KABLOSUZ MULTİMEDYA ALGILAYICI AĞLARDA VERİMLİ GÖRÜNTÜ SIKIŞTIRMA VE İLETİM TEKNİKLERİ

Ali DERİNOĞLU

YÜKSEK LİSANS TEZİ
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Dr. Öğr. Üyesi Ali AKMAN

İstanbul
T.C. Maltepe Üniversitesi
Fen Bilimleri Enstitüsü
Mayıs, 2018

JÜRİ VE ENSTİTÜ ONAYI

ALİ DERİNOĞLU'nun "Kablosuz Multimedya Algılayıcı Ağlarda Verimli Görüntü Sıkıştırma ve İletim Teknikleri" başlıklı tezi 22/05/2018 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Maltepe Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliği"nin ilgili maddeleri uyarınca, Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans/Doktora tezi **oy birliğiyle** / ~~**oy çokluğuyla**~~ olarak kabul edilmiştir.

	Unvanı. Adı ve Soyadı	İmza
Üye (Tez Danışmanı)	: Dr. Öğr. Üyesi Ali AKMAN	.....
Üye	: Prof. Dr. Mesut RAZBONYALI	.....
Üye	: Dr. Öğr. Üyesi Erdal GÜVENOĞLU	.....

Prof.Dr. İlter BÜYÜKDIĞAN

Enstitü Müdürü

maltepe üniversitesi	ETİK İLKE VE KURALLARA UYUM BEYANI	Doküman No	FR-178
		İlk Yayın Tarihi	01.03.2018
		Revizyon Tarihi	
		Revizyon No	00
		Sayfa	1/1

Revizyon Takip Tablosu

REVİZYON NO	TARİH	AÇIKLAMA
00	01.03.2018	İlk yayın.

ETİK İLKE VE KURALLARA UYUM BEYANI

01/06/2018

Bu tezin bana ait, özgün bir çalışma olduğunu; çalışmamın hazırlık, veri toplama, analiz ve bilgilerin sunumu olmak üzere tüm aşamalarından bilimsel etik ilke ve kurallara uygun davrandığımı; bu çalışma kapsamında elde edilmeyen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi; çalışmanın Maltepe Üniversitesinde kullanılan "bilimsel intihal tespit programı" ile tarandığını ve öngörülen standartları karşıladığını beyan ederim.

Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçlara razı olduğumu bildiririm.

Ali DERİNOĞLU

Hazırlayan İlgili Birim	Kalite Koordinatörü Yrd. Doç. Dr. Şafak GÜNDÜZ	Kurumsal Yetkili Prof. Dr. Belma AKŞİT

(Dokuman No: FR-178; Yayın Tarihi: 01.03.2018; Revizyon Tarihi: ; Revizyon No:00)

İNTİHAL RAPORU

Yüksek Lisans Tezi

ORİJİNALLİK RAPORU

%11	%9	%4	%3
BENZERLİK ENDEKSİ	İNTERNET KAYNAKLARI	YAYINLAR	ÖĞRENCİ ÖDEVLERİ

BİRİNCİL KAYNAKLAR

1	www.ebi.ac.uk İnternet Kaynağı	%2
2	accord.googlecode.com İnternet Kaynağı	%2
3	R.M. Banakar. "Throughput Efficient Parallel Implementation of SPIHT Algorithm", 21st International Conference on VLSI Design (VLSID 2008), 01/2008 Yayın	%1
4	sundoc.bibliothek.uni-halle.de İnternet Kaynağı	%1
5	Submitted to Anadolu University Öğrenci Ödevi	%1
6	polen.itu.edu.tr İnternet Kaynağı	%1
7	Kimpe, T., T. Bruylants, Y. Sneyders, R. Deklerck, P. Schelkens, and Joseph M. Reinhardt. "", Medical Imaging 2007 Image	<%1

Dr. Öğr. Üyesi Ali AKUAN

Danışman

TEŞEKKÜR

Bu tez çalışmasının seçiminde, yürütülmesinde ve sonuçlandırılmasında destek ve yardımlarını esirgemeyen danışman hocam Sayın Dr. Öğr. Üyesi Ali AKMAN'a teşekkür ederim.

Vermiş olduğu desteğinden dolayı sevgili eşime sevgilerimi sunarım.

Ali DERİNOĞLU

Mayıs 2018

ÖZ

KABLOSUZ MULTİMEDYA ALGILAYICI AĞLARDA VERİMLİ GÖRÜNTÜ SIKIŞTIRMA VE İLETİM TEKNİKLERİ

Ali DERİNOĞLU
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Dr. Öğr. Üyesi Ali AKMAN
Maltepe Üniversitesi Fen Bilimleri Enstitüsü, 2018

Günümüzde yarıiletken teknolojisinin ilerlemesi ve güç tüketiminin azalması ile birlikte kablosuz algılayıcı ağların kullanımı oldukça yaygınlaşmıştır. Kablosuz algılayıcı ağlar ile ısı, ışık, nem gibi fiziksel büyüklükler ölçülerek mevcut ortam hakkında bilgi elde edilebilmektedir.

Kablosuz algılayıcı ağlar ile mevcut ortam hakkında daha detaylı bilgi edinebilmek için, düğüm noktalarına görsel ve işitsel algılayıcılar ekleyerek kablosuz multimedya algılayıcı ağlar oluşturulmaktadır. Görüntü ve ses gibi büyük verilerin, kısıtlı enerji kaynağı ve bant genişliğine sahip olan kablosuz algılayıcı ağlarda iletilmesi için verimli bir sıkıştırma ve iletim yöntemi ihtiyacı karşımıza çıkmaktadır.

Bu tez çalışmasında SPIHT görüntü sıkıştırma algoritması incelenmiş ve kameradan alınan görüntülerin verimli şekilde iletildiği bir kablosuz multimedya algılayıcı ağ oluşturulmuştur.

Anahtar Sözcükler: Kablosuz algılayıcı ağlar, Dalgacık dönüşümü, Görüntü sıkıştırma, Görüntü iletimi, SPIHT

ABSTRACT

EFFICIENT IMAGE COMPRESSION AND TRANSMISSION TECHNIQUES IN WIRELESS MULTIMEDIA SENSOR NETWORKS

Ali DERİNOĞLU
Master Thesis
Computer Engineering Department
Thesis Advisor: Assist. Prof. Dr. Ali AKMAN
Maltepe University Science and Engineering Graduate School, 2018

Nowadays, with the progress of semiconductor technology and the reduction of power consumption, the use of wireless sensor networks has become widespread. With wireless sensor networks, physical quantities such as temperature, light, humidity can be measured and information about the current environment can be obtained.

In order to obtain more detailed information about the current environment, wireless multimedia sensor networks are created by adding visual and auditory sensors to node points. There is a need for an efficient compression and transmission method for transmitting information containing large amounts of data such as video and audio to wireless sensor networks having limited energy resources and bandwidth.

In this thesis, SPIHT image compression algorithm have been analyzed and a wireless multimedia sensor network has been implemented in which the images taken from the camera are transmitted efficiently.

Keywords: Wireless sensor networks, Wavelet transform, Image compression, Image transmission, SPIHT

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR

α : Alfa

β : Beta

γ : Gamma

δ : Delta

DCT : Discrete Cosine Transform

DWT : Discrete Wavelet Transform

EZW : Embedded Zerotree Wavelet

SPIHT : Set Partitioning in Hierarchical Trees

PSNR : Peak Signal to Noise Ratio

WSN : Kablosuz Algılayıcı Ağlar

WMSN : Kablosuz Multimedya Algılayıcı Ağlar

MSE : Mean Squared Error

MSB : Most Significant Bit

LSB : Least Significant Bit

bpp : Bit per pixel

ÖZGEÇMİŞ

Ali DERİNOĞLU

Eğitim

Derece	*Yıl*	*Üniversite, Enstitü, Anabilim/Anasanat Dalı*
Y.Ls.	....	Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı
Ls.	2013	Anadolu Üniversitesi, İşletme Fakültesi, İşletme Anabilim Dalı
Ön.Ls.	2003	Kocaeli Üniversitesi, Endüstriyel Elektronik
Lise	2001	Körfez Teknik Lisesi, Elektronik

İş/İstihdam

Yıl	*Görev*
2015 -	Gömülü Yazılım Uzmanı, Aktifbank A.Ş.
2013 - 15	Gömülü Yazılım Uzmanı, Açıkkart A.Ş.
2008 - 12	Gömülü Yazılım Uzmanı, Asis A.Ş.

Mesleki Birlik/Dernek Üyelikleri

Alınan Burs ve Ödüller

Yayınlar ve Diğer Bilimsel/Sanatsal Faaliyetler

Kişisel Bilgiler

Doğum yeri ve yılı : Çorum 1983
Yabancı diller : İngilizce

BÖLÜM 1. GİRİŞ

Kablosuz algılayıcı ağlar bulunduğu ortamdaki ısı, ışık, nem gibi fiziksel ya da çevresel koşulları iletmek ve uzaktan izlemek için kullanılan ağlardır. Fiziksel bir iletim altyapısı gerektirmemesi kurulum maliyetini oldukça azaltmaktadır. Tekrarlayıcı düğüm noktaları kullanarak çok uzun mesafelerdeki ağlara ulaşmak mümkündür. Bu avantajlarından dolayı kablosuz algılayıcı ağların kullanımını oldukça yaygınlaştırmıştır.

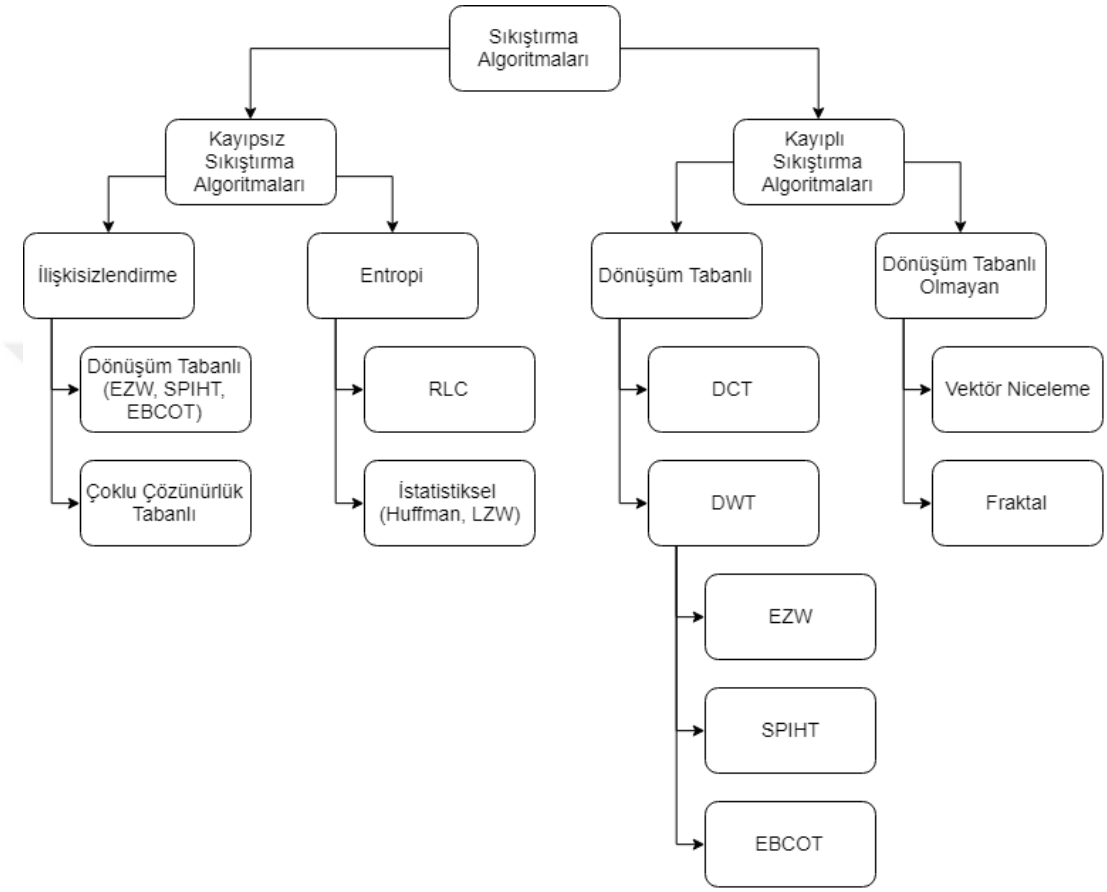
Bazı ortamlarda ısı, ışık, nem gibi sadece sayısal verilerin elde edilmesi ve gözlemlenmesi yeterli olmamaktadır. Bulunulan ortam hakkında daha detaylı bilgi edinme ihtiyacı ortaya çıkmaktadır. Örneğin ana yollarda bulunan ısı, yağmur ve hız algılayıcıları bize ortamın iklimi ve araçların hareket bilgisini verebilmektedir. Bu algılayıcılar ile araç hareketlerinin azaldığı bir bölgede sadece trafiğin yoğunlaştığı bilgisi elde edilebilir, ancak bu yoğunluğun neden kaynaklandığı tam olarak belirlenemez.

Sayısal ortam verilerinin iletilmesinde oldukça başarılı olan kablosuz algılayıcı ağların düğüm noktalarına görsel ve işitsel algılayıcılar eklenerek kablosuz multimedya algılayıcı ağlar oluşturulur. Kablosuz multimedya algılayıcı ağlar sayısal veriler dışında bize ortam hakkında görsel ve işitsel veriler sunarak daha detaylı bilgi elde edebilmemizi sağlar. Yukarıda verilen örnekte araç hareketlerinin azaldığı ve trafiğin yoğunlaştığı bir bölgede görsel algılayıcılardan alınan veriler ile yoğunluk hakkında daha detaylı bilgi edinilebilir. Örneğin trafik kazası, yol üzerinde araç hareketini engelleyen cisim gibi detaylar kablosuz multimedya algılayıcı ağlar ile elde edilebilir.

