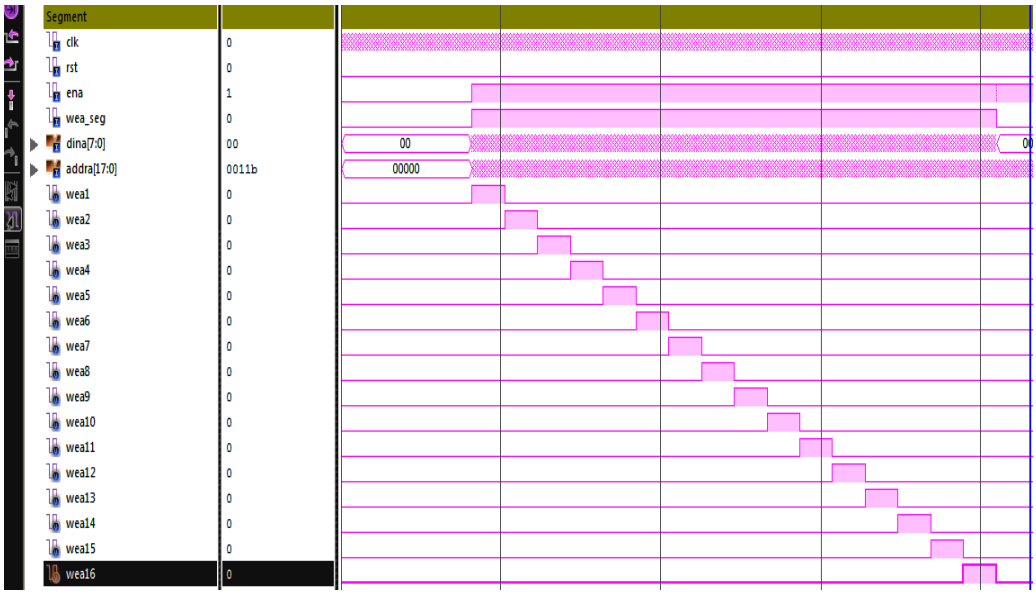
S çerçevesi 16 adet bellek öbeğine yazdırılacak olup, yazdırma işleminin verilog kodları ile gerçeklenmesini gösteren simülasyon çıktısı Şekil 4.18'deki gibidir. Şekil 4.13'de gösterildiği gibi ena ve wea sinyali 1'e çekilerek yazma işlemleri başlamaktadır. dina S çerçevesinden gelen 8 bitlik verileri göstermek üzere, her saat darbesinde dina

datası aynı saat darbesindeki adrese yazdırılmaktadır. S bellek öbekleri doldurulurken, tüm adresler 8 bitlik datalarla doldurulmakta olup; toplam 4096 adet data bir bellek öbeğine yazdırıldıktan sonra adres 0'dan başlamakta, wea1 sinyali 0'a çekilmekte ve wea2 sinyali ise 1'e çekilmektedir. Bu işlem Şekil 4.18 üzerinde açık bir şekilde gözükmektedir.



Şekil 4.18. S'ye Ait Tek Bir Bellek Öbeğinin Yazdırılması

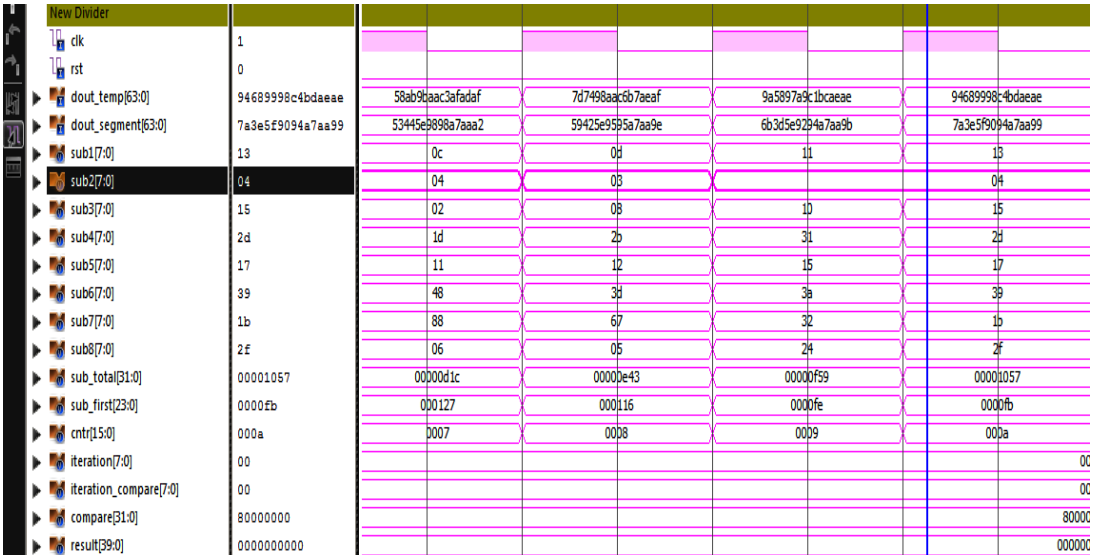
S bellek öbeklerinin tamamının doldurulmasını gösteren simülasyon çıktısı ise Şekil 4.19'daki gibidir. Toplam 16 adet bellek öbeği doldurulduktan sonra bir sonraki çerçeveye kadar yazdırma işlemleri olmayacağı için wea sinyalleri 0'a çekilmiştir.



