T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖMÜLÜ YAYA BELİRLEME SİSTEMİ TASARIMI VE GERÇEKLEMESİ

AHMET REMZİ ÖZCAN

YÜKSEK LİSANS TEZİ
ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ ANABİLİM DALI
ELEKTRONİK PROGRAMI

DANIŞMAN
PROF. DR. VEDAT TAVŞANOĞLU

İSTANBUL, 2013

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

GÖMÜLÜ YAYA BELİRLEME SİSTEMİ TASARIMI VE GERÇEKLEMESİ

Ahmet Remzi Özcan tarafından hazırlanan tez çalışması 13.08.2013 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Haberleşme Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Vedat TAVŞANOĞLU

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Prof. Dr. Vedat TAVŞANOĞLU

Yıldız Teknik Üniversitesi _____

Prof. Dr. Herman SEDEF

Yıldız Teknik Üniversitesi _____

Prof. Dr. Oruç BİLGİÇ

İstanbul Kültür Üniversitesi _____

ÖNSÖZ

Üniversite hayatım boyunca değerli fikirlerinden istifade ettiğim, tez danışmanım Prof. Dr. Vedat Tavşanoğlu hocama, tez çalışmalarım süresince bilgi ve deneyimlerini benimle paylaşan, motive eden ve anlayış gösteren Nerhun Yıldız ve Evren Cesur'a teşekkürlerimi bir borç bilirim.

Tez çalışmam süresince bana yol gösteren babama teşekkür eder, hayatım boyunca benden maddi manevi desteğini esirgemeyen, özellikle tez çalışmam süresince büyük fedakarlıkta bulunan sevgili aileme en derin şükranlarımı sunarım.

Temmuz, 2013

Ahmet Remzi Özcan

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

α_i	Pozitif Lagrange çarpanları
b	Hiper-düzlem eğilim değeri
B	RGB sinyali mavi renk bileşeni
f	Normalize edilmiş blok çıkış vektörü
$g_x(x,y)$	Gradyan yatay bileşeni
$g_y(x,y)$	Gradyan dikey bileşeni
G	RGB sinyali yeşil renk bileşeni
h(n)	n. hücre histogramı
Ht	Hog tanımlayıcı sayısı
k	Normalizasyon bloğu boyutu
L_p	Lagrange eşitliği
m(x,y)	Gradyan büyüklüğü
p	Gradyan yön bölgesi sayısı
R	RGB sinyali kırmızı renk bileşeni
θ(x,y)	Gradyan yönü
$\|v\|_n$	n. dereceden norm
v	Blok çıkış vektörü
w	Hiper-düzlem normal vektörü
x_i	Öznitelikler vektörü
ξ_i	Eğitim hatası sapması değeri
y_i	Sınıf etiketleri
Y	Lüminans değeri

KISALTMA LİSTESİ

ADC	Analog Digital Converter
AES	Advanced Encryption Standard
ARM	Acorn RISC Machine
BBA	Bağımsız Bileşen Analizi
CANBUS	Controller Area Network Bus
CoHED	Co-occurence Histograms of Pairs of Edge Orientations and Color Differences
CoHD	Co-occurrence Histograms of Color Differences
CoHOG	Co-occurence HOG
DAA	Doğrusal Ayırma Analizi
DDR2	Double Data Rate RAM
DoG	Difference-of-Gaussian
DoH	Determinant-of-Hessian
DHCP	Dynamic Host Configuration Protocol
EMIF	External Memory Interface
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GLOH	Gradient Location and Orientation Histogram
GPIO	General Purpose Input/Output
GPL	General Public License
GPMC	General Purpose Memory Controller
HOG	Histogram of Oriented Gradients
I2C	Inter-Integrated Circuit
INRIA	Institut national de recherche en informatique et en automatique
KB	KiloByte
KKT	Karush-Kuhn-Tucker Conditions
LCD	Liquid-Crystal Display
LED	Light-Emitting Diode
LVDS	Low-Voltage Differential Signaling
LoG	Laplacian-of-Gaussian
mA	MiliAmper
MB	MegaByte
MicroSD	Micro Secure Digital
MHz	MegaHertz
MIT	Massachusetts Institute of Technology

MMC	MultiMediaCard
MSER	Maksimum Durağan Uç Bölgeler
NAND	Negated AND
NICTA	National Information and Communications Technology Australia
NOR	Negated OR
PCA	Principal Components Analysis
PKA	Public Key Authentication
PLL	Phase-Locked Loop
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RGB	Red Green Blue color model
RNG	Random Number Generator
ROC	Receiver Operating Characteristic
RISC	Reduced Instruction Set Computing
RS232	Recommended Standard 232
SD	Secure Digital
SHA	Secure Hash Algorithm
SIFT	Scale-Invariant Feature Transform
SIMD	Single Instruction, Multiple Data
SPI	Serial Peripheral Interface
SRAM	Static RAM
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
TBA	Temel Bileşen Analizi
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