1.1 Sıkıştırma Algoritmaları

Büyük veri içeren bilgiler, iletim ortamının bant genişliğini daha verimli kullanmak için iletimden önce sıkıştırılır ve gönderilecek verinin boyutu düşürülür. Alıcı tarafında ise sıkıştırılmış halde alınan veri açılarak sıkıştırmadan önceki hali elde edilir. Sıkıştırma işlemi verimli iletim dışında verinin saklanması için gereken alandan da tasarruf sağlar.

İlerleyen teknoloji ile amaca uygun farklı sıkıştırma algoritmaları geliştirilmiştir. Sıkıştırılacak verinin türüne ve içeriğine göre kullanılacak sıkıştırma yöntemi seçilmelidir. Aşağıdaki şekilde yaygın olarak kullanılan sıkıştırma algoritmalarının sınıflandırılması gösterilmiştir.



Şekil 1.1 - Sıkıştırma Algoritmaları [1]

Kayıpsız sıkıştırma algoritmaları sıkıştırılacak verinin veri kaybı olmadan yeniden oluşturulması gereken, veri bütünlüğünün önemli olduğu uygulamalarda tercih edilir. Kayıpsız sıkıştırma sağladığından sıkıştırma oranı düşüktür. Görüntü sıkıştırma işlemlerinde ise geri dönüştürülen görüntünün kayıpsız olmasından ziyade insan gözünün fark edemeyeceği kalite kaybı ile en yüksek sıkıştırma oranı elde edilmesi istenir. Bu sebeple görüntü sıkıştırma işlemlerinde sıkıştırma oranı daha yüksek olan kayıplı sıkıştırma algoritmaları tercih edilir [2].

1.2 Literatür Araştırması

Görüntü sıkıştırma ile ilgili yapılan çalışmalar incelendiğinde ilk çalışmaların Ayrık Kosinüs Dönüşüm (DCT) tabanlı olduğu görülmektedir. Bu çalışmalardan en popüler olanı ve en yaygın kullanılanı 1992 yılında yayınlanan JPEG formatıdır [3].

Ayrık kosinüs dönüşümünde kullanılan trigonometrik fonksiyonların işlem karmaşıklığını arttırması ve sıkıştırma için gereken işlem süresini uzatması enerji tüketimini de arttırmaktadır. Bu sebeple kablosuz algılayıcı ağlar gibi sınırlı enerji kaynağına sahip olan uygulamalarda daha az işlem karmaşıklığı dolayısıyla daha hızlı sıkıştırma sağlayan "Ayrık Dalgacık Dönüşümü"nün (DWT) sıkıştırma performansı ve enerji tüketimi açısından daha verimli olduğu görülmüştür [4].

Ayrık Dalgacık Dönüşümünü kullanan ilk görüntü sıkıştırma algoritması olan "Gömülü Sıfır-Ağaç Dalgacık Dönüşümü" (EZW) 1993 yılında J.Shapiro tarafından yayınlanmıştır [5]. JPEG'e göre daha performanslı sıkıştırma sağlamasının yanında, sıkıştırma yapmadan önce görüntüyü bloklara bölmediği için bloklar arası geçişte yaşanan "blok etkisi" oluşmamaktadır [6].

1996 yılında Said ve Pearlman tarafından "Hiyerarşik Ağaçlarda Küme Bölümleme" (SPIHT) yayınlanmıştır [7]. Temel olarak "Gömülü Sıfır-Ağaç Dalgacık Dönüşümü"ne dayanmaktadır. EZW'den farklı olarak listeler kullanarak daha büyük altkümelerdeki önemsiz katsayıları bir arada tutma eğilimi ile çalışır. Bu sayede EZW'ye göre daha yüksek sıkıştırma oranı ve daha iyi PSNR değeri verir [8].

SPIHT algoritması sıkıştırma işlemi sırasında kullandığı listeler sebebiyle yoğun dinamik bellek kullanımı gerektirmektedir. Liste boyutlarının sıkıştırma işlemi sırasında sıkıştırılacak görüntü içeriğine göre dinamik olarak değişmesi ve gerekli liste boyutunun sıkıştırma işlemi öncesinde belirlenememesi SPIHT algoritmasının donanım olarak tasarlanmasını zorlaştırmaktadır. Bu zorlukları aşmak için 2000 yılında Wheeler ve Pearlman tarafından liste kullanmayan SPIHT algoritması (No List SPIHT) yayınlanmıştır [9].

2002 yılında Ritter, Fey ve Molitor tarafından "No List SPIHT" (NLS) algoritması donanım (FPGA) üzerinde uygulanarak NLS'ye göre %23 daha az bellek kullanımı ve paralel işlem yapılması sağlanmıştır [10].

1.3 Problemin Tanımı

Kablosuz algılayıcı ağlar kısıtlı enerji kaynağı ile çalışmaktadır. Buna bağlı olarak işlem kapasiteleri ve bant genişlikleri de düşüktür. Sıkıştırmada kullanılan algoritmanın karmaşıklığı ve bellek kullanımı arttıkça enerji tüketimi de artmaktadır. Sıkıştırma oranı düşük ise sıkıştırılmış görüntünün boyutu da büyük olacağı için bu verinin iletimi de daha uzun sürecektir. Daha uzun iletim süresi de daha fazla enerji tüketimi demektir.

SPIHT algoritması dalgacık dönüşümü tabanlı olması, yüksek kalite ve yüksek sıkıştırma oranı sağlaması sebebiyle kablosuz multimedya algılayıcı ağlara uygun gözükmektedir. Bu çalışmada SPIHT algoritmasının kablosuz multimedya algılayıcı ağlarda uygulanması hedeflenmektedir.

Orijinal SPIHT algoritmasının yoğun dinamik bellek kullanımı nedeniyle işlem süresi uzamakta ve yüksek enerji tüketimine sebep olmaktadır. Orijinal SPIHT algoritması değiştirilip dinamik bellek kullanımı ve işlem süresi azaltılarak enerji tüketiminin düşürülmesi amaçlanmaktadır.

Sıkıştırma işlemi sonrası elde edilen kodlamanın verimli bir şekilde iletilmesini sağlayan protokol geliştirilecektir.

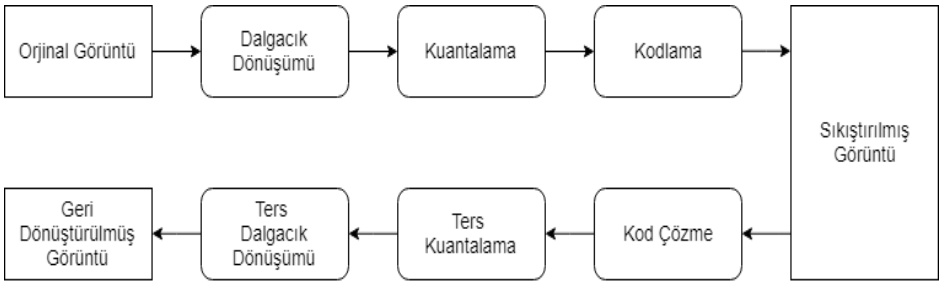
BÖLÜM 2. YÖNTEM

Bu bölümde öncelikle görüntü sıkıştırmada kullanılacak olan dönüşüm tabanlı sıkıştırma ve dalgacık dönüşümü incelenecek, ardından kodlama için kullanılacak olan SPIHT ve iyileştirilmiş listesiz SPIHT algoritmaları tanıtılacaktır.

2.1 Dönüşüm Tabanlı Sıkıştırma

Bir görüntünün asıl anlam ifade eden verileri düşük frekanslı sinyallerde bulunur, yüksek frekanslı sinyaller ise ayrıntıyı içerir. İnsan gözü yüksek frekanslı sinyallere daha az duyarlı olduğu için görüntü sıkıştırma işlemi ağırlıklı olarak bu yüksek frekanslı sinyaller üzerinde yapılır [11].

Dönüşüm tabanlı sıkıştırmada görüntü sinyali öncelikle dalgacık dönüşümü ile zaman uzayından frekans uzayına dönüştürülür ve katsayılar elde edilir. Bu katsayılardan yüksek frekanslı olanlar kodlanmayarak sıkıştırma elde edilmiş olur. Dönüşüm tabanlı bir görüntü sıkıştırma ve çözme işleminin aşamaları Şekil 2.1'de görülmektedir.



Şekil 2.1 - Dönüşüm Tabanlı Görüntü Sıkıştırma Aşamaları [12]

2.2 Dalgacık Dönüşümü

Fourier dönüşümü ile bir sinyalin frekans bileşenleri elde edilebilmektedir. Analiz edilecek sinyal durağan olmayan bir sinyal ise sadece frekans bilgisi yeterli olamamaktadır. Durağan olmayan bir sinyale Fourier dönüşümü uygulandığında frekans bileşenleri elde edilmekte ancak frekans bileşeninin ortaya çıktığı zaman bilgisi kaybolmaktadır. Bu gibi durumlarda Fourier dönüşümü yetersiz kalmaktadır [13].

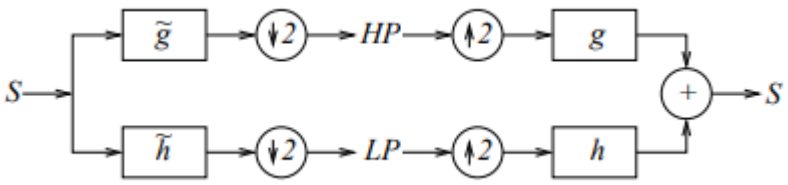
Fourier dönüşümün bu eksikliğini gidermek için 1981 yılında J. Morlet tarafından dalgacık dönüşümü yöntemi ortaya koyulmuştur. Dalgacık dönüşümü ile sinyalin frekans bilgileri elde edilmekle birlikte, frekansların zaman bilgileri de korunmaktadır.

Fourier dönüşümünde sinyal sinüs fonksiyonunun farklı frekansları cinsinden ifade edilir. Dalgacık dönüşümünde ise sinyal ana dalgacığın (mother wavelet) belirli bir ölçekte ve zamanda kaydırılmasıyla elde edilir [14].

2.2.1 Ayrık Dalgacık Dönüşümü

Dalgacık dönüşümü sırasında ana dalgacığın olası tüm ölçekleri ve sinyalin tüm zaman aralığı kullanılırsa hesaplanan katsayılar çok büyük veri yığınları oluşmasına sebep olur. Bu katsayıların tümü üzerinde işlem yapmak zorlaşır. Bu zorluğu aşmak için belirli ölçek ve konum aralıklarında dönüşüm yapılır. Ölçek ve konum değerlerinin ikinin kuvvetleri olacak şekilde seçilmesi dönüşüm işlemini en hızlı şekilde olmasını sağlayacaktır. Bu işleme ayrık dalgacık dönüşümü denir [14].

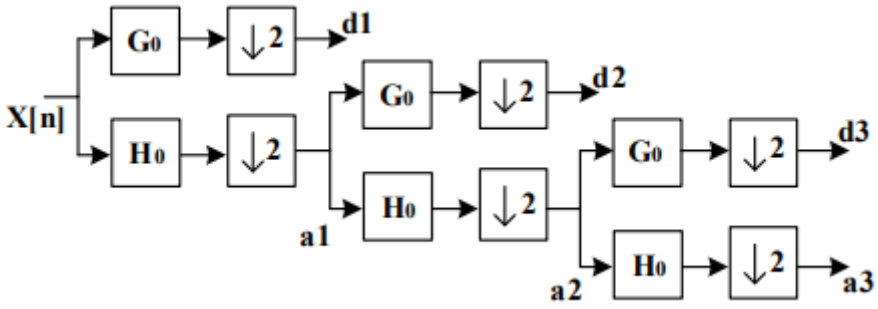
Ayrık dalgacık dönüşümünün uygulanması için verimli bir yöntem filtreler kullanılarak 1988 yılında Mallat tarafından geliştirilmiştir [15]. Bu yöntemde sinyal (S) eş zamanlı olarak alçak geçiren (h) ve yüksek geçiren (g) filtrelerden geçirilir. Alçak geçiren filtrenin çıktısı sinyal hakkında yaklaşık (approximation) bilgiyi verirken, yüksek geçiren filtre çıktısı ise detay (detail) bilgisini verir. Filtrelerden çıkan sinyalin örnek sayısı alt örnekleme (downsampling) yapılarak yarıya düşürülür. Sonuç olarak giriş sinyalinin yarısı kadar düşük frekanslı (LP) ve yüksek frekanslı (HP) örnekler elde edilir. Şekil 2.2'de filtre bankası kullanılan ayrık dalgacık dönüşümü görülmektedir.



Şekil 2.2 - Filtre Bankası ile Ayrık Dalgacık Dönüşümü [16]

Ters dönüşüm için ilk olarak HP ve LP sinyalleri her örnek arasına sıfırlar koyularak üst örnekleme (upsampling) yapılır. Bundan sonra alçak geçiren (h) ve yüksek geçiren (g) filtrelerden geçirilerek filtrelenir ve sonuçlar birleştirilir. Kayıpsız geri dönüşüm durumunda elde edilen sinyal orijinal S sinyaline eşittir [16].