Şekil 4.19. S'e Ait Tüm Bellek Öbeklerinin Sırası ile Doldurulması

4.3.2. SAD değerinin hesaplanması

Bu tezde şablon eşleştirme için SAD algoritması kullanılmış olup, SAD algoritmasına göre her turda uygun S pikselleri T piksellerinden çıkarılmakta, farkın mutlak değeri alınmakta ve tüm mutlak farklar toplanarak her bir tura ait SAD değeri üretilmektedir. Tüm turlar tamamlandıktan sonra en küçük SAD değerinin olduğu turdaki eşleşme tam eşleşmeyi göstermektedir. Çıkarma işlemi algoritmanın gerçeklenmesi sürecinde önemli bir yere sahip olmak üzere önerilen yöntem ve önerilen yöntemin FPGA bellek öbekleri ile 8 kat daha hızlandırılması sonucu yapılması gereken toplam çıkarma sayısı 1.052.672'dir (Sadece iki çerçevenin birbiri ile karşılaştırılması sonucu gerekli olan çıkarma işlemi). Görüldüğü gibi algoritma çok fazla sayıda çıkarma işlemi içermektedir. Çıkarma işlemleri Şekil 4.9'daki gibi olmak üzere her saat darbesinde 8 adet piksel okunmakta ve 8 adet çıkarma işlemi yapılmaktadır. Çıkarma işlemlerinin ve SAD değerinin hesabının verilog kodları ile gerçeklenmesi ve bu kodların simülasyon çıktısı Şekil 4.20'deki gibidir.



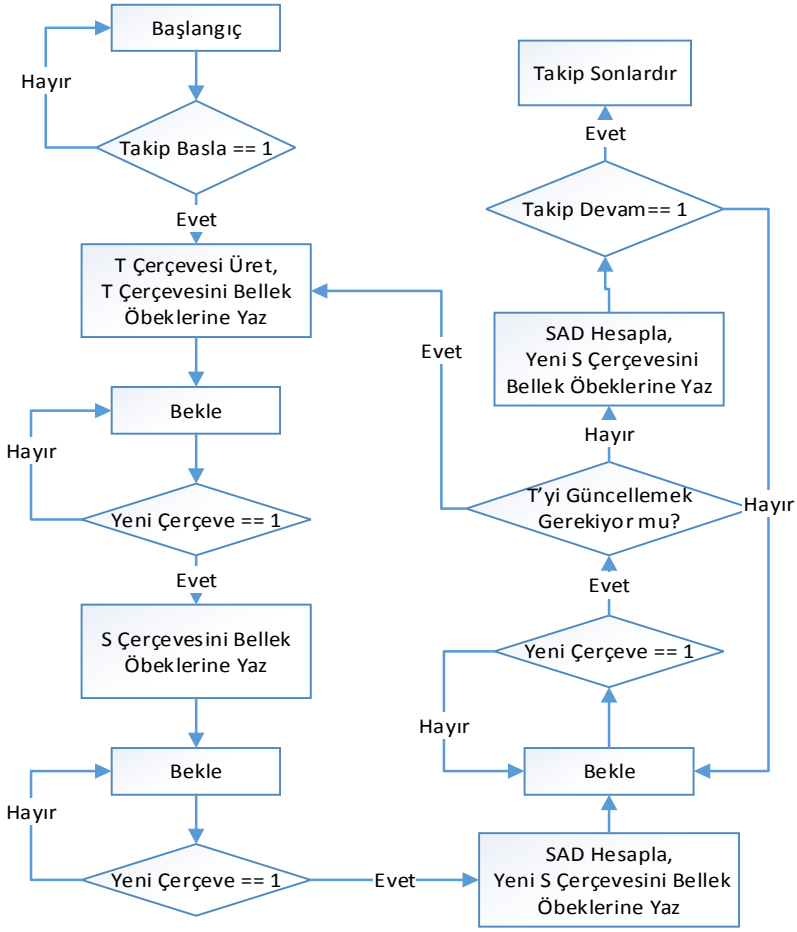
Şekil 4.20. Çıkarma İşlemleri ve SAD Hesabı

Şekilden görüldüğü gibi her saat darbesinde 64 bitlik dout_temp (T çerçevesine ait 8 adet piksel) ve dout_segment (S çerçevesine ait 8 adet piksel) verileri okunmaktadır. Bir sonraki saat darbesinde ise okunan bu 8 piksellik veriler birbirinden çıkartılmaktadır. 8 adet farkın mutlak değeri bir sonraki saat darbesinde subfirst olarak hesaplanmakta ve her saat darbesinde bulunan subfirst değeri bir sonraki saat darbesinde subtotal değerine eklenmektedir. Bu şekilde tüm tur tamamlandıktan sonra SAD değeri hesaplanmış olmaktadır. Şekil 4.20 üzerinden örnek vermek gerekirse: 8. turda yani cntr değeri 8 iken,dout_temp = 7d7498aac6b7aeaf ve dout_segment = 59425e9595a7aa9e olarak okunmaktadır. Bu durumda bir sonraki saat darbesinde $\text{sub1} = |9\text{e} - \text{af}| = 11$, $\text{sub2} = |\text{aa} - \text{ae}| = 4$, ... $\text{sub8} = |59 - 7\text{d}| = 24$ olarak hesaplanmaktadır. Tüm bu değerler bir sonraki saat darbesinde toplanarak subfirst değeri hesaplanmaktadır. Bu durumda $\text{subfirst} = 11 + 04 + 10 + 31 + 15 + 3\text{a} + 32 + 24 = \text{FB}$ olarak hesaplanır. Her turda elde edilen subfirst değerleri subtotal değerine eklenmektedir. Şekle göre cntr = 7 iken: $\text{subtotal} = \text{D1C} + 127 = \text{E43}$ değerini alır. Bir sonraki saat darbesinde yani