GÖMÜLÜ YAYA BELİRLEME SİSTEMİ TASARIMI VE GERÇEKLEMESİ

Ahmet Remzi ÖZCAN

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Prof. Dr. Vedat TAVŞANOĞLU

Artan görüntü işleme çalışmaları ve gelişen teknoloji, görüntü işleme temelli sistemlerin uygulama alanlarını hızla genişletmektedir. Günümüzde, bilgisayarla görü sistemleri ile yüz tanıma, plaka tanıma, yaya tanıma gibi nesne tanıma uygulamalarına sıkça rastlanılmaktadır. Son yıllarda yaya tanıma sistemleri, otomotiv teknolojileri alanında da kendine uygulama sahası bulmakta, trafikte yaşanan kazalar ve nedenleri üzerine hazırlanan raporlar, bu alanda yapılan çalışmalar için motivasyon kaynağı olmaktadır. Dünya Sağlık Örgütü tarafından hazırlanan raporlara göre, trafik kazaları dünya genelinde yılda 1.200.000 insanın ölümüne neden olurken, bu ölümlerin büyük bir çoğunluğunu yayalar oluşturmaktadır.

Bu çalışmada Yönlü Gradyanların Histogramı (HOG) algoritması ve Destek Vektör Makinesi (SVM) kullanılarak bir yaya tanıma uygulaması gerçekleştirilmiştir. Yaya tanıma uygulamalarında öznitelik çıkarıcı bir yöntem olarak sıklıkla kullanılan HOG algoritması Sahada Programlanabilir Kapı Dizileri (FPGA) ile donanımsal olarak gerçeklenmiş, çıkarılan özniteliklerin sınıflandırılması ve yaya tanıma işi ise mikroişlemci üzerinde, SVM ile yazılımsal olarak yapılmıştır.

Tez çalışması kapsamında, tasarlanan sistemi gerçeklemek üzere FPGA ve mikroişlemcili bir düzenek kurulmuş, düzeneğin birimleri olan FPGA ve mikroişlemciyi haberleştirmek üzere basit bir arayüz kartı tasarlanmıştır. Kurulan düzenekte yer alan FPGA ve mikroişlemci birimlerinin seçiminde düzeneğin taşınabilirliği ve güç tüketimi

ön planda tutulmuştur. Buna göre FPGA birimi olarak Terasic firmasının *Altera DE0 Nano* uygulama geliştirme kartı, mikroişlemci birimi olarak ise ARM mimarisi kullanan *BeagleBone* uygulama geliştirme kartı kullanılmıştır.

Gerçeklenen sistem, NICTA yaya veri tabanından alınan 2000 örnek ile eğitilmiş, aynı veritabanından alınan 2000 farklı örnek ile yapılan testlerde sınıflandırma başarımı %98.15 olarak ölçülmüştür.

Anahtar Kelimeler: Yönlü Gradyanların Histogramı, Destek Vektör Makinesi, Yaya Tanıma, Sahada Programlanabilir Kapı Dizileri, Gömülü Mikroişlemci

ABSTRACT

EMBEDDED DESIGN AND IMPLEMETATION OF PEDESTRIAN DETECTION SYSTEM

Ahmet Remzi ÖZCAN

Department of Electronics and Communications Engineering

MSc. Thesis

Advisor: Prof. Dr. Vedat TAVŞANOĞLU

Image processing based systems rapidly expand the application areas with developing image processing studies and technology. Object recognition applications such as face, licience plate, pedestrian recognition with computer vision systems is frequently encountered nowadays. In recent years, pedestrian recognition systems find its field of application on automotive technologies areas to prevent traffic accidents. Reports prepared on reasons of this accidents be motivation source for studies which is done in this area. According to the reports prepared by World Health Organisation (WHO), traffic accidents caused deaths of 1.200.000 person per year in the world and the vast majority of these deaths are pedestrians.

In this study, a pedestrian recognition application was developed by using Histogram of Oriented Gradients (HOG) algorithm and Support Vector Machine (SVM). HOG algorithm which is frequently used as a feature extraction method in pedestrian recognition applications was implemented in FPGA hardware then classification of extracted features and pedestrian recognition are done by SVM software by microprocessor.