Yukarıdaki ayrık dalgacık dönüşümünde filtreleme ve alt örnekleme işlemi sadece bir kez yapılmıştır. Filtreleme ve alt örnekleme işlemi yaklaşık bilgiyi veren alçak geçiren filtre (h) çıkışında ardışık olarak devam ettirilirse, sinyalin asıl anlam ifade eden düşük frekanslı bileşenlerinin daha detaylı analizi yapılmış olur. Bu işlem Çoklu Çözünürlük Analizi (Multiresolution Analysis) olarak adlandırılmaktadır. Şekil 2.3'de 3 seviyeli bir ayrık dalgacık dönüşümü görülmektedir.

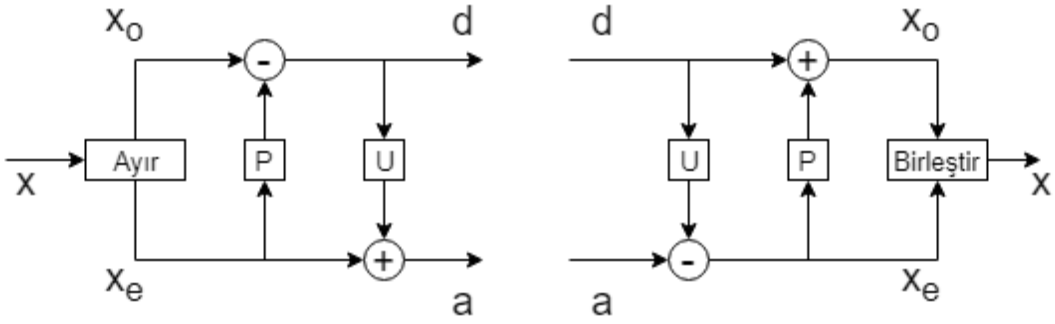


Şekil 2.3 - 3 Seviyeli Ayrık Dalgacık Dönüşümü [17]

2.2.2 Lifting Tabanlı Ayrık Dalgacık Dönüşümü

Geleneksel dalgacık dönüşümü işlemlerinde filtreler ve alt örnekleme kullanılmaktadır. Filtrelerde kullanılan karmaşık matematik işlemlerini azaltmak için 1996 yılında Sweldens tarafından Lifting yöntemi geliştirilmiştir. Lifting yöntemi dalgacık dönüşümü için en basit ve en verimli yöntemdir [17].

Lifting yönteminde sinyal tek (odd) ve çift (even) örneklerine ayrılır. Filtreler yerine daha basit olan kestirim (prediction) ve güncelleme (update) işlemleri uygulanır. Bu işlemler için geleneksel yöntemlerdeki gibi karmaşık hesaplamalara gerek duyulmaz. Şekil 2.4'de Lifting dalgacık dönüşümü ve ters dönüşümü görülmektedir.



Şekil 2.4 - Lifting Dalgacık Dönüşümü ve Ters Dönüşüm [18]

Orijinal x sinyali öncelikle denklem 2.1 ve 2.2'deki gibi tek ve çift bileşenlerine ayrılır.

$$x_e = x_{2k} \tag{2.1}$$

$$x_o = x_{2k+1} \tag{2.2}$$

Bu iki örnek arasında güçlü bir bağıntı vardır. Kestirim (predict) adımında tek (odd) örnekler çift (even) örneklerden yararlanılarak yaklaşık olarak elde edilmeye çalışılır. Bu adım yüksek geçiren filtre görevi görür ve sinyalin yüksek frekanslı bileşenleri (d) elde edilir.

$$d = x_o - P(x_e) \tag{2.3}$$

Güncelleme adımında (update) örnekleri düzeltmek için ölçeklendirme yapılır ve daha sonra dönüşüm için alçak geçiren filtreli değerleri sağlamak üzere çift örneklerle eklenir. Bu örnekler sinyal hakkında yaklaşık bilgiyi (a) veren düşük frekanslı bileşenleri içerir.

$$a = x_e + U(d) \tag{2.4}$$

Elde edilen d ve a sinyallerine dönüşümde kullanılan işlemlerin tam tersi uygulanarak ters dönüşüm elde edilir.

$$x_o = d + P(x_e) \tag{2.5}$$

$$x_e = a - U(d) \tag{2.6}$$

2.2.3 Dalgacık Aileleri

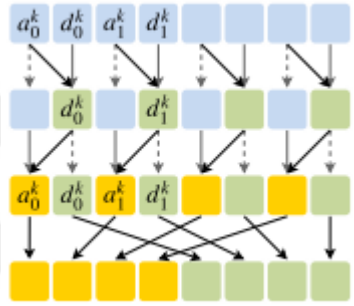
Dalgacık dönüşümü işleminde farklı dalgacık aileleri kullanılabilmektedir. Görüntü sıkıştırmada sıkça kullanılan dalgacıklar Haar, CDF 5/3 ve CDF 9/7 dalgacıklarıdır.

Her dalgacık ailesi için öncelikle giriş sinyali denklem 2.7 ve 2.8'de görüldüğü gibi tek ve çift bileşenlerine ayrılır.

$$a^k[n] = x^{k-1}[2n] \tag{2.7}$$

$$d^k[n] = x^{k-1}[2n+1] \tag{2.8}$$

Haar dalgacık dönüşümü Şekil 2.5'de görülmektedir. Haar dalgacık dönüşümünde örnekler ikili gruplar halinde işleme alınır.



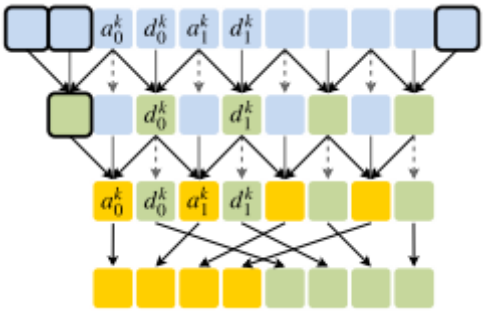
Şekil 2.5 - Haar Dalgacık Dönüşümü [19]

Örneklere P (predict) ve U (update) işlemleri uygulanarak örneklerin detay (d) ve yaklaşık (a) bileşenleri elde edilir. Son olarak detay (d) ve yaklaşık (a) bileşenler kendi aralarında gruplanarak işlem tamamlanır. Haar dalgacık dönüşümü hesaplamaları denklem 2.9 ve 2.10'da verilmiştir.

$$(L_P^{\alpha}): d_j^k = d_j^k - \alpha\left(a_j^k\right) \tag{2.9}$$

$$\left(L_U^{\beta}\right): a_j^k = a_j^k + \beta\left(d_j^k\right) \tag{2.10}$$

CDF 5/3 dalgacık dönüşümü Şekil 2.6'da görülmektedir. CDF 5/3 dalgacık dönüşümünde detay (d) ve yaklaşık (a) bileşenler hesaplanırken ikili gruplar haricinde komşu örneklerden de yararlanılır. P (predict) ve U (update) işlemleri sonrası detay (d) ve yaklaşık (a) bileşenler kendi aralarında gruplanarak dönüşüm tamamlanır.



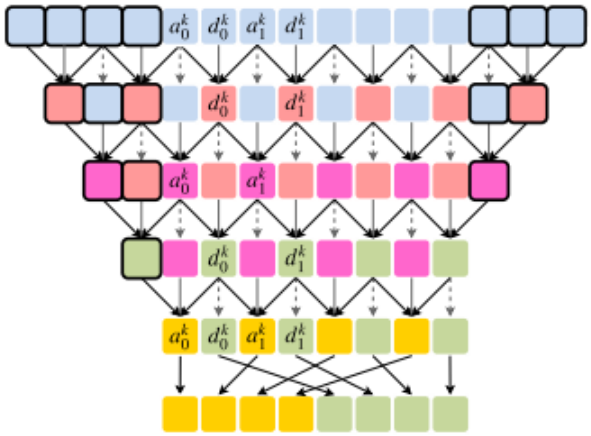
Şekil 2.6 - CDF 5/3 Dalgacık Dönüşümü [19]

CDF 5/3 dalgacık dönüşümü hesaplamaları denklem 2.11 ve 2.12'de verilmiştir.

$$(L_P^{\alpha}): d_j^k = d_j^k - \alpha\left(a_j^k + a_{j+1}^k\right) \tag{2.11}$$

$$\left(L_U^{\beta}\right): a_j^k = a_j^k + \beta\left(d_j^k + d_{j+1}^k\right) \tag{2.12}$$

CDF 9/7 dalgacık dönüşümü Şekil 2.7'de görülmektedir. CDF 9/7 dalgacık dönüşümü de CDF 5/3'e benzer şekilde komşu örnekleri kullanmaktadır. CDF 5/3'den farklı olarak örneklere ikinci bir P (predict) ve U (update) adımı uygulanmaktadır.



Şekil 2.7 - CDF 9/7 Dalgacık Dönüşümü [19]

CDF 9/7 dalgacık dönüşümü hesaplamaları denklem 2.13, 2.14, 2.15 ve 2.16'da verilmiştir.

$$(L_P^{\alpha}): d_j^k = d_j^{k-1} - \alpha\left(a_j^k + a_{j+1}^k\right) \tag{2.13}$$

$$\left(L_U^{\beta}\right): a_j^k = a_j^{k-1} + \beta\left(d_j^k + d_{j+1}^k\right) \tag{2.14}$$

$$\left(L_P^{\gamma}\right): d_j^k = d_j^{k-1} - \gamma\left(a_j^k + a_{j+1}^k\right) \quad (2.15)$$

$$\left(L_U^{\delta}\right): a_j^k = a_j^{k-1} + \delta\left(d_j^k + d_{j+1}^k\right) \quad (2.16)$$

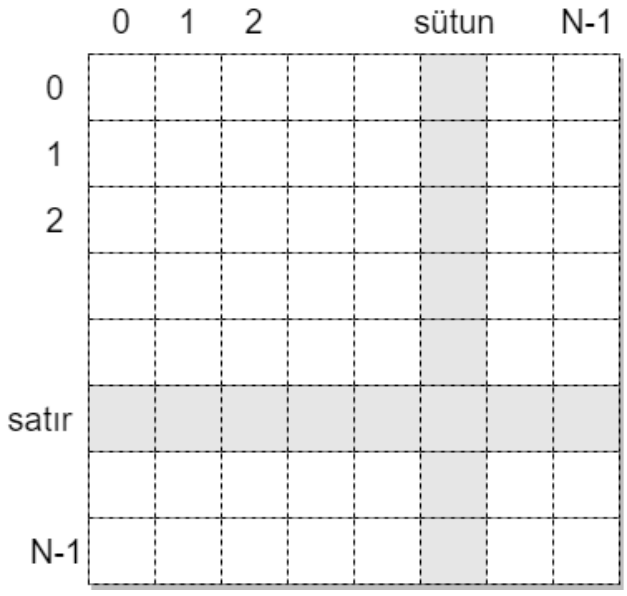
Haar, CDF 5/3 ve CDF 9/7 dalgacık dönüşümü hesaplamalarında kullanılan Lifting katsayıları Tablo 2.1'de verilmiştir.

Tablo 2.1 - Lifting Katsayıları [19]

	Haar	**CDF 5/3**	**CDF 9/7**
$\boldsymbol{\alpha}$	-1.0	-0.50	-1.58613434
$\boldsymbol{\beta}$	+0.5	+0.25	-0.05298012
$\boldsymbol{\gamma}$	-	-	+0.88291108
$\boldsymbol{\delta}$	-	-	+0.44350685

2.2.4 Dalgacık Dönüşümünün Görüntü Üzerine Uygulanması

Şu ana kadar görülen dalgacık dönüşümleri tek boyutlu sinyal üzerinde uygulanmaktadır. Ancak görüntü verisi iki boyutludur. İki boyutlu görüntü üzerinde dalgacık dönüşümü uygulamak için, tek boyutlu dalgacık dönüşümü satır ve sütunlar üzerinde ayrı ayrı uygulanır. Şekil 2.8'de boyutları NxN olan bir görüntü görülmektedir.



Şekil 2.8 - NxN Boyutunda Görüntü Satır ve Sütunları

Satırlar yukarıdan aşağıya, sütunlar soldan sağa sıfırdan başlayarak N-1'e kadar numaralandırılır. Öncelikle satırlar için 0'dan başlayarak N-1'e kadar dalgacık dönüşümü uygulanır. İlk dönüşüm sonucunda görüntünün sol yarısında düşük frekanslı bileşenler (L) yani görüntünün yaklaşık içeriği elde edilir. Görüntünün sağ yarısı ise görüntünün detayı olan yüksek frekanslı bileşenleri (H) içerir. Bu işlem Şekil 2.9'da görülmektedir.

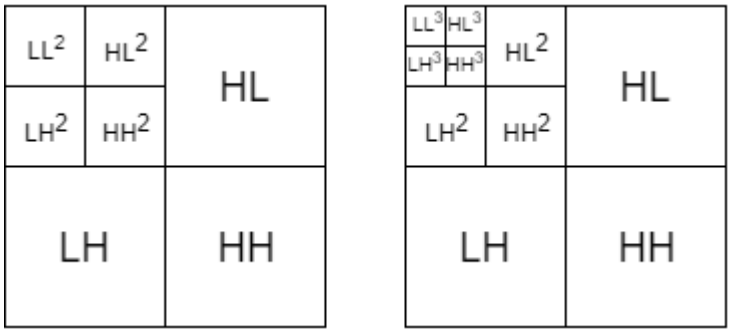
Şekil 2.9 - Dalgacık Dönüşümünün Satırlara Uygulanması

Dönüşümün ikinci aşamasında ise, elde edilen görüntüye bu kez sütunlar için 0'dan başlayarak N-1'e kadar dalgacık dönüşümü uygulanır. Bu işlem sonucunda görüntünün LL, HL, LH ve HH alt bantları elde edilir. Bu işlem Şekil 2.10'da görülmektedir.