cntr=8 iken, $subtotal = E43 + 116 = F59$ olarak hesaplanmaktadır. Tüm turlar tamamlandıktan sonra subtotal değeri compare değeri ile karşılaştırılmakta, yeni bulunan SAD değeri compare değerinden küçükse compare değeri bulunan SAD değeri ile güncellenmektedir. Tüm bu işlemler ilk aşamada ve ikinci aşamada yapıldıktan sonra, en son hesaplanan compare değeri hangi iteration ve cntr değerinde hesaplanmış ise buda bize SAD(r,c) ifadesindeki r ve c değerlerini yani eşleşmenin olduğu piksel noktalarını göstermektedir. Önerilen yöntem FPGA kodları ile bu şekilde gerçeklendikten sonra SAD(r,c) değerleri bir sonraki çerçeveden önce hesaplanmış olup, sistem gerçek zamanlı hale getirilmiş olmaktadır.

4.3.3. S ve T çerçevelerinin güncellenmesi ve bellek yönetimi

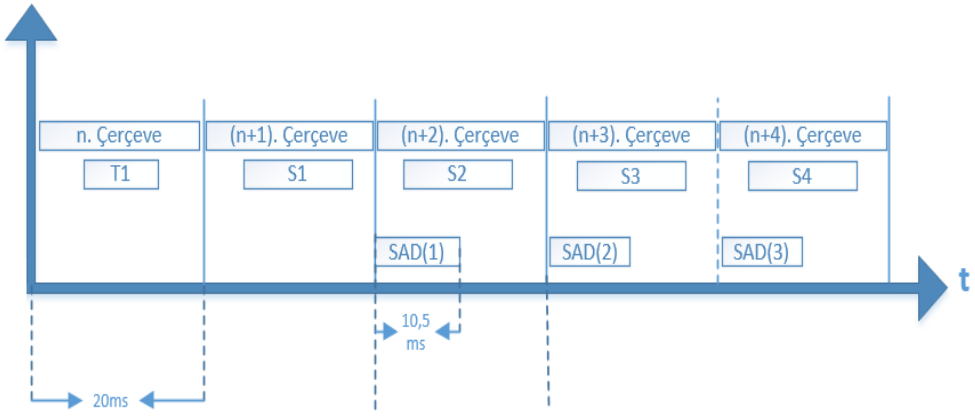
Şekil 4.6'daki gibi İHA ilk T görüntüsünü oluşturduktan sonra T alanı etrafında dolaşarak bu alanı takip etmeye çalışacaktır. Ancak İHA sürekli hareket halinde olduğu için (takip edilecek alan etrafında yapılacak dairesel bir hareket) her ne kadar kamera aynı alanı gösteriyor olsa da zamanla perspektif ve gölgeler aşırı değişeceği için uygun zamanlarda T çerçevesinin güncellenmesi gerekecektir. Takip işleminin başlaması ve bellek öbeklerinin güncellenmesi Şekil 4.21'de temsili olarak gösterilmiştir.



Şekil 4.21. Bellek Öbeklerinin Güncellenmesi

İHA'nın o anki hızı, yerden yüksekliği , S ve T çerçevesinin boyutları ve kamera ile ne kadar yakınlaştırma yapıldığı T çerçevesinin güncellenmesini etkileyen faktörler olacaktır. Yine daha önce bahsedilen kamera konumunun her yeni gelen çerçeveden önce konumunun değiştirilmesi yine bu faktörlere bağlı olacaktır. T, S üzerinde küçük kaymalar yapıyorsa kamera konumunu her SAD değeri için değiştirmek gerekmeyebilir. Şöyle ki n. çerçeveden T üretildi, (n+1). çerçeveden S üretildi ve ilk eşleşme (51,62) noktasında oldu. (n+2). çerçeveden üretilen S ile (52,66) noktasında eşleşme olursa sağa doğru 1, aşağı doğru 4 piksel kayma vardır denilir. (n+3). çerçeveden üretilen S ile yapılan eşleştirme (53,72) noktasında olur ise ilk S'e göre (2,10) kayma, ikinci S'e göre (1,6) kayma vardır. Bu kayma miktarları halen çok küçük olmakla birlikte algoritma için henüz sorun teşkil etmemektedir. Çünkü son eşleşme noktası olan (53,72) için 53+128 = 181, 72+128 = 200 değerleri halen S boyutlarında yani 256 değerinden küçüktür. Böylece her SAD hesabından sonra değil de, muhtemel kritik (r+128>256, c+128>256) eşleşme noktalarından önce kameraya yön verme işlemleri yapılması gerekmektedir. Tüm bu öneriler ve yöntemler çok daha fazla İHA görüntüsü üzerinde denendikten sonra en uygun yönteme karar verilebilir.