In this thesis, a mechanism established including FPGA and microprocessor to implement designed system also a basic interface card designed for communicate with FPGA and microprocessor. Mobility and the power consumption are the first priority of selection that FPGA and microprocessor units mounted on assembled mechanism. *Altera DE0 Nano* application development board produced by Terasic company is used

as a FPGA unit and *BeagleBone* application development board which use ARM architecture is selected as a microprocessor unit.

The implemented system is trained with the 2000 train samples of NICTA pedestrian database and obtained 98.15% classification performance with 2000 test samples.

Keywords: Histogram of Oriented Gradients, Support Vector Machine, Pedestrian Recognition, Field Programable Gate Array, Embedded Processor

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Bir görüntü üzerinde yayaları tanıma üzerine yapılan çalışmalar 80'li yıllara kadar uzanmaktadır. Bu alanda yapılan ilk çalışmalardan biri Tsukiyama ve Shirai'ye aittir. 1985 yılında televizyon görüntüleri üzerinde yaya hareketlerinin tespit eden bir sistem geliştirmişlerdir [1]. 1997 yılında Oren vd. tarafından yapılan çalışmada durağan bir görüntü üzerinde yayaları tanımak üzere dalgacık şablonlarının kullanımı önerilmiştir [2]. Gavrila ve Philomin, 1999'da akıllı araçlar için gerçek zamanlı nesne tanımaya yönelik görünüş tabanlı bir yöntem geliştirmişlerdir [3]. Bu yöntemde birçok nesne görüntüsü için uzaklık dönüşümü kullanılarak bir şablon sıradüzeni oluşturulmakta, oluşturulan bu sıradüzen görüntü üzerinde gerçek zamanlı olarak eşleme amacıyla kullanılmaktadır. Elde edilen uyum derecesine göre ilgili bölgede aranan nesne olup olmadığına karar verilmektedir.

2000 yılında Papageorgiou ve Poggio nesne tanımaya yönelik genel amaçlı, eğitilebilir bir sistem geliştirmiştir [4]. İlgili sistemde görüntüden öznitelik çıkarma amacıyla Haar dalgacık dönüşümünden yararlanılmakta, çıkarılan öznitelikler SVM ile sınıflandırılarak sistem eğitilmektedir. Viola vd. 2003'de görüntüdeki piksel parlaklığı ve değişim bilgisini kullanarak yaya tanıyan bir yöntem önermiştir [5]. Bu yöntemde, oluşturulan Adaboost tabanlı algılayıcı yürümekte olan insanları, iki ardışık hareketli görüntü karesindeki hareket ve görünüş bilgilerini kullanarak saptayabilmektedir.

Dalal ve Triggs, 2005 yılında yayınladıkları "Histogram of Oriented Gradients for Human Detection" adlı makalede yaya tanımaya yönelik olarak HOG tanımlayıcılarını kullanan

bir yöntem önermişlerdir [6]. Önerilen yöntemde HOG algoritması öznitelik çıkarma amacıyla kullanılmış, çıkarılan öznitelikleri sınıflandırmada doğrusal SVM'den yararlanılmıştır. MIT [7] ve INRIA [8] yaya veritabanı üzerinde yapılan performans testlerinde yaya tanımada %85'lere varan bir başarım elde edilmiştir. 2006 yılında Dalal vd. HOG algoritması ile hareket tabanlı tanımlayıcıları birlikte kullanarak, hareketli görüntü üzerinde yaya tanıyan bir yöntem geliştirmiştir [9]. Sabzmeydani ve Mori, 2007'de durağan görüntü üzerinde yaya tanıma problemine yönelik olarak Shapelet özniteleyicilerini kullanan bir algoritma geliştirmiştir [10]. Shapelet özniteleyicileri, Adaboost zayıf sınıflandırıcısı kullanılarak, yönlü gradyan cevabının bir türevini olarak oluşturulmaktadır. Adaboost aynı zamanda çıkarılan özniteliklerin sınıflandırılması amacıyla da kullanılmıştır. Wu ve Nevatia, 2007 yılında yaptıkları çalışmada Edgelet tabanlı bir tanıma algoritması önermiştir [11]. Bu algoritma ile insan vücudu parçalara ayrılarak, vücudun farklı pozisyonlarda kolay tanınabilmesi amaçlanmıştır.