Şekil 2.10 - Dalgacık Dönüşümü Sonrası Elde Edilen Alt Bantlar

Bu aşamaya kadar yapılan işlem bir seviyeli dalgacık dönüşümü analizi olarak adlandırılır. Dalgacık dönüşümü görüntünün düşük frekanslı bileşenlerini yani yaklaşık içeriği barındıran LL bandında N/2 satır ve sütuna kadar aynı şekilde devam ettirilebilir. Her yeni seviyede dalgacık dönüşümü yapılacak satır ve sütun sayısı yarıya indirilerek

LL bandında sadece bir katsayı kalana kadar dönüşüm yapılabilir. Şekil 2.11'de iki ve üç seviyeli dalgacık dönüşümü sonrası elde edilebilecek alt bantlar görülmektedir.



Şekil 2.11 - 2 ve 3 Seviyeli Dalgacık Dönüşümü

Dalgacık dönüşümü uygulanabilmesi için kaynaktan alınan görüntünün satır/sütun sayısı eşit (NxN) ve ikinin kuvveti olmalıdır. Bu şartlar sağlanmıyor ise öncelikle görüntü *upScaleImage* fonksiyonu ile bir üst seviyeye ölçeklendirilir.

```
01: Image *upScaleImage(Image *im)
02: {
03:    int max = MAX(im->cols, im->rows);
04:    int d = pow(2, ceil(log((float)max)/log((float)2)));
05:
06:    Image *img = new Image(d, d);
07:    img->org_cols = im->cols;
08:    img->org_rows = im->rows;
09:
10:    for (int c = 0; c < im->cols; c++)
11:           for (int r = 0; r < im->rows; r++)
12:                  img->at(c, r) = im->at(c, r);
13:
14:    return img;
15: }
```

Şekil 2.12 - upScaleImage Fonksiyonu

upScaleImage fonksiyonunda NxN görüntü boyutunu sağlayabilmek için MAX fonksiyonu ile satır ve sütun sayılarından büyük olanın değer alınır ve bu değerin iki tabanında logaritması bulunur. Bulunan değer ikinin tam kuvveti değil ise *ceil* fonksiyonu ile bir üst kuvvete yuvarlanır ve yeni NxN satır/sütun boyutu elde edilir. Bu boyutlar ile yeni bir *Image* nesnesi oluşturulur. Orijinal görüntünün kendi boyutu içerisindeki piksel bilgileri oluşturulan yeni nesneye kopyalanır. Yeni *Image* nesnesi

oluşturulurken tüm piksellerin içeriği sıfırlandığı için yeni görüntünün orijinal görüntü dışındaki pikselleri sıfır olarak kalır. Bu işlem Şekil 2.12'deki kod parçası ile yapılmaktadır.

Dalgacık dönüşümüne uygun hale getirilen görüntüye *dwt_img* fonksiyonu ile LL alt bandında sadece bir katsayı kalana kadar yani satır ve sütun sayısı bir olana kadar iki boyutlu dalgacık dönüşümü uygulanır. Bu işlem Şekil 2.13'deki kod parçası ile yapılmıştır.

```
01: int dwt_img(Image *m)
02: {
03:     int size_x = m->cols;
04:     int size_y = m->rows;
05:
06:     int i, level = INT_MAX;
07:
08:     for (i = 0; i < level; i++) {
09:             dwt2d(m, size_x, size_y);
10:             size_x /= 2;
11:             size_y /= 2;
12:
13:             if ((size_x == 1) || (size_y == 1))
14:                     break;
15:     }
16:
17:     return i + 1;
18: }
```

Şekil 2.13 - dwt_img Fonksiyonu

Her dalgacık dönüşümü seviyesinde bir önceki seviyenin yarısı kadar satır ve sütuna *dwt2d* fonksiyonu ile iki boyutlu dalgacık dönüşümü uygulanır. Satır ve sütunlara ayrı ayrı dalgacık dönüşümü uygulayan kod parçası Şekil 2.14'de görülmektedir.

```
01: void dwt2d(Image *m, int size_x, int size_y)
02: {
03:     for (int y = 0; y < size_y; y++)
04:             dwt_row(m, y, size_x);
05:     for (int x = 0; x < size_x; x++)
06:             dwt_col(m, x, size_y);
07: }
```

Şekil 2.14 - dwt2d Fonksiyonu

dwt_row ve *dwt_col* fonksiyonları kullanılarak satır ve sütunlara istenen dalgacık ailesi ile (Haar, CDF 5/3, CDF 9/7) dalgacık dönüşümü uygulanır. Şekil 2.15 ve Şekil 2.16'da bu fonksiyonlar görülmektedir.

```
01: void dwt_row(Image *m, int row, int num_items)
02: {
03:    int *temp = new int[num_items];
04:
05:    for (int i = 0; i < num_items; i++)
06:           temp[i] = m->getPixel(row, i);
07:
08:    if (DWT_TYPE == HAAR)
09:           fwt_HAAR(temp, num_items);
10:    else if (DWT_TYPE == CDF97)
11:           fwt_cdf97(temp, num_items);
12:    else if (DWT_TYPE == CDF53)
13:           fwt_cdf53(temp, num_items);
14:
15:    for (int i = 0; i < num_items; i++)
16:           m->setPixel(row, i, temp[i]);
17:
18:    delete[] temp;
19: }
```

Şekil 2.15 - dwt_row Fonksiyonu

```
01: void dwt_col(Image *m, int col, int num_items)
02: {
03:    int *temp = new int[num_items];
04:
05:    for (int i = 0; i < num_items; i++)
06:           temp[i] = m->getPixel(i, col);
07:
08:    if (DWT_TYPE == HAAR)
09:           fwt_HAAR(temp, num_items);
10:    else if (DWT_TYPE == CDF97)
11:           fwt_cdf97(temp, num_items);
12:    else if (DWT_TYPE == CDF53)
13:           fwt_cdf53(temp, num_items);
14:
15:    for (int i = 0; i < num_items; i++)
16:           m->setPixel(i, col, temp[i]);
17:
18:    delete[] temp;
19: }
```

Şekil 2.16 - dwt_col Fonksiyonu

Dönüşümde kullanılan dalgacık ailelerinin (Haar, CDF 5/3, CDF 9/7) fonksiyonları Ek-1'de verilmiştir. Dalgacık dönüşümü sonrası elde edilen dalgacık katsayılarına ters dönüşüm uygulanarak tekrar dönüşüm öncesi görüntü elde edilir.

2.3 SPIHT

SPIHT algoritması ayrık dalgacık dönüşümü sonrasında elde edilen katsayıların genliğine göre sıralanması ve alt bantlardaki aynı uzaysal düzlemde bulunan katsayıların benzerliğinden faydalanılarak kümelenmesi temeline dayanmaktadır. Ürettiği gömülü bit akışı sayesinde kodlama ve kod çözme işlemleri istenilen noktada durdurulabilir. Kodlama ve kod çözme işlemi durdurulmadan sonuna kadar yapılırsa kayıpsız sıkıştırma elde edilmiş olur.

Kodlama işlemi öncesinde orijinal görüntüdeki dalgacık katsayıları c_{ij} ve kod çözme işlemi sonrasında elde edilen geri dönüştürülmüş görüntüdeki dalgacık katsayıları $\hat{c}_{ij}$ olmak üzere iki görüntü arasındaki gürültüyü ortalama karesel hata (MSE) ile denklem 2.17'deki gibi hesaplayabiliriz.

$$D_{MSE} = \frac{1}{N}\sum_i \sum_j \left(c_{ij} - \hat{c}_{ij}\right)^2 \tag{2.17}$$

Bu ifadeye göre kod çözücüye c_{ij} katsayısının kendisi veya en yakın değeri gönderilirse ortalama karesel hata $\frac{c_{ij}^2}{N}$ değerine göre azalmaktadır. Dolayısıyla elde edilen görüntüdeki gürültü değeri de azalmış olacaktır. Buradan verimli bir iletim için en büyük katsayıların önce iletilmesi gerekliliği ortaya çıkmaktadır [7].

SPIHT algoritması bunu gerçekleştirmek için katsayıları mutlak değeri üzerinden büyüklüklerine göre sıralar ve öncelikle katsayıların en yüksek değerli bitini (MSB) iletir. Kodlama işlemi iki aşamadan oluşur. Bunlar sıralama (sorting) aşaması ve iyileştirme (refinement) aşamasıdır. İlk adımda iyileştirme aşaması gerçekleşmez. Öncelikle denklem 2.18 ile katsayılardaki en büyük değer kullanılarak eşik değeri bulunur.

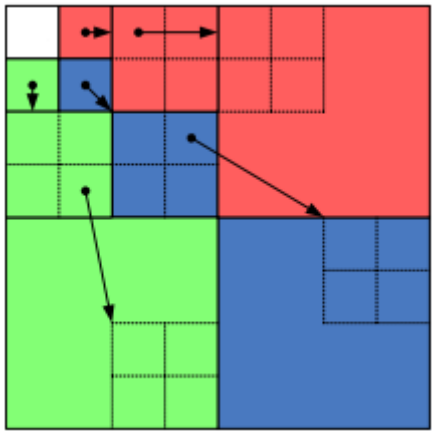
$$n = \left\lfloor log_2\left(max_{(i,j)}\{|c_{ij}|\}\right)\right\rfloor \tag{2.18}$$

Sıralama aşamasında $2^n \leq |c_{ij}| < 2^{n+1}$ ifadesini sağlayan katsayıların en yüksek değerli biti (MSB) ve işareti (sign) iletilir. İfadeyi sağlamayan katsayılar için sıfır gönderilir. Sıralama aşamasındaki önemlilik testi denklem 2.19'daki gibidir.

$$S_n = \begin{cases} 1, |c_{ij}| \geq 2^n \\ 0 \end{cases} \quad (2.19)$$

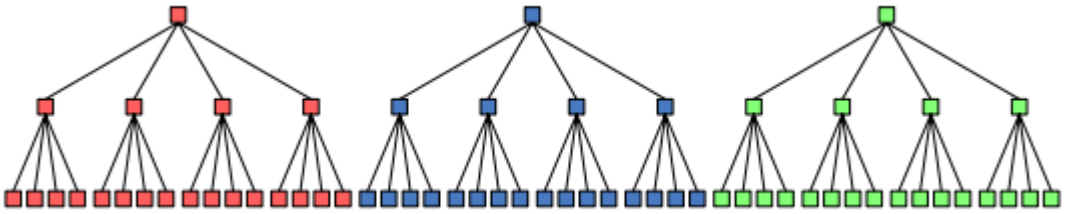
İyileştirme aşamasında $|c_{ij}| \geq 2^{n+1}$ ifadesini sağlayan yani bir önceki sıralama adımında eşik değerini geçen katsayıların mevcut eşik değerindeki bit değerleri iletilir. Eşik değeri (n) sıfır olana kadar veya istenilen eşik değerine ulaşıncaya kadar eşik değeri bir azaltılarak tekrar sıralama aşamasına geçilir.

SPIHT algoritması katsayıları taramaya en düşük frekanslı bileşenleri yani en yüksek katsayıları içeren LL bandından başlar. Her banttaki katsayılar bir alt banttaki katsayılar ile ilişkilidir. Yani önemlilik testi yapılırken bir banttaki katsayı önemlilik testini geçiyor ise, o katsayının alt bantlarındaki katsayılarının da önemlilik testini geçmesi yüksek olasılıklıdır. Katsayılar taranırken alt bantlar arasındaki bu ilişki de dikkate alınır. Katsayılar ve alt bantlar arasındaki ilişki Şekil 2.17'de görülmektedir.



Şekil 2.17 - Katsayılar ve Alt Bantlar Arasındaki İlişki [20]

LL bandında (0,0) koordinatındaki katsayı hariç her alt banttaki her katsayının doğrudan ilişkili 2x2 grup halinde 4 adet alt katsayısı bulunur. Bu yapı ağaç kullanımına oldukça uygun bir yapı olup, katsayılar ve alt bantlar arasındaki ilişki ağaç yapısı olarak Şekil 2.18'de görülmektedir.



Şekil 2.18 - Katsayılar ve Ağaç Yapısı [20]

Yukarıda anlatılan SPIHT görüntü kodlama işlemi için gerçeklenen algoritmanın sahte kodu Şekil 2.19'da verilmiştir. Bu algoritmada;

O(i,j): *(i,j)* koordinatındaki dalgacık katsayısının çocuklarının koordinatlarının kümesi
D(i,j): *(i,j)* koordinatındaki dalgacık katsayısının tüm neslinin koordinatlarının kümesi
H(i,j): en yüksek seviyeli alt bantta bulunan ağaç köklerinin koordinatlarının kümesi
L(i,j): *D(i,j)* - *O(i,j)* (i,j) koordinatındaki dalgacık katsayısının kendi çocukları hariç tüm neslinin koordinatlarının kümesidir.

Algoritmada kullanılan listeler ise;

LIP: Önemsiz dalgacık katsayılarının koordinatlarının listesi
LSP: Önemli dalgacık katsayılarının koordinatlarının listesi
LIS: Ağaç yapısındaki önemsiz dalgacık katsayı kümelerinin kök koordinatlarının listesidir.