Bu tez çalışmasında kullanılan S = 256x256 ve T = 128x128 çerçeveleri üzerinden bellek yönetimini anlatmak gerekirse: önerilen yöntemin kullanıldığı sistemde kamera hızı 50fps'dir. Bu durumda her 20ms'de bir yeni S ve daha önce bahsedilen sebeplerden dolayı da farklı zamanlarda T çerçeveleri güncellenmektedir. FPGA saat hızı 100Mhz iken bölüm 4.3.2.'de anlatılan SAD değerinin hesabı ise 10,526ms sürmektedir. Tüm bu işlemlerin zaman grafiği üzerinde gösterilmesi Şekil 4.22'deki gibi olacaktır:



Şekil 4.22. Zaman Grafiği

Şekilde sadece S çerçevelerinin güncellenmesi gösterilmiştir. İhtiyaca göre farklı zamanlarda T çerçeveleri de güncellenmektedir. S çerçeveleri 16 adet bellek öbeğine yazılmış olup, S çerçevelerinin güncellenmesi için 16 adet farklı bellek öbeğine ihtiyaç duyulmaktadır. Yine T çerçeveleri güncellenirken 8 farklı bellek öbeği kullanılabilir ya da bellek öbeklerinin boş kalan yarısı kullanılabilir. Bu işlem daha önce anlatılan algoritma 1'de üretilen tüm adreslere 128 ekleyerek gerçekleştirilebilir. Bu sayede 8 adet bellek öbeğinden tasarruf edilmiş olunur.

5. SONUÇLAR VE GELECEK ÇALIŞMALAR

Bu çalışmada, işlem yükü yüksek, Tüm Çerçeve Taramasına dayalı Şablon Eşleştirme Algoritmasını hızlandırmak için hesap yükü düşük yeni bir yaklaşım önerilmiştir. İşlenen imgenin satır ve sütunlarındaki ilerleme adımına bağlı olarak önerilen yöntem, hesap yükünü azaltabilmektedir. İşlem yükü önemli ölçüde azalırken tam eşleşme başarılı bir şekilde gerçeklenmiştir. Yine algoritmanın farklı boyutlarda çerçevelerde kullanılabileceği ve önerilen yöntemin geliştirilmeye açık olduğu testlerle detaylı bir şekilde gösterilmiştir.

Önerilen yöntemin son hali FPGA donanımı üzerinde gerçekleneceği için, donanım maliyeti düşük bir yöntem önerilmeye çalışılmıştır. Nitekim tüm parametreler 2^n kümesinden seçildiği için donanım maliyeti düşük bir yöntem elde edilmiştir. FPGA üzerindeki bellek öbekleri etkin bir şekilde kullanılarak, eşleştirme işlemi gerçek zamanda çalışabilecek hale getirilmiştir. FPGA üzerindeki bellek öbekleri kullanma sayısına bağlı olarak algoritmanın hesap yükü daha da azaltılabilmektedir. Elde edilen sonuçlar, önerilen yöntemin İHA görüntülerinin eşleştirilmesinde kullanılabileceğini düşündürmektedir.

Önerilen yöntem ve FPGA gerçeklemesi gelecekte daha da geliştirilerek İHA'lar üzerinde kullanılması düşünülmektedir. İHA'lardan elde edilecek görüntülerde yapılacak gerçek zamanlı testler sonucunda en uygun parametrelere (satır ve sütunda ilerleme sayısı, S ve T çerçevelerinin boyutları, kullanılacak bellek öbekleri sayısı) karar verilecektir. Yine İHA'nın irtifası, hızı ve takip edilecek alanın büyüklüğüne göre, parametrelerin duruma göre güncellenmesi gerektiği düşünülmektedir. En uygun parametrelere karar verildikten sonra elde edilecek veriler (satır ve sütundaki kayma miktarları) İHA üzerinde bulunan kameraya yön vermek için kullanılacaktır. Bu haliyle yapılan tez çalışmaları literatürde kendine yer bulmakla birlikte uygulamada da kendine geniş bir yer bulacağı düşünülmektedir.

Yapılan iyileştirmelerle FPGA bellek öbeklerini uygun şekilde kullanarak hızlandırılan yöntemin, işlem yükü yüksek diğer gerçek zamanlı görüntü işleme uygulamaları için de uyarlanabilir olduğu düşünülmektedir. Benzer biçimde, önerilen iki aşamalı yöntemin, yüksek sığalı verilerin (hiperspektral görüntüler gibi), gerçek zamanlı olarak işlenmesi ve/veya sıkıştırılması amacıyla da kullanılabileceği öngörülmektedir.

6. KAYNAKLAR

AKTAŞ, H., SEVER, R. ve TÖREYİN, B.U. 2015. "İki Aşamalı Şablon Eşleştirme Algoritması ve Yüksek Hızlı FPGA Gerçeklemesi". *SİU-2015: Sinyal İşleme ve İletişim Uygulamaları Kurultayı* 3:2114–17

AMDAHL, G.M. 1967. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, 483–85. AFIPS '67 (Spring). New York, NY, USA: ACM. doi:10.1145/1465482.1465560.

BACKUS, J. 1978. "Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs". *Commun. ACM* 21 (8): 613–41. doi:10.1145/359576.359579.

BIERLING, M. 1988. "Displacement estimation by hierarchical block matching", 942–51. doi:10.1117/12.969046.

BRAHIM, B.S., JOSEFINA J. and NOURAIN, N. 2011. "Fast Template Matching Method based Optimized Sum of Absolute Difference Algorithm for Face Localization". *International Journal of Computer Applications (IJCA)*, Mart. http://eprints.utp.edu.my/4685/.