Görüntü üzerinde yaya tanıma için önerilen yöntemlerde karşılaşılan ortak problem işlenecek verini fazlalığı ve buna bağlı olarak yüksek hesaplama performansı ihtiyacıdır. Bu nedenle yüksek başarım sağlayan birçok algoritma ile klasik bilgisayarlarda gerçek zamanlı yaya tanıma uygulamaları oluşturmak mümkün olamamaktadır. Bauer vd. 2009 yılında yaptıkları çalışmada HOG tabanlı hibrid bir yaya tanıma sistemi oluşturmuştur [12]. Oluşturulan sistemde HOG algoritması FPGA'da, SVM grafik kartı üzerinde gerçeklenerek ihtiyaç duyulan yüksek işlem performansı karşılanmaktadır. Aynı yıl Kadota vd. yaptıkları çalışmada, yaya tanıma uygulamalarında sıklıkla kullanılan HOG algoritmasının donanımsal gerçeklemesinde karşılaşılan problemler için bazı yaklaşımlar önermiştir [13]. Yapılan çalışmada, FPGA üzerinde donanımsal olarak gerçeklenen HOG algoritması saniyede 30 karelik VGA görüntüyü gerçek zamanlı olarak işleyebilmektedir.2011 yılında Negi vd. HOG'un donanımsal gerçeklemesi için önerilen yaklaşımları geliştirerek tamamen FPGA üzerinde gerçek zamanlı bir yaya tanıma uygulaması oluşturmuştur [14]. Oluşturulan uygulamada FPGA üzerinde HOG algoritması dışında Adaboost sınıflandırıcısı da bulunmaktadır. Tasarlanan sistem saniyede 62,5 kare hızında, 320x240 çözünürlüğündeki görüntüyü gerçek zamanlı olarak işleyebilmektedir.

1.2 Tezin Amacı

Gelişen teknoloji sayesinde hayatın birçok alanında akıllı sistemlerin kullanımı yaygınlaşmaktadır. Bu alanlardan biri olan otomotiv alanında özellikle sürüş güvenliği konusunda bu akıllı sistemler kilit rol oynamaktadır. Trafikte karşılaşılan kaza durumları incelendiğinde hatanın daha çok insan faktöründen kaynaklanıyor olması, sürücüyü akıllı sistemler ile destekleme fikrini doğurmuştur. Trafik kazaları üzerine hazırlanan raporlar, ölümlü trafik kazalarında yaşanan ölümlerin dikkate değer bir kısmını yaya kayıplarının oluşturduğunu göstermektedir [15]. Buna bağlı olarak otomotiv teknolojisinde yaya tanıma sistemleri üzerine yapılan çalışmalar da ivmelenmiştir.

Bu tez çalışmasında, Yönlü Gradyanların Histogramı (Histogram of Oriented Gradients – HOG) algoritması ve Destek Vektör Makinesi (Support Vector Machine – SVM) kullanılarak bir yaya tanıma uygulaması gerçekleştirilmiştir. Gerçek zamanlı çalışan bir altyapı oluşturabilmek amacıyla, HOG algoritması FPGA ile donanımsal olarak gerçeklenmiş, sınıflandırma ve yaya tanıma için kullanılan SVM mikroişlemci üzerinde yazılımsal olarak oluşturulmuştur. Bu kapsamda tez içerisinde, nesne tanıma ve öznitelik çıkarma yöntemleri hakkında bilgi verilmiş, tez çalışmasında kullanılan HOG algoritması ve SVM ayrıntılı olarak ele alınmıştır. HOG algoritmasının FPGA üzerinde gerçeklenmesi ve gömülü mikroişlemci üzerinde SVM ile sınıflandırma uygulamasının gerçekleştirilmesi ayrıntılı olarak anlatılmıştır. FPGA üzerinde gerçeklenen HOG ve mikroişlemci üzerinde oluşturulan SVM yapıları birleştirilerek bir yaya tanıma uygulaması yapılmıştır.

Tasarlanan sistem gerçek zamanlı olarak çalışabilecek bir yapıda olmalı ve yaya tanımada kabul edilebilir bir performans ortaya koymalıdır. Uygulamanın gerçeklendiği donanımsal altyapı için taşınabilirlik, maliyet ve güç konuları dikkate alınmalı, tasarım bu kıstaslar gözetilerek yapılmalıdır.

1.3 Orijinal Katkı

HOG algoritması işlenecek verini fazlalığı nedeniyle yüksek hesaplama performansına ihtiyaç duymaktadır. Mikroişlemci üzerinde gerçek zamanlı olarak çalıştırmakta

karşılaşılan zorluk nedeniyle HOG algoritmasının FPGA ve GPU gibi donanımlarla hızlandırılması yoluna gidilmiştir.

HOG algoritması Bauer vd. [12] tarafından GPU kullanılarak hızlandırılmış, Negi vd. [14] tarafından ise algoritmanın hızlandırılması için FPGA kullanılmıştır. Negi vd. yaptıkları çalışmada HOG algoritmasıyla birlikte sınıflandırıcıyı da FPGA üzerinde gerçeklemiş, sınıflandırma için Adaboost sınıflandırıcısını kullanmıştır.