1. İlklendirme

01: $n := \lfloor log_2(\max|C_{ij}|) \rfloor$

02: $LIP := H$

03: $LSP := \emptyset$

04: $LIS := (i,j) \in H \; ve \; D(i,j) \neq \emptyset$

2. Sıralama

05: ***Her*** $(i,j) \in LIP$ ***için***

06: $S_n(i,j)$ *değerini çıkış dizisine gönder*

07: ***Eğer*** $S_n(i,j) = 1$ ***ise***

08: (i,j) *yi* LSP'*ye taşı*

09: C_{ij} *nin işaret bitini çıkış dizisine gönder*

10: ***Her*** $(i,j) \in LIS$ ***için***

11: ***Eğer*** (i,j) *tip A* ***ise***

12: $S_n(D(i,j))$ *değerini çıkış dizisine gönder*

13: ***Eğer*** $S_n(D(i,j)) = 1$ ***ise***

14: ***Her*** $(k,l) \in O(i,j)$ ***için***

15: $S_n(k,l)$ *değerini çıkış dizisine gönder*

16: ***Eğer*** $S_n(k,l) = 1$ ***ise***

17: (k,l) *yi* LSP'*ye ekle*

18: C_{kl} *nin işaret bitini çıkış dizisine gönder*

19: ***Değil ise***

20: (k,l) *yi LIP sonuna ekle*

21: ***Eğer*** $L(i,j) \neq \emptyset$ ***ise***

22: (i,j) *yi tip B olarak* LIS'*e taşı*

23: *Adım* $10'a$ ***git***

24: ***Değil ise***

25: (i,j) *yi* LIS'*den çıkar*

26: ***Eğer*** (i,j) *tip B* ***ise***

27: $S_n(L(i,j))$ *değerini çıkış dizisine gönder*

28: ***Eğer*** $S_n(L(i,j)) = 1$ ***ise***

29: ***Her*** $(k,l) \in O(i,j)$ ***için*** *tip A olarak LIS sonuna ekle*

30: $(i,j)\ yi\ LIS'den\ çıkar$

3. İyileştirme

31: $\boldsymbol{Her}\ (i,j) \in LSP\ \boldsymbol{için}$

32: $\boldsymbol{Eğer}\ S_{n+1}(i,j) = 1\ \boldsymbol{ise}$

33: $C_{ij}\ nin\ n.bitini\ çıkış\ dizisine\ gönder$

4. Seviye Güncelleme

34: $n = 0\ \boldsymbol{olana\ kadar}\ n = n - 1\ yap\ ve\ adım\ 5'e\ \boldsymbol{git}$

Şekil 2.19 - SPIHT Algoritması

2.4 İyileştirilmiş Listesiz SPIHT

SPIHT algoritması kodlama ve kod çözme işlemleri sırasında yoğun bir şekilde dinamik bellek kullanımı gerektirir. Hangi dalgacık katsayılarına önemlilik testi yapılacağı ve önemlilik testi yapılmış dalgacık katsayılarını takip etmek için dinamik listeler kullanılmaktadır. Kullanılan dinamik listeler bellek yönetim karmaşıklığını arttırmakta ve algoritmanın donanım üzerinde uygulanmasını zorlaştırmaktadır [9].

Bu zorlukları aşmak için orijinal SPIHT algoritmasında J. Ritter, G. Fey ve P. Molitor tarafından iki önemli değişiklik yapılarak iyileştirilmiş listesiz SPIHT (MSPIHT) algoritması geliştirilmiştir [10]. İlk olarak dalgacık katsayılarının önemlilik testi bilgilerinin saklanması için dinamik listeler yerine iki boyutlu statik bellek haritaları kullanılmıştır. Bu sayede kodlama ve kod çözme işlemleri sırasında gerekli olacak bellek boyutu görüntü büyüklüğüne bağlı olarak işlem başlangıcında statik olarak belirlenebilmektedir.

Orijinal SPIHT algoritmasında iyileştirme aşaması sıralama aşamasından sonra yapılır. Bu işlem için sıralama aşamasından sonra önemli dalgacık katsayılarının bulunduğu listedeki (LSP) elemanlara bir önceki eşik değeriyle (n + 1) önemlilik testi uygulanır.

MSPIHT algoritmasında liste yerine harita kullanıldığından, hangi dalgacık katsayılarının önemli olduğunu bulabilmek için tüm haritayı taramak ve önemli olarak işaretlenmiş dalgacık katsayılarına bir önceki eşik değeri ile önemlilik testi yapmak gerekmektedir. Bu problemi çözmek için ikinci önemli değişiklik olarak MSPIHT algoritmasında iyileştirme aşamasının yeri değiştirilerek sıralama aşamasının öncesine

alınmıştır. Sıralama aşaması iyileştirme aşamasından sonra çalıştığı için LSP haritasında her zaman bir önceki eşik değeriyle önemlilik testini geçen dalgacık katsayıları bulunmaktadır. Bu sayede önemlilik testine gerek duyulmadan iyileştirme işlemi yapılabilmektedir.

Yukarıda anlatılan değişikliklerin gerçeklendiği MSPIHT algoritmasının sahte kodu aşağıda verilmiştir.

1. İlklendirme

01: $n := \lfloor log_2(\max|C_{ij}|) \rfloor$

02: $\boldsymbol{Her}\ 0 \leq i,j \leq N-1\ \boldsymbol{için}$

03: $LSP(i,j) = 0$

04: $LIP(i,j) = \begin{cases} 1, \boldsymbol{Eğer}\ (i,j) \in H\ \boldsymbol{ise} \\ 0, \boldsymbol{Değil\ ise} \end{cases}$

05: $\boldsymbol{Her}\ 0 \leq i,j \leq \frac{N}{2} - 1\ \boldsymbol{için}$

06: $LIS(i,j) = \begin{cases} A, \boldsymbol{Eğer}\ (i,j) \in H\ \boldsymbol{ve}\ O(i,j) \neq \emptyset\ \boldsymbol{ise} \\ 0, \boldsymbol{Değil\ ise} \end{cases}$

2. İyileştirme ve LIP için Sıralama

07: $\boldsymbol{Her}\ 0 \leq i,j \leq N-1\ \boldsymbol{için}$

08: $\boldsymbol{Eğer}\ LSP(i,j) = 1\ \boldsymbol{ise}$

09: $C_{ij}\ nin\ n.\ bitini\ çıkış\ dizisine\ gönder$

10: $\boldsymbol{Eğer}\ LIP(i,j) = 1\ \boldsymbol{ise}$

11: $S_n(i,j)\ değerini\ çıkış\ dizisine\ gönder$

12: $\boldsymbol{Eğer}\ S_n(i,j) = 1\ \boldsymbol{ise}$

13: $LSP(i,j) = 1$

14: $LIP(i,j) = 0$

15: $C_{ij}\ nin\ işaret\ bitini\ çıkış\ dizisine\ gönder$

3. LIS için Sıralama

16: $\boldsymbol{Her}\ 0 \leq i,j \leq N/2 - 1\ \boldsymbol{için}$

17: $\boldsymbol{Eğer}\ LIS(i,j) = A\ \boldsymbol{ise}$

18: $S_n\big(D(i,j)\big)\ değerini\ çıkış\ dizisine\ gönder$

19: $\boldsymbol{Eğer}\ S_n(D(i,j)) = 1\ \boldsymbol{ise}$

20: $\boldsymbol{Her}\ (k,l) \in O(i,j)\ \boldsymbol{için}$

21: $S_n(k,l)$ *değerini çıkış dizisine gönder*

22: **Eğer** $S_n(k,l) = 1$ **ise**

23: $LSP(k,l) = 1$

24: C_{kl} *nin işaret bitini çıkış dizisine gönder*

25: **Değil ise**

26: $LIP(k,l) = 1$

27: **Eğer** $L(i,j) \neq \emptyset$ **ise**

28: $LIS(i,j) = B$

29: **Değil ise**

30: $LIS(i,j) = 0$

31: **Eğer** $LIS(i,j) = B$ **ise**

32: $S_n\big(L(i,j)\big)$ *değerini çıkış dizisine gönder*

33: **Eğer** $S_n(L(i,j)) = 1$ **ise**

34: **Her** $(k,l) \in O(i,j)$ **için**

35: $LIS(k,l) = A$

36: $LIS(i,j) = 0$

4. Seviye Güncelleme

37: $n = 0$ **olana kadar** $n = n - 1$ *yap ve adım* $7'ye$ **git**

Şekil 2.20 - MSPIHT Algoritması

BÖLÜM 3. KABLOSUZ MULTİMEDYA ALGILAYICI AĞ KURULUMU VE GÖRÜNTÜ İLETİMİ

Bu bölümde kablosuz multimedya algılayıcı ağlarda görüntü sıkıştırma ve iletim için kullanacağımız donanım ve yazılım yapısı incelenecektir.

3.1 Donanım

Görüntü iletiminde kullanılan donanım yapısı üç ana bileşenden oluşmaktadır. Bu bileşenleri aşağıdaki gibi sıralayabiliriz.

1) Görüntü Kaynağı
2) Görüntü İşleyici Birim
3) RF Haberleşme Birimi

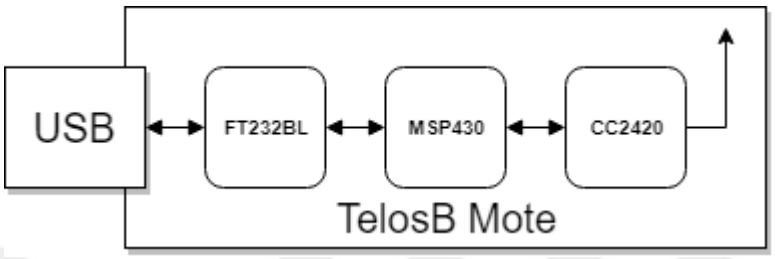
Görüntü kaynağı olarak USB WebCam kullanılmıştır. Kameranın çözünürlüğü 640x480 pikseldir. Kameradan görüntü almak için *OpenCV* kütüphanesi kullanılmıştır. Dalgacık dönüşümü uygulanabilmesi için kameradan alınan görüntü bir üst NxN çözünürlük olan 1024x1024 piksele dönüştürülmektedir.

Görüntü işleyici birim olarak RaspberryPi kullanılmıştır. RaspberryPi 4 adet USB porta sahiptir. Bu portlardan bir tanesine USB WebCam, bir diğerine de RF Haberleşme modülü bağlanmaktadır. 2 adet daha USB port kullanılabilir durumdadır. RaspberryPi teknik özellikleri Tablo 3.1'de görülmektedir.

Tablo 3.1 - RaspberryPi Teknik Özellikleri

SoC	BCM2837
CPU	Quad Cortex A53 @ 1.2GHz
Komut Seti	ARMv8-A
GPU	400MHz VideoCore IV
RAM	1GB SDRAM
Hafıza	micro-SD (4GB)
USB	4 USB2.0
Ethernet	10/100 Mbit
Kablosuz Ağ	802.11n / Bluetooth 4.0
Görüntü Çıkış	HDMI / Kompozit
Ses Çıkış	HDMI / Kulaklık
Güç	5V / 400mA (Ort.)

RF Haberleşme modülü olarak TelosB Mote kullanılmıştır. Mote üzerinde USB-UART çevirici, MSP430 MCU ve IEEE 802.15.4 uyumlu CC2420 RF Alıcı-Verici bulunmaktadır. Mote USB üzerinden bağlandığında enerjisini USB üzerinden alarak çalışmakta, USB üzerinden bağlanmaz ise 2xAA pil takılarak çalışması sağlanabilmektedir. TelosB Mote yapısı Şekil 3.1'de görülmektedir.



Şekil 3.1 - TelosB Mote Blok Diyagram

USB-UART çevirici sayesinde Mote ile seri port üzerinden veri alış verişi yapılabilmektedir. Seri porttan gönderilen veri paketleri öncelikle MCU tarafından alınmakta ve sonrasında RF üzerinden gönderilmek üzere alıcı-verici yongasına gönderilmektedir. Mote PCB üzerinde dahili anten bulunmakta ve harici antene ihtiyaç duymamaktadır. TelosB Mote üzerinde çalışan ve paket alış verişini sağlayan yazılım *Contiki-OS* kullanılarak yazılmıştır. TelosB Mote teknik özellikleri Tablo 3.2'de görülmektedir.

Tablo 3.2 - TelosB Mote Teknik Özellikleri

MCU	16-bit RISC (MSP430F1611) @ 3.9MHz
Program Flash	48 KB
RAM	10 KB
RF Alıcı-Verici	CC2420 (IEEE 802.15.4 Uyumlu)
Frekans Bandı	2400MHz - 2483.5MHz (ISM Band)
Veri Hızı	250 Kbps
RF Çıkış Gücü	-24 dBm - 0 dBm
RF Alıcı Duyarlılığı	-94 dBm
Güç	USB 5V veya 2xAA Pil
Arabirim	USB 2.0 FT232BL USB-UART

3.2 Yazılım

Görüntü iletimi için hazırlanan yazılım temel olarak görüntünün kameradan alınması, kodlanması ve RF haberleşme birimi üzerinden gönderilmesini sağlamaktadır. Yazılım C++ ile yazılmış olup, Raspberry Pi'a yüklenen bir Linux dağıtımı olan Raspbian üzerinde çalışmaktadır.

Kodlayıcı çıktısında elde edilen bit dizisi, kodlanan görüntünün geri elde edilmesi için tek başına yeterli olmamaktadır. Kodlanan görüntünün boyutu ve başlangıç eşik değeri gibi bilgilere de gerek duyulmaktadır. Bu sebeple bit dizisinin başına ek olarak bu bilgiler eklenmiştir. Kodlanan görüntü bilgilerini de içeren veri paket yapısı Şekil 3.2'de görülmektedir. Orijinal genişlik ve yükseklik görüntünün kaynağından elde edilen halindeki boyutlarıdır. Genişlik ve yükseklik ise görüntünün dalgacık dönüşümü uygulayabilmek için uygun hale getirilmiş boyutlarıdır.