BROWN, L. G. 1992. "A Survey of Image Registration Techniques". *ACM Comput. Surv.* 24 (4): 325–76. doi:10.1145/146370.146374.

CHALIDABHONGSE, J. and KUO, C.C.J. 1997. "Fast motion vector estimation using multiresolution-spatio-temporal correlations". *IEEE Transactions on Circuits and Systems for Video Technology* 7 (3): 477–88. doi:10.1109/76.585927.

CHEN, L.G., CHEN, W.T., JEHNG, Y.S. and CHIUCH, T.D. 1991. "An efficient parallel motion estimation algorithm for digital image processing". *IEEE Transactions on Circuits and Systems for Video Technology* 1 (4): 378–85. doi:10.1109/76.120779.

CHEN, M.J., CHEN, L.G. and CHİUEH, T.D. 1994. "One-dimensional full search motion estimation algorithm for video coding". *IEEE Transactions on Circuits and Systems for Video Technology* 4 (5): 504–9. doi:10.1109/76.322998.

CHEN, O.T.C. 2000. "Motion estimation using a one-dimensional gradient descent search". *IEEE Transactions on Circuits and Systems for Video Technology* 10 (4): 608–16. doi:10.1109/76.845006.

CHEUNG, C.H. and PO, L.M. 2002. "A novel cross-diamond search algorithm for fast block motion estimation". *IEEE Transactions on Circuits and Systems for Video Technology* 12 (12): 1168–77. doi:10.1109/TCSVT.2002.806815.

CHOI, J.H., LEE, K.H., CHA, K.C., KWON, J.S., KIM, D.W. and SONG, H.K. 2006. "Vehicle Tracking using Template Matching based on Feature Points". *2006 IEEE International Conference on Information Reuse and Integration*, 573–77. doi:10.1109/IRI.2006.252477.

CHRISTOPOULOS, V., and CORNELIS, J. 2000. "A center-biased adaptive search algorithm for block motion estimation". *IEEE Transactions on Circuits and Systems for Video Technology* 10 (3): 423–26. doi:10.1109/76.836287.

COPE, B., CHEUNG, P.Y.K., LUK, W. and WITT, S. 2005. "Have GPUs made FPGAs redundant in the field of video processing?". *2005 IEEE International Conference on Field-Programmable Technology, 2005. Proceedings*, 111–18. doi:10.1109/FPT.2005.1568533.

DOUGHERTY, E.R., and LAPLANTE, P.A. 1995. *Introduction to Real-Time Imaging*. SPIE Press.

FERRARI, V., TUYTELAARS, T. and GOOL, L. V. 2004. "Simultaneous Object Recognition and Segmentation by Image Exploration". *Computer Vision - ECCV 2004*, Tomás Pajdla ve Jiří Matas, 40–54. Lecture Notes in Computer Science 3021. Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007/978-3-540-24670-1_4.

GEER, D. 2005. "Chip makers turn to multicore processors". *Computer* 38 (5): 11–13. doi:10.1109/MC.2005.160.

GHANBARI, M. 1990. "The cross-search algorithm for motion estimation [image coding]". *IEEE Transactions on Communications* 38 (7): 950–53. doi:10.1109/26.57512.

GONZALEZ, R.C., and RICHARD, E.W. 2008. *Digital Image Processing*. Prentice Hall.

HE, Z.L., TSUİ,C.Y., CHAN,K.K. and LIOU, M.L. 2000. "Low-power VLSI design for motion estimation using adaptive pixel truncation". *IEEE Transactions on Circuits and Systems for Video Technology* 10 (5): 669–78. doi:10.1109/76.856445.

HANNA, G., 2011. Object Tracking. Hard Cover, Publisher: In Tech, Subject: Artificial Intelligence, pp: 284, ISBN: 978-953-307-360-6

HUANG, Y.W., CHEN, C.Y., TSAI,C.H., SHEN,C.F. and CHEN, L.G. 2006. "Survey on Block Matching Motion Estimation Algorithms and Architectures with New Results". *J. VLSI Signal Process. Syst.* 42 (3): 297–320. doi:10.1007/s11265-006-4190-4.

HUANG, Y.W., MA,S.Y., SHEN,C.F. and CHEN, L.G. 2003. "Predictive line search: an efficient motion estimation algorithm for MPEG-4 encoding systems on

multimedia processors". *IEEE Transactions on Circuits and Systems for Video Technology* 13 (1): 111–17. doi:10.1109/TCSVT.2002.808093.

HU, W., TAN, T., WANG, L. and MAYBANK, S. 2004. "A survey on visual surveillance of object motion and behaviors". *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 34 (3): 334–52. doi:10.1109/TSMCC.2004.829274.

JAIN, J., and JAIN, A. 1981. "Displacement Measurement and Its Application in Interframe Image Coding". *IEEE Transactions on Communications* 29 (12): 1799–1808. doi:10.1109/TCOM.1981.1094950.

KAPPAGANTULA, S. and RAO, K.R. 1985. "Motion Compensated Interframe Image Prediction". *IEEE Transactions on Communications* 33 (9): 1011–15. doi:10.1109/TCOM.1985.1096415.

KAWANISHI, T., KUROZUMI, T., KASHINO,K. and TAKAGI, S. 2004. "A fast template matching algorithm with adaptive skipping using inner-subtemplates' distances". *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*, 3:654–57 Vol.3. doi:10.1109/ICPR.2004.1334614.