Bu tez çalışmasında HOG algoritması FPGA üzerinde gerçeklenerek hızlandırılmıştır. Sınıflandırma başarımının yüksek olabilmesi amacıyla sınıflandırıcı olarak SVM tercih edilmiştir. SVM sınıflandırıcısının FPGA üzerinde gerçeklenmesinin zorluğu nedeniyle bu görev mikroişlemciye verilmiştir. Sınıflandırıcının mikroişlemci tarafından gerçeklenmesiyle sistem çok daha kolay bir şekilde konfigüre edilebilir hale gelmiştir. Sistemin HOG algoritması dışındaki kısımlarının yazılımsal olarak gerçeklenmesi yaya tanıma dışında, araç tanıma, plaka tanıma gibi farklı uygulamaların hızlı bir şekilde oluşturulabilmesine olanak sağlayacaktır.

Çalışmada diğer mikroişlemcili çözümlerden farklı olarak bir ARM tabanlı gömülü işlemci kullanılmıştır. Dolayısıyla sistem klasik bilgisayarlara göre kabul edilebilir bir performans sunmakla birlikte çok daha az tüketmekte, taşınabilirlik ve maliyet konularında büyük avantaj sunmaktadır.

BÖLÜM 2

NESNE TANIMA

Nesne tanıma, bir görüntü üzerinde önceden bilinen belirli bir nesneyi bulma olarak tanımlanabilir. Nesne tanıma sistemleri tanımlı modeller ve algoritmalar yardımıyla, gerçek dünyadan alınan bir görüntü üzerinde nesneleri tanımaya çalışır. İnsanlar karmaşık bir görüntü üzerindeki nesneleri bile çok küçük bir çaba ile tanırken, nesne tanıma sistemleri birçok basit görüntüde bile hatalı sonuçlar verebilmektedir.

2.1 Nesne Tanıma Yöntemleri

Nesne tanıma yöntemleri iki sınıfa ayrılmaktadır. Bunlar görünüş tabanlı ve öznitelik tabanlı nesne tanıma yöntemleridir [16]. Görünüş tabanlı nesne tanıma yöntemlerinde ilgili nesneye ait bir şablon ya da o nesne için çıkarılmış örnek bir resim kullanılmaktadır. Öznitelik tabanlı nesne tanıma yöntemlerinde ise daha önce ilgili nesne için çıkarılmış öznitelikler görüntü üzerinden elde edilen özniteliklerle karşılaştırmada kullanılır.

2.1.1 Görünüş Tabanlı Nesne Tanıma Yöntemleri

Görünüş tabanlı nesne tanıma yöntemlerinde aranan nesne için seçilmiş örnek imgeler kullanılmaktadır. Bu yöntemlerde nesnenin görünüşünün önemi büyüktür ve görünüşü etkileyen durumlarda nesne tanıma başarımı düşmektedir. Bu durumlar;

- Nesnenin ışık durumunun veya renginin değişmesi,
- Görme açısının değişmesi,
- Nesnenin uzaklığının ya da şeklin değişmesi olarak sıralanabilir.

Doğal ortam içerisinden alınan görüntülerde ya da dinamik uygulamalarda bu yöntemlerin kullanılması ile iyi sonuçlar elde edilemez. Bu yöntemler daha çok özel ortamlardan, özel koşullarda alınmış görüntüler üzerinde kullanılabilir.

Görünüş tabanlı nesne tanıma yöntemlerine "kenar eşleme", "gri ölçekli eşleme" ve "gradyan eşleme" yöntemleri örnek olarak verilebilir. Kenar belirleme teknikleri kullanılarak yapılan nesne tanıma yöntemi,"kenar eşleme" yöntemi olarak adlandırılır. Bunlara Sobel Filtresi, Canny Kenar Saptayıcısı (Canny Edge Detector) örnek olarak verilebilir. Bu yöntemde renk ve ışık koşuları, görüntü üzerindeki kenarları çok etkilemeyeceğinden, sistem performansını genellikle değiştirmemektedir. Kenar eşleme yönteminde öncelikle şablon görüntü üzerindeki kenarlar tespit edilir. Ardından şablon görüntüden elde edilen kenarlar örnek görüntülerden elde edilen kenarlar ile karşılaştırılır.

Kenar eşleme yönteminde görüntü üzerinde yer alan nesneye ait birçok bilgi atılmaktadır. Bunun yerine gri ölçekli görüntü üzerinde nesne tanıma yapan "gri ölçekli eşleme