16 bit	16 bit	16 bit	16 bit	8 bit	N bit
Genişlik	Yükseklik	Orj. Genişlik	Orj. Yükseklik	Eşik Değeri	Kodlayıcı Çıktısı

Şekil 3.2 - Kodlanmış Görüntü Veri Paket Yapısı

Kodlanmış görüntü bilgilerini içeren veri paket başlığı her zaman sabit 9 byte uzunluğundadır. Kodlayıcı çıktısının boyutu (N) kodlama işleminin başında istenilen uzunluk kadar belirlenebilmektedir.

RF haberleşme biriminin bir seferde gönderebileceği veri uzunluğu sınırlı olduğundan, kodlayıcı çıktısında oluşan veri paketinin daha küçük paketlere bölünerek gönderilmesi gerekmektedir. Ayrıca birden fazla vericinin olduğu algılayıcı ağlarda alıcıya ulaşan paketin hangi vericiye ait olduğunu da belirleyebilmemiz gerekmektedir. RF haberleşme birimi üzerinden bu iletimi sağlayabilmek için oluşturulan paket yapısı Şekil 3.3'de görülmektedir.

3 byte	1 byte	2 byte	64 byte
"SPT"	Cihaz No	Paket No	Paket

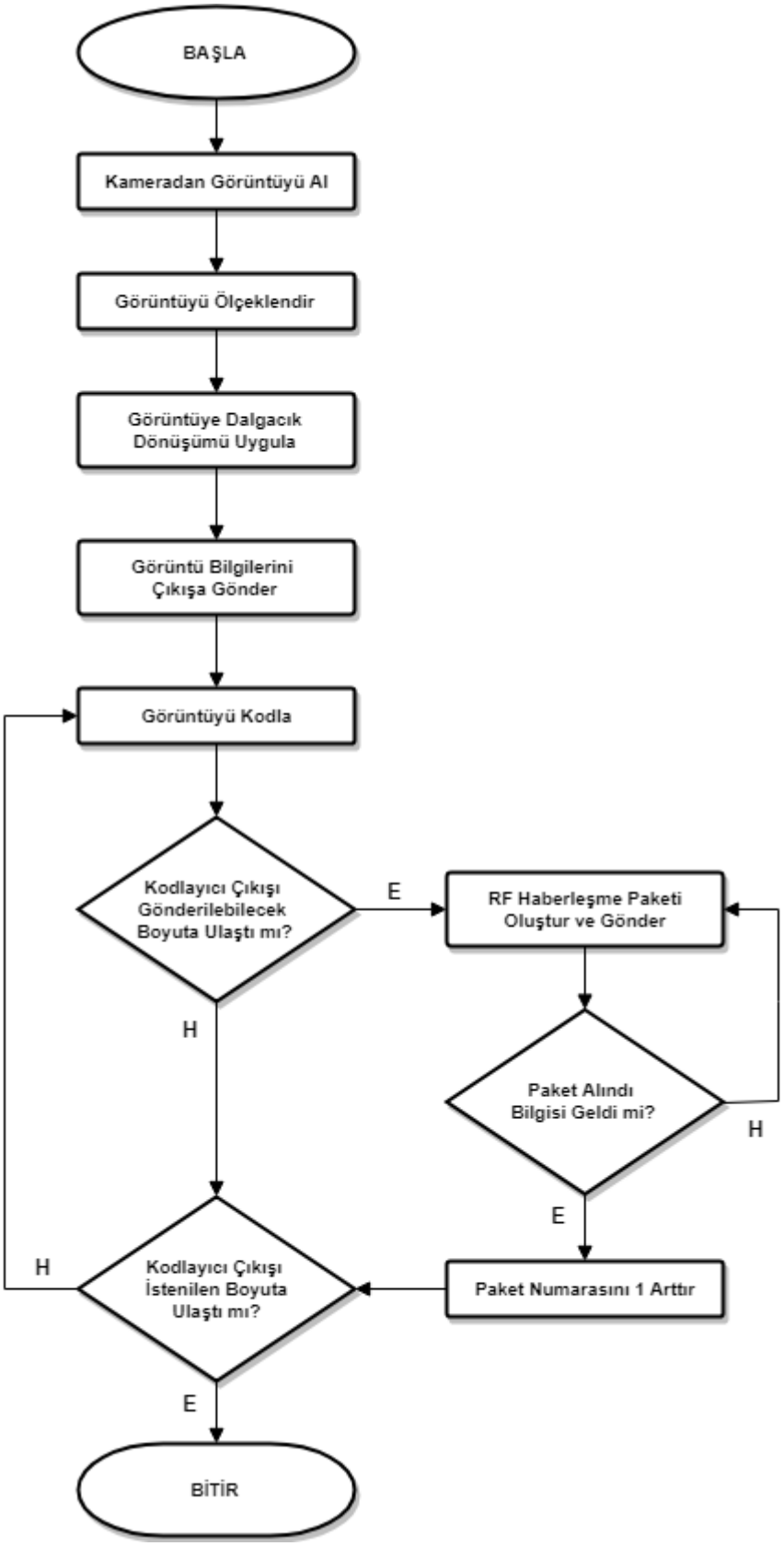
Şekil 3.3 - RF Haberleşme Birimi Paket Yapısı

“SPT” vericiden gönderilen paketin alıcıya ulaştığında paketin bütünlüğünü kontrol etmek için kullanılan 3 byte uzunluğunda bir sabittir. “Cihaz No” ağda bulunan her bir verici cihaza tekil olarak verilen numaradır. “Paket No” kodlanmış görüntü verisi paketlere bölünürken her pakette bir arttırılarak elde edilen sıra numarasıdır. Sıfır numaralı paket görüntü bilgilerini de içeren ilk paket olup, kodlayıcı çıktısının başlangıcını ifade eder.

Paketlere bölme ve iletme işlemini kodlama işlemi bittiğinde yapmak yerine, kodlama işlemi devam ederken RF haberleşme biriminin gönderebileceği paket boyutuna her ulaştığında yaparsak, alıcı tarafta kodlama ile eş zamanlı olarak görüntünün yaklaşık içeriğinden daha detay içeriğine doğru geri dönüşümünü elde etmiş oluruz. Bu sayede kodlama işlemi yarım kalsa bile alıcı tarafta görüntünün o ana kadar ki kodlanmış içeriği geri dönüştürülebilmektedir.

SPIHT algoritmasının ürettiği gömülü bit akışı sayesinde alıcı tarafa paketler ulaştıkça, ulaşan paket sayısı kadar görüntü geri dönüşümü yapılabilmektedir. Kodlama ve kod çözme işlemleri aynı sıra ile yapıldığından, iletimde bu sıra bilgisi kaybolur ise görüntü geri dönüşümünün devamı mümkün olmamaktadır. Bu sebeple vericiden gönderilen paketlerin aynı sıra ile alıcıya ulaştığından emin olunmalıdır. Bunu sağlayabilmek için alıcı tarafta paketler “Paket No” ile takip edilmekte ve çift taraflı el sıkışma (handshake) yöntemi ile vericiden gönderilen bir paketin alıcı tarafından alındı bilgisi gelmeden yeni paket gönderilmemektedir. Vericiden gönderilen bir paket için 5 saniye içerisinde alıcı tarafından alındı bilgisi gelmez ise, paket sırasının bozulmaması için verici aynı paketi tekrar göndermektedir.

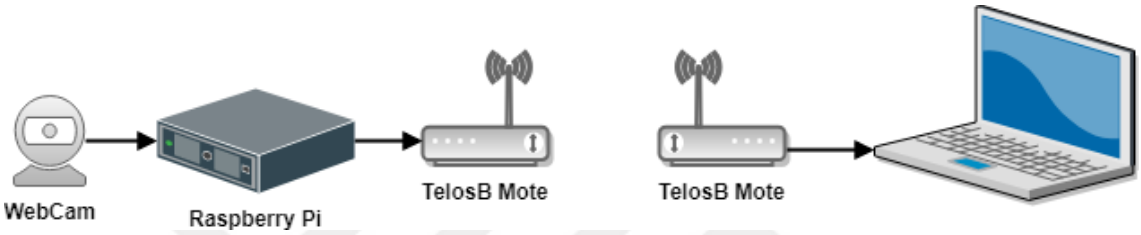
Yukarıda anlatılan yapıya uygun görüntü iletimi için gerçeklenen görüntü iletim algoritması Şekil 3.4’de verilmiştir.



Şekil 3.4 - Görüntü İletimi Algoritması

BÖLÜM 4. DENEYSEL BULGULAR

SPIHT ve MSPIHT algoritmalarının karşılaştırılması için 256x256, 512x512 ve 1024x1024 piksel görüntüler kullanılmıştır. Her görüntü 0.1 bpp, 0.5 bpp, 1.0 bpp ve 2.0 bpp parametreleri ile SPIHT ve MSPIHT algoritmaları kullanılarak ayrı ayrı kodlanmış, kodlama süresi, kodlamada kullanılan bellek miktarı ve geri dönüştürülen görüntünün PSNR değeri ölçülmüştür. WebCam'den görüntü iletimi için ise Şekil 4.1'de görülen düzenek kurulmuştur.



Şekil 4.1 - Görüntü İletim Düzeneği

Öncelikle farklı dalgacık ailelerinin görüntü kodlama ve geri dönüşümü üzerindeki etkisini incelemek için, Şekil 4.6'da verilen 512x512 piksel test görüntüsüne Haar, CDF 5/3 ve CDF 9/7 dalgacık fonksiyonları ayrı ayrı uygulanmıştır. Görüntü kodlama ve çözmede 1.0 bpp değeri ile MSPIHT algoritması kullanılmıştır. Geri dönüştürülen görüntülerin PSNR değerleri ve dalgacık dönüşümünde harcanan süre karşılaştırılmıştır.

Tablo 4.1 - Dalgacık Ailelerinin Dönüşüm Süresi ve PSNR Değerleri Sonuçları

	Süre (mS)	PSNR
CDF 9/7	85	26.0
CDF 5/3	71	28.9
HAAR	69	29.5

Tablo 4.1'deki sonuçlardan görüleceği üzere, görüntü kodlamada Haar dalgacık ailesi kullanılırsa daha yüksek PSNR değeri ve daha hızlı dönüşüm elde edilmektedir. Bu sebeple bundan sonraki deneylerde Haar dalgacık ailesi kullanılacaktır.

Algoritmaların karşılaştırılmasına 256x256 piksel görüntü ile başlanılmıştır. Bu karşılaştırma için Şekil 4.2'deki görüntü kullanılmıştır.

Şekil 4.2 - 256x256 Piksel Orijinal Test Görüntüsü

Şekil 4.2'deki test görüntüsü 0.1 bpp, 0.5 bpp, 1.0 bpp ve 2.0 bpp değerleri kullanılarak her iki algoritma ile kodlanmış ve geri dönüştürülmüştür. Elde edilen sonuçlar Tablo 4.2'de verilmiştir.

Tablo 4.2 - 256x256 Piksel Görüntü Kodlama Sonuçları

	SPIHT			MSPIHT		
	Süre (mS)	**Bellek (byte)**	**PSNR**	**Süre (mS)**	**Bellek (byte)**	**PSNR**
0.1 bpp	43	13909	20.8	40	147456	21.1
0.5 bpp	87	58530	26.3	50	147456	26.8
1.0 bpp	154	103354	29.5	56	147456	30.3
2.0 bpp	375	162069	33.2	64	147456	33.5

Tablo 4.2'deki sonuçlardan görüleceği üzere geri dönüştürülen görüntülerin PSNR değerleri birbirine yakındır. SPIHT algoritmasının artan bir şekilde kodlama süresi ve bellek kullanımına karşı, MSPIHT algoritmasının birbirine yakın kodlama süresi ve sabit bellek boyutu ile çalıştığı sonucu ortaya çıkmaktadır. Her iki algoritma tarafından geri dönüştürülen görüntüler Şekil 4.3, Şekil 4.4, Şekil 4.5 ve Şekil 4.6'da görülmektedir.

a) SPIHT

b) MSPIHT

Şekil 4.3 - 256x256 Piksel 0.1 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.4 - 256x256 Piksel 0.5 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.5 - 256x256 Piksel 1.0 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.6 - 256x256 Piksel 2.0 bpp Görüntü Geri Dönüşümü

İkinci karşılaştırma olan 512x512 piksel görüntü karşılaştırması için Şekil 4.7'de görülen test görüntüsü kullanılmıştır.

Şekil 4.7 - 512x512 Piksel Orijinal Test Görüntüsü

Bu görüntü de 0.1 bpp, 0.5 bpp, 1.0 bpp ve 2.0 bpp değerleri kullanılarak her iki algoritma tarafından kodlanmış ve geri dönüştürülmüştür. Elde edilen sonuçlar Tablo 4.3'de verilmiştir.