KEHTARNAVAZ, N. and GAMADIA, M. 2006. "Real-Time Image and Video Processing: From Research to Reality". *Synthesis Lectures on Image, Video, and Multimedia Processing* 2 (1): 1–108. doi:10.2200/S00021ED1V01Y200604IVM005.

KETTNAKER, V. and ZABIH, R. 1999. "Bayesian multi-camera surveillance". *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, 2:-259 Vol. 2. doi:10.1109/CVPR.1999.784638.

KOGA, T. 1981. "Motion-compensated interframe coding for video conferencing". *Proc. NTC 81*, C9. 6. 1–9. 6. 5.

LEE, J.H. and LEE, N.S. 2004. "Variable block size motion estimation algorithm and its hardware architecture for H.264/AVC"., 741–44. http://cat.inist.fr/?aModele=afficheN&cpsidt=18144409.

LI, R., ZENG, B. and LİOU, M.L. 1994. "A new three-step search algorithm for block motion estimation". *IEEE Transactions on Circuits and Systems for Video Technology* 4 (4): 438–42. doi:10.1109/76.313138.

LIU, B. and ZACCARİN, A. 1993. "New fast algorithms for the estimation of block motion vectors". *IEEE Transactions on Circuits and Systems for Video Technology* 3 (2): 148–57. doi:10.1109/76.212720.

LIU, L.K. and FEIG, E. 1996. “A block-based gradient descent search algorithm for block motion estimation in video coding”. *IEEE Transactions on Circuits and Systems for Video Technology* 6 (4): 419–22. doi:10.1109/76.510936.

LI, W. and SALARI, E. 1995. “Successive elimination algorithm for motion estimation”. *IEEE Transactions on Image Processing* 4 (1): 105–7. doi:10.1109/83.350809.

LOWE, D.G. 2004. “Distinctive Image Features from Scale-Invariant Keypoints”. *International Journal of Computer Vision* 60 (2): 91–110. doi:10.1023/B:VISI.0000029664.99615.94.

LUO, J.H., WANG, C.N. and CHIANG, T. 2002. “A novel all-binary motion estimation (ABME) with optimized hardware architectures”. *IEEE Transactions on Circuits and Systems for Video Technology* 12 (8): 700–712. doi:10.1109/TCSVT.2002.800859.

MAINI, R. and AGGARWAL, H. 2010. journal of computıng, volume 2, ıssue 3, march 2010, ıssn 2151-9617

MANOCHA, D. 2005. “General-Purpose Computations Using Graphics Processors”. *Computer*.

MATTOCCIA, S., TOMBARİ, F. and STEFANO, L.D. 2008. “Fast Full-Search Equivalent Template Matching by Enhanced Bounded Correlation”. *IEEE Transactions on Image Processing* 17 (4): 528–38. doi:10.1109/TIP.2008.919362.

MCLAUGHLIN, J. 2000. The development of a Java image processing framework. Master of Technology Thesis, Institute of Information Sciences and Technology, Massey University: Palmerston North, New Zealand.

MOSQUERON, R., DUBOIS, J. and PAINDAVOINE, M. 2007. “High-speed Smart Camera with High Resolution”. *EURASIP J. Embedded Syst.* 2007 (1): 23–23. doi:10.1155/2007/24163.

NVIDIA 2006. Technical Brief – NVIDIA GeForce 8800 GPU Architecture Overview, NVIDIA Corporation.

PELEG, A., WILKIE, S. and WEISER, U. 1997. “Intel MMX for Multimedia PCs”. *Commun. ACM* 40 (1): 24–38. doi:10.1145/242857.242865.

PO, L.M. and MA, W.C. 1996. “A novel four-step search algorithm for fast block motion estimation”. *IEEE Transactions on Circuits and Systems for Video Technology* 6 (3): 313–17. doi:10.1109/76.499840.

PRATT, W.K. 2007. DIGITAL IMAGE PROCESSING, PIKS Scientific Inside, Fourth Edition, PixelSoft, Inc.Los Altos, California 2007

ROMA, N., DİAS,T. and SOUSA, L. 2003. "Customisable Core-Based Architectures for Real-Time Motion Estimation on FPGAs". *Field Programmable Logic and Application*, Peter Y. K. Cheung ve George A. Constantinides, 745–54. Lecture Notes in Computer Science 2778. Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007/978-3-540-45234-8_72.

SAHANI, S.K., ADHIKARI, G. and DAS, B.K. 2011. "A fast template matching algorithm for aerial object tracking". *2011 International Conference on Image Information Processing (ICIIP)*, 1–6. doi:10.1109/ICIIP.2011.6108841.

SARAVANAN, C. and SURENDER, M. 2013. "Algorithm for Face Matching Using Normalized Cross-Correlation". *International Journal of Engineering and Advanced Technology (IJEAT) ISSN*, 2249–8958.

SOLOMON, C. and BRECKON, T. 2011. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. John Wiley & Sons.

SRINIVASAN, R. and RAO, K.R. 1985. "Predictive Coding Based on Efficient Motion Estimation". *IEEE Transactions on Communications* 33 (8): 888–96. doi:10.1109/TCOM.1985.1096398.

TOURAPIS, A.M., AU, O.C. and LIOU, M.L. 2002. "Highly efficient predictive zonal algorithms for fast block-matching motion estimation". *IEEE Transactions on Circuits and Systems for Video Technology* 12 (10): 934–47. doi:10.1109/TCSVT.2002.804894.

WANG, Y., WANG Y.and KURODA, H. 2000. "A globally adaptive pixel-decimation algorithm for block-motion estimation". *IEEE Transactions on Circuits and Systems for Video Technology* 10 (6): 1006–11. doi:10.1109/76.867940.

WONG, S., VASSILIADIS, S. and COTOFANA, S. 2002. "A sum of absolute differences implementation in FPGA hardware". *Euromicro Conference, 2002. Proceedings. 28th*, 183–88. doi:10.1109/EURMIC.2002.1046155.

XU, R.Y.D., ALLEN, J.G. and JIN, J.S. 2004. "Robust Real-time Tracking of Non-rigid Objects". *Proceedings of the Pan-Sydney Area Workshop on Visual Information Processing*, 95–98. VIP '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc. http://dl.acm.org/citation.cfm?id=1082121.1082138.

YILMAZ, A., JAVED, O. and SHAH, M. 2006. "Object Tracking: A Survey". *ACM Comput. Surv.* 38 (4). doi:10.1145/1177352.1177355.

ZHU, S. and MA, K.K. 2000. "A new diamond search algorithm for fast block-matching motion estimation". *IEEE Transactions on Image Processing* 9 (2): 287–90. doi:10.1109/83.821744.

7. EKLER

EK1- MATLAB KODLARI

```
clear all
close all
clc

t = imread('1.jpg','jpg');
t = rgb2gray(t);
template = t( 350:477,750:877,:); %% (sütun,satır) (column,row)
imwrite(template,'Template_128x128.jpg','jpg');
template2 = reshape(template',1,[]);

s = imread('39.jpg','jpg');
s=rgb2gray(s);
segment = s(280:535,700:955);
imwrite(segment,'segment_256x256.jpg','jpg');
segment2 = reshape(segment',1,[]);
segment2 = double(segment2);

%%%%%%%%%%%%%%% Blok Ramlerin Oluşturulması%%%%%%%%%%%%%%%%%%%%%

t1 = zeros(1,4096);
t2 = zeros(1,4096);
t3 = zeros(1,4096);
t4 = zeros(1,4096);
t5 = zeros(1,4096);
t6 = zeros(1,4096);
t7 = zeros(1,4096);
t8 = zeros(1,4096);

s1  = zeros(1,4096);
s2  = zeros(1,4096);
s3  = zeros(1,4096);
s4  = zeros(1,4096);
s5  = zeros(1,4096);
s6  = zeros(1,4096);
s7  = zeros(1,4096);
s8  = zeros(1,4096);
s9  = zeros(1,4096);
s10 = zeros(1,4096);
s11 = zeros(1,4096);
s12 = zeros(1,4096);
s13 = zeros(1,4096);
s14 = zeros(1,4096);
s15 = zeros(1,4096);
s16 = zeros(1,4096);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 imshow(t);
 figure
 imshow(template);
 figure
 imshow(s)
 figure
 imshow(segment);
```

```
a = int32(1);

%%%%%segment piksellerinin 16 adet block rame yazdırılması%%%%%%

for i =1:4096,
    s1(i)  = segment2(a);
    s2(i)  = segment2(a+4096);
    s3(i)  = segment2(a+8192);
    s4(i)  = segment2(a+12288);
    s5(i)  = segment2(a+16384);
    s6(i)  = segment2(a+20480);
    s7(i)  = segment2(a+24576);
    s8(i)  = segment2(a+28672);
    s9(i)  = segment2(a+32768);
    s10(i) = segment2(a+36864);
    s11(i) = segment2(a+40960);
    s12(i) = segment2(a+45056);
    s13(i) = segment2(a+49152);
    s14(i) = segment2(a+53248);
    s15(i) = segment2(a+57344);
    s16(i) = segment2(a+61440);
a = a+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = int32(1);

%%%%% template piksellerinin 8 adet block rame yazdırılması%%%%%%%%

for j = 1:16,
for i =1:128,
    t1(i+(j-1)*256) = template2(a);
    t2(i+(j-1)*256) = template2(a+2048);
    t3(i+(j-1)*256) = template2(a+4096);
    t4(i+(j-1)*256) = template2(a+6144);
    t5(i+(j-1)*256) = template2(a+8192);
    t6(i+(j-1)*256) = template2(a+10240);
    t7(i+(j-1)*256) = template2(a+12288);
    t8(i+(j-1)*256) = template2(a+14336);
a = a+1;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%



%%%%%%%%%%%%%%%%%% corelation işleminin yapılması
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

column    = double(0);
row       = double(0);

temp1     = double(0);
temp2     = double(0);
temp3     = double(0);
temp4     = double(0);
temp5     = double(0);
temp6     = double(0);
```

```
temp7     = double(0);
temp8     = double(0);

%%%%%% Her ne kadar segmentleri yazmak için 16 adet blok ram
oluşturulsada simulasyonda kolaylık için segmentler için 8 farklı
adres üretilip büyük resim(segment2) üzerinden direk okuma
yapılmıştır. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

seg1      = double(0);
seg2      = double(0);
seg3      = double(0);
seg4      = double(0);
seg5      = double(0);
seg6      = double(0);
seg7      = double(0);
seg8      = double(0);

addr_temp = double(0);
aadr_seg  = double(0);
sub       = double(0);
subtotal  = double(0);
result    = double(999999999);

index     = zeros(1,2); %%%% index bize resmin ne kadar sağa ve aşağı
gittiği bilgisini verecektir.


for x = 1:272,   %%% 128 / 8 = 16 kere şağa gidilecek, + ilk başlangıç
durumu = 17 kere , 16 tane 8 lik satır var bu durumda toplam 16*17 =
272 durum vardır
for j = 1:16,
for i = 1:128,

    addr_temp = (i+(j-1)*256);
    aadr_seg  = addr_temp + (column) + (row)*256;
    seg1 = segment2 (aadr_seg);
    seg2 = segment2 (aadr_seg+ 4096);
    seg3 = segment2 (aadr_seg+ 8192);
    seg4 = segment2 (aadr_seg+ 12288);
    seg5 = segment2 (aadr_seg+ 16384);
    seg6 = segment2 (aadr_seg+ 20480);
    seg7 = segment2 (aadr_seg+ 24576);
    seg8 = segment2 (aadr_seg+ 28672);

    temp1 = t1(addr_temp);
    temp2 = t2(addr_temp);
    temp3 = t3(addr_temp);
    temp4 = t4(addr_temp);
    temp5 = t5(addr_temp);
    temp6 = t6(addr_temp);
    temp7 = t7(addr_temp);
    temp8 = t8(addr_temp);

    sub = ( abs(temp1-seg1) + abs(temp2-seg2) + abs(temp3-seg3) +
abs(temp4-seg4) + abs(temp5-seg5) + abs(temp6-seg6) + abs(temp7-seg7)
+ abs(temp8-seg8));
    subtotal = subtotal + sub;

if j == 16 & i == 128
```

```
if subtotal < result
           result = subtotal;
           index = [column ,row]; %% satırda ilerleme columna
ekleniyor ancak sonuctaki sayı sütun değeri
           subtotal  = 0;
else result = result;
           index = index;
           subtotal  = 0;
end

       column = column + 8;

end

if column == 128;
           column = 0;
           row = row + 8;
end

end
end
end


result_image_1 = segment( index(2):index(2)+127 ,
index(1):index(1)+127 );
figure
imshow(result_image_1);

%%%%%%%%%%%%%%%%%%%% İkinci Aşama ( birer kaydırma ile 1 piksel
hassasiyet)%%%%%%%%%%%%%%%%%%

result      = double(999999999);
index2      = zeros(1,2);
index_final = zeros(1,2);

column    = double(0);
row       = double(0);

for x = 1:225,  %% 14  kere sağa + başlangıç durumu = 15 durum
for j = 1:16,  %% toplam 15 satır var 15*15 = 225 durum
for i = 1:128,

    addr_temp = (i+(j-1)*256);
    aadr_seg  = addr_temp + ((index(1)-7)+column) + ((index(2)-
7)+row)*256;
    seg1 = segment2 (aadr_seg);
    seg2 = segment2 (aadr_seg+ 4096);
    seg3 = segment2 (aadr_seg+ 8192);
    seg4 = segment2 (aadr_seg+ 12288);
    seg5 = segment2 (aadr_seg+ 16384);
    seg6 = segment2 (aadr_seg+ 20480);
    seg7 = segment2 (aadr_seg+ 24576);
    seg8 = segment2 (aadr_seg+ 28672);


    temp1 = t1(addr_temp);
    temp2 = t2(addr_temp);
    temp3 = t3(addr_temp);
```

```
    temp4 = t4(addr_temp);
    temp5 = t5(addr_temp);
    temp6 = t6(addr_temp);
    temp7 = t7(addr_temp);
    temp8 = t8(addr_temp);

    sub = ( abs(temp1-seg1) + abs(temp2-seg2) + abs(temp3-seg3) +
abs(temp4-seg4) + abs(temp5-seg5) + abs(temp6-seg6) + abs(temp7-seg7)
+ abs(temp8-seg8));
    subtotal = subtotal + sub;

if j == 16 & i == 128
if subtotal < result
           result = subtotal;
           index2 = [column ,row];
           subtotal  = 0;
else result = result;
           index2 = index2;
           subtotal  = 0;
end

       column = column + 1;

end

if column == 15;
           column = 0;
           row = row + 1;
end

end
end
end

index
index2
index_final = [ ( index(1)-7+index2(1) ) , ( index(2)-7+index2(2) ) ]

result_image_2 = segment( index_final(2):index_final(2)+127
,index_final(1):index_final(1)+127 );
figure
imshow(result_image_2);
figure
subplot(1,3,1) ; imshow(template); title(' a) template');
subplot(1,3,2) ; imshow(result_image_1); title('b) ilk tur sonucu');
subplot(1,3,3) ; imshow(result_image_2); title('c) ikinci tur
sonucu');
```

ÖZGEÇMİŞ

Hakan AKTAŞ 1984 yılında Niğde'de doğdu. Niğde Fen Lisesinden mezun olduktan sonra İstanbul Üniversitesi Elektrik-Elektronik Mühendisliğinden 2008 yılında mezun oldu. 2008-2011 yılları arasında özel sektörde farklı işlerde çalıştı. Ocak 2012-Haziran 2015 yılları arasında, Akdeniz Üniversitesi Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı'nda Yüksek Lisans öğrenimini tamamladı. Yine Ocak 2012'den beri Akdeniz Üniversitesi Elektrik-Elektronik Mühendisliğinde Araştırma görevlisi olarak çalışmaktadır.