Tablo 4.3 - 512x512 Piksel Görüntü Kodlama Sonuçları

	SPIHT			MSPIHT		
	Süre (mS)	**Bellek (byte)**	**PSNR**	**Süre (mS)**	**Bellek (byte)**	**PSNR**
0.1 bpp	249	54569	21.8	270	589824	21.9
0.5 bpp	814	243613	25.9	325	589824	26.2
1.0 bpp	1967	405998	28.9	371	589824	29.5
2.0 bpp	5045	639414	32.9	409	589824	32.9

Tablo 4.3'de de 256x256 piksel görüntü kodlama sonuçlarına benzer sonuçlar elde edilmiştir. PSNR değerleri birbirine yakın olmakla birlikte, kodlama süresi ve bellek kullanımı SPIHT algoritmasında giderek artmakta olup, MSPIHT algoritmasında sabit bellek kullanımı ve kodlama sürelerinde daha küçük değişimler olduğu ortaya çıkmıştır. Her iki algoritma tarafından geri dönüştürülen görüntüler Şekil 4.8, Şekil 4.9, Şekil 4.10 ve Şekil 4.11'de görülmektedir.

a) SPIHT

b) MSPIHT

Şekil 4.8 - 512x512 Piksel 0.1 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.9 - 512x512 Piksel 0.5 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.10 - 512x512 Piksel 1.0 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.11 - 512x512 Piksel 2.0 bpp Görüntü Geri Dönüşümü

Üçüncü ve son karşılaştırma olan 1024x1024 piksel görüntü karşılaştırması için Şekil 4.12'de görülen test görüntüsü kullanılmıştır.

Şekil 4.12 - 1024x1024 Piksel Orijinal Test Görüntüsü

Bu görüntü de 0.1 bpp, 0.5 bpp, 1.0 bpp ve 2.0 bpp değerleri kullanılarak her iki algoritma tarafından kodlanmış ve geri dönüştürülmüştür. Elde edilen sonuçlar Tablo 4.4'de verilmiştir.

Tablo 4.4 - 1024x1024 Piksel Görüntü Kodlama Sonuçları

	SPIHT			MSPIHT		
	Süre (mS)	**Bellek (byte)**	**PSNR**	**Süre (mS)**	**Bellek (byte)**	**PSNR**
0.1 bpp	1348	184283	27.6	2390	2359296	27.6
0.5 bpp	8636	853030	31.5	2676	2359296	31.5
1.0 bpp	26166	1889420	32.4	2882	2359296	32.7
2.0 bpp	94423	3390856	33.4	3101	2359296	33.7

Tablo 4.4'deki sonuçlara göre PSNR değeri her iki algoritma için de yaklaşık aynı olduğu görülmüştür. MSPIHT algoritmasında kodlama süreleri arasında küçük farklar olduğu ancak bellek kullanımının değişmediği, SPIHT algoritmasında ise bellek kullanımının ve özellikle kodlama süresinin oldukça arttığı sonuçları ortaya çıkmıştır. Her iki algoritma tarafından geri dönüştürülen görüntüler Şekil 4.13, Şekil 4.14, Şekil 4.15 ve Şekil 4.16'da görülmektedir.

a) SPIHT

b) MSPIHT

Şekil 4.13 - 1024x1024 Piksel 0.1 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

Şekil 4.14 - 1024x1024 Piksel 0.5 bpp Görüntü Geri Dönüşümü

a) SPIHT

b) MSPIHT

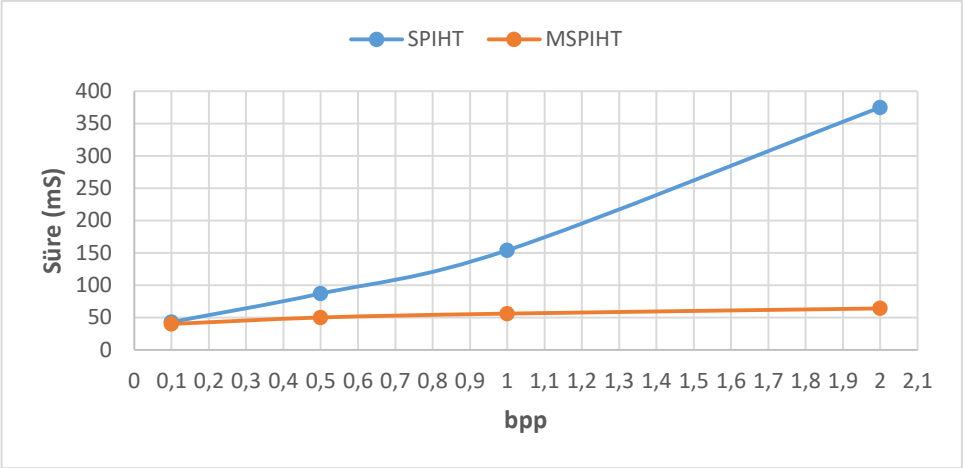
Şekil 4.15 - 1024x1024 Piksel 1.0 bpp Görüntü Geri Dönüşümü

a) SPIHT

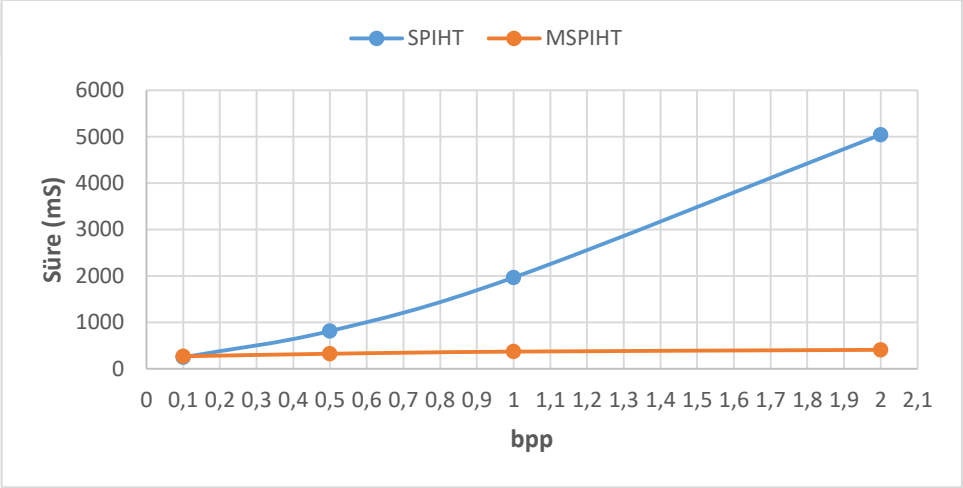
b) MSPIHT

Şekil 4.16 - 1024x1024 Piksel 2.0 bpp Görüntü Geri Dönüşümü

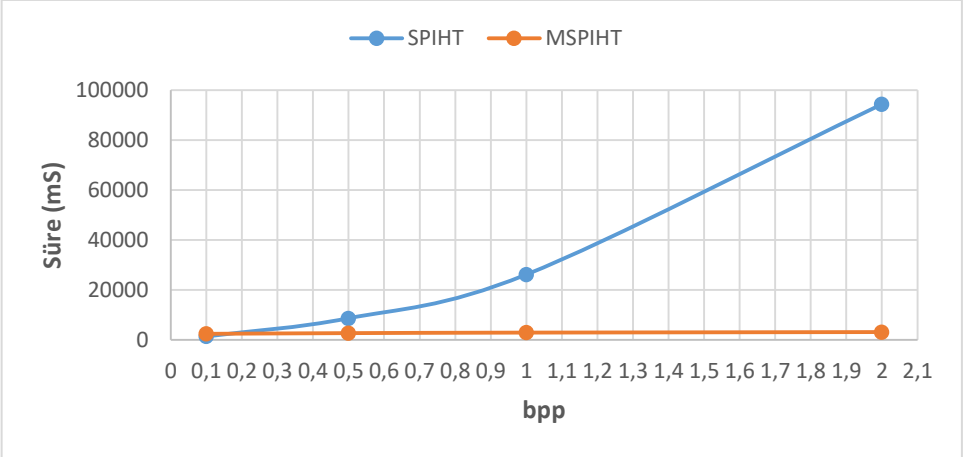
Yukarıda elde edilen sonuçların grafik olarak gösterimi Şekil 4.17, Şekil 4.18 ve Şekil 4.19'da verilmiştir.



a) 256x256 Piksel Görüntü Kodlama bpp-Süre Grafiği

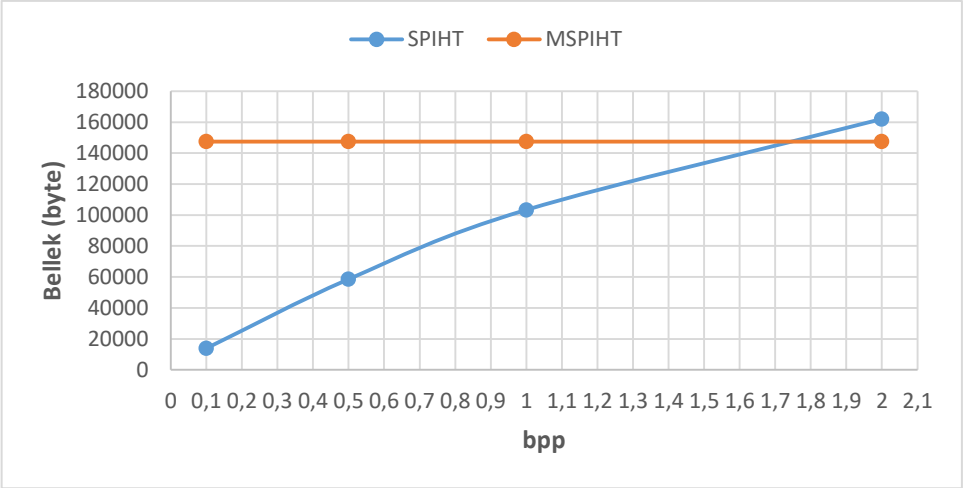


b) 512x512 Piksel Görüntü Kodlama bpp-Süre Grafiği

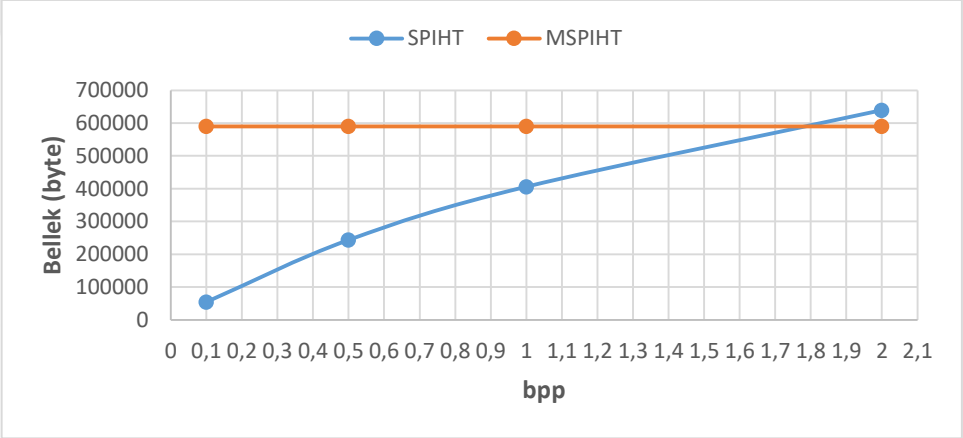


c) 1024x1024 Piksel Görüntü Kodlama bpp-Süre Grafiği

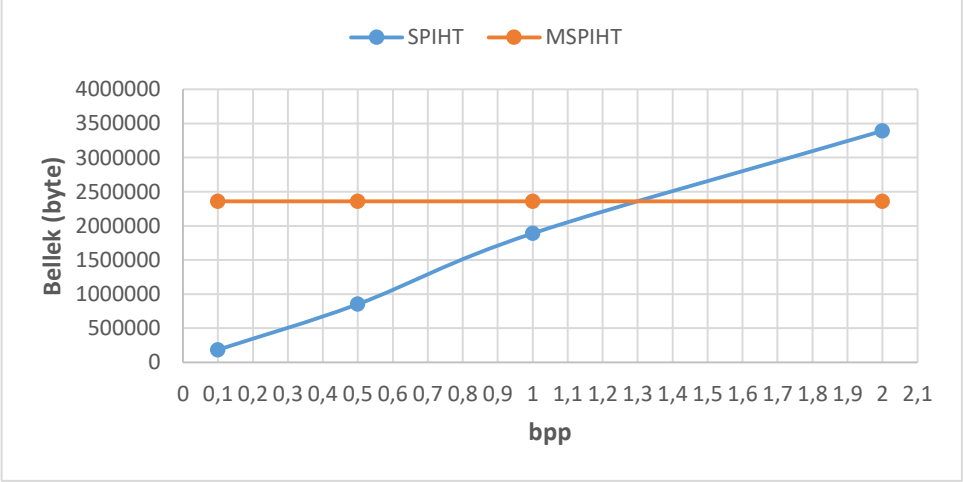
Şekil 4.17 - Farklı Çözünürlükteki Görüntülerin Kodlama İşlem Süresi Grafikleri



a) 256x256 Piksel Görüntü Kodlama bpp-Bellek Grafiği

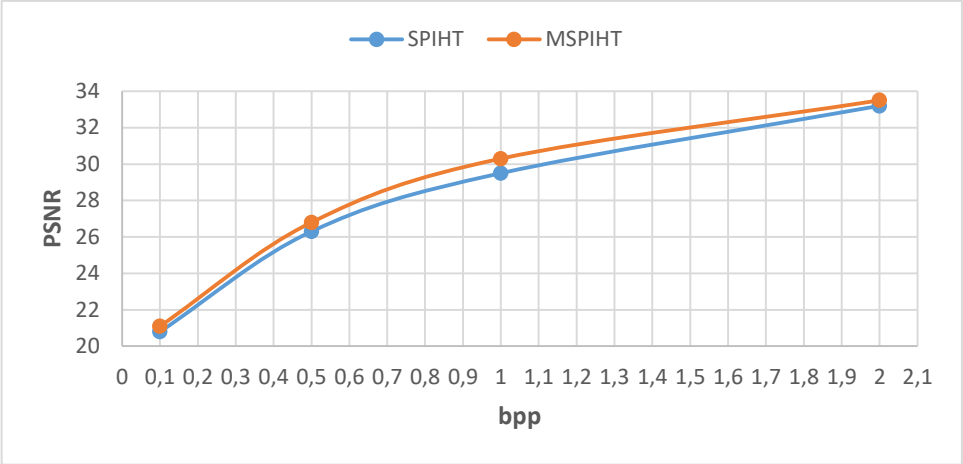


b) 512x512 Piksel Görüntü Kodlama bpp-Bellek Grafiği

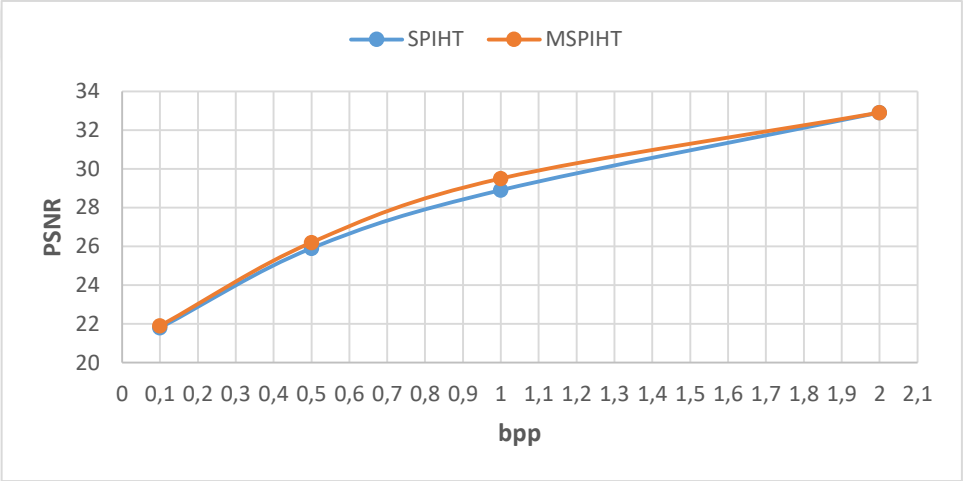


c) 1024x1024 Piksel Görüntü Kodlama bpp-Bellek Grafiği

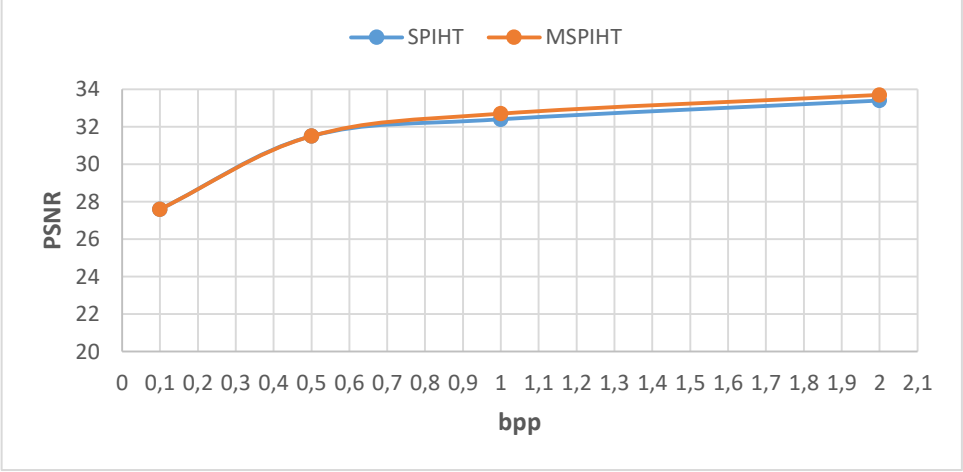
Şekil 4.18 - Farklı Çözünürlükteki Görüntülerin Kodlama Bellek Kullanım Grafikleri



a) 256x256 Piksel Görüntü bpp-PSNR Grafiği



b) 512x512 Piksel Görüntü bpp-PSNR Grafiği



c) 1024x1024 Piksel Görüntü bpp-PSNR Grafiği

Şekil 4.19 - Farklı Çözünürlükteki Görüntülerin PSNR Grafikleri

BÖLÜM 5. SONUÇ

Bu tez çalışmasında kablosuz algılayıcı ağlarda görüntü iletimi için SPIHT algoritması incelenmiş, değiştirilmiş algoritma ve verimli bir görüntü iletim yöntemi ile donanım üzerinde uygulanmıştır. Orijinal SPIHT ve değiştirilmiş MSPIHT algoritmaları ile çeşitli boyutlardaki görüntüler sıkıştırılmış, işlem süresi, bellek kullanımı ve PSNR değerleri karşılaştırılmıştır. PSNR değerlerinin her iki algoritma için birbirlerine çok yakın olduğu görülmüştür. İşlem süresi MSPIHT algoritmasında çıkış görüntü boyutuna göre çok fazla değişmemekle birlikte, SPIHT algoritmasında çıkış görüntü boyutu arttıkça işlem süresinin arttığı görülmüştür. Bellek kullanımı olarak MSPIHT algoritması işlem başlangıcında görüntü boyutuna göre belirlenen sabit bir bellek miktarı gerektirdiği görülmüştür. Bu bellek miktarı küçük boyutlu görüntü çıktıları için SPIHT'e göre yüksek olsa da, çıkış görüntü boyutu arttıkça SPIHT algoritmasının bellek ihtiyacının artan bir şekilde yükseldiği görülmüştür.

EK'LER

1. Haar, CDF 5/3, CDF 9/7 Dalgacık Dönüşümü Fonksiyonları

```
01: void fwt_HAAR(int *x, int n)
02: {
03:    int i;
04:
05:    // Predict 1
06:    double alpha = -1.0;
07:    for (i = 1; i < n; i += 2)
08:           x[i] += alpha * x[i - 1];
09:
10:    // Update 1
11:    double beta = 0.5;
12:    for (i = 0; i < n; i += 2)
13:           x[i] += beta * x[i + 1];
14:
15:    // Scale
16:    double s = 1 / sqrt(2.0);
17:    for (i = 0; i < n; i++) {
18:           if (i % 2) x[i] *= s;
19:           else x[i] /= s;
20:    }
21:
22:    // Pack
23:    int *tempbank = new int[n];
24:    for (i = 0; i < n; i++) {
25:           if (i % 2 == 0) tempbank[i / 2] = x[i];
26:           else tempbank[n / 2 + i / 2] = x[i];
27:    }
28:    for (i = 0; i < n; i++) x[i] = tempbank[i];
29:    delete[] tempbank;
30: }
```

```
01: void fwt_cdf53(int *x, int n)
02: {
03:    int i;
04:
05:    // Predict 1
06:    double alpha = -0.5;
07:    for (i = 1; i < n - 2; i += 2)
08:           x[i] += alpha * (x[i - 1] + x[i + 1]);
09:    x[n - 1] += 2 * alpha * x[n - 2];
10:
11:    // Update 1
12:    double beta = 0.25;
13:    for (i = 2; i < n; i += 2)
14:           x[i] += beta * (x[i - 1] + x[i + 1]);
15:    x[0] += 2 * beta * x[1];
16:
17:    // Scale
18:    double s = 1 / sqrt(2.0);
```

```
19:     for (i = 0; i < n; i++) {
20:           if (i % 2) x[i] *= s;
21:           else x[i] /= s;
22:     }
23:
24:     // Pack
25:     int *tempbank = new int[n];
26:     for (i = 0; i < n; i++) {
27:           if (i % 2 == 0) tempbank[i / 2] = x[i];
28:           else tempbank[n / 2 + i / 2] = x[i];
29:     }
30:     for (i = 0; i < n; i++) x[i] = tempbank[i];
31:     delete[] tempbank;
32: }
```

```
01: void fwt_cdf97(int *x, int n)
02: {
03:     int i;
04:
05:     // Predict 1
06:     double alpha = -1.586134342;
07:     for (i = 1; i < n - 2; i += 2)
08:           x[i] += alpha * (x[i - 1] + x[i + 1]);
09:     x[n - 1] += 2 * alpha * x[n - 2];
10:
11:     // Update 1
12:     double beta = -0.05298011854;
13:     for (i = 2; i < n; i += 2)
14:           x[i] += beta * (x[i - 1] + x[i + 1]);
15:     x[0] += 2 * beta * x[1];
16:
17:     // Predict 2
18:     double gamma = 0.8829110762;
19:     for (i = 1; i < n - 2; i += 2)
20:           x[i] += gamma * (x[i - 1] + x[i + 1]);
21:     x[n - 1] += 2 * gamma * x[n - 2];
22:
23:     // Update 2
24:     double delta = 0.4435068522;
25:     for (i = 2; i < n; i += 2)
26:           x[i] += delta * (x[i - 1] + x[i + 1]);
27:     x[0] += 2 * delta * x[1];
28:
29:     // Scale
30:     double s = 1 / 1.149604398;
31:     for (i = 0; i < n; i++) {
32:           if (i % 2) x[i] *= s;
33:           else x[i] /= s;
34:     }
35:
36:     // Pack
37:     int *tempbank = new int[n];
38:     for (i = 0; i < n; i++) {
39:           if (i % 2 == 0) tempbank[i / 2] = x[i];
40:           else tempbank[n / 2 + i / 2] = x[i];
41:     }
42:     for (i = 0; i < n; i++) x[i] = tempbank[i];
43:     delete[] tempbank;
44: }
```

2. Deney Düzeneği

KAYNAKÇA

[1] M. A. E. H. A. A. Hanaa ZainEldin, «A modified listless strip based SPIHT for wireless multimedia sensor networks,» *Computers and Electrical Engineering archive,* cilt 56, no. C, pp. 519-532, 2016.

[2] M. S. D. C. Kaimal A.B, «Image Compression Technique: a Survey,» *International Journal of Engineering Inventions,* cilt 2, no. 4, 2013.

[3] G. K. Wallace, «The JPEG Still Picture Compression Standard,» *IEEE Transactions on Consumer Electronics,* cilt 38, no. 1, 1992.

[4] O. Ghorbel, «DCT & DWT Images Compression Algorithms in Wireless Sensors Networks: Comparative Study and Performance Analysis,» *International Journal of Wireless & Mobile Networks,* cilt 4, p. 45, 2012.

[5] J. Shapiro, «Embedded Image Coding Using Zerotrees of Wavelet Coefficients,» *IEEE Trans. Signal Proc.,* cilt 40, no. 12, pp. 3445-3462, 1993.

[6] B. ROQUE ve J. SALVADO, «A comparative study on JPEG-Like and EZW based image coders,» *WSEAS Transactions on Computers,* cilt 2, no. 3, pp. 521-526, 2003.

[7] A. Said ve W. Pearlman, «A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Tress,» *IEEE Trans. Circuits And Systems For Video Technology,* cilt 6, no. 3, pp. 243-250, 1996.

[8] K. Nagamani ve A. Ananth, «EZW and SPIHT Image Compression Techniques for High Resolution Satellite Imageries,» *International Journal of Advanced Engineering Technology Computer Application,* cilt 2, no. 2, pp. 82-86, 2011.

[9] F. W. Wheeler ve W. A. Pearlman, «SPIHT Image Compression without Lists,» *IEEE International Conference on Acoustics, Speech and Signal Processing,* cilt 4, pp. 2047-2050, 2000.

J. Ritter, G. Fey ve P. Molitor, «SPIHT implemented in a XC4000 device,» [10] *Proceedings of IEEE 45th Midwest Symposium on Circuits and Systems,* 2002.

A. Mesut, «Veri Sıkıştırmada Yeni Yöntemler,» *Bilgisayar Mühendisliği* [11] *Anabilim Dalı, Trakya Üniversitesi Y. Lisans Tezi,* 2006.

T. Kumar ve D. Chaudhary, «Compression Study Between ezw, spiht, stw, [12] wdr, aswdr and spiht_3d,» *International Journal of Scientific & Research,* cilt 4, pp. 710-716, 2013.

F. RAJABLI, «Darbantlı Video İletişim Sistemlerinde Dalgacık [13] Dönüşümünün Kullanılması,» *Ankara Üniversitesi Y. Lisans Tezi,* 2005.

R. A. Abbak, «Jeodezide Zaman Dizilerinin Dalgacık (Wavelet) Analizi,» [14] *Doktora Semineri, Selçuk Üniversitesi Fen Bilimleri Enstitüsü,* 2007.

M. Osadebey ve A. A Georgakis, «Spread spectrum wavelet watermarking [15] system,» 2006.

G. Uytterhoeven, D. Roose ve A. Bultheel, «Wavelet Transforms Using [16] the Lifting Scheme,» 1997.

P. Hardik Anilkumar ve P. Augusta Sophy Beulet, «Lifting-based Discrete [17] Wavelet Transform for Real-Time Signal Detection,» *Indian Journal of Science and Technology,* cilt 8, 2015.

P. J., «Wavelet transforms, Lifting,» *Computer Graphics Group, Charles* [18] *University,* 2016.

T. M. Quan ve W.-K. Jeong, «A Fast Discrete Wavelet Transform Using [19] Hybrid Parallelism on GPUs,» *IEEE Transactions on Parallel and Distributed Systems,* 2016.

T. Blaser, S. Senn ve P. Stadelmann, «Wavelet-based Compression using [20] the SPIHT Algorithm,» Integrated Systems Laboratory (IIS) Microelectronics Design Center ETH Zurich, 2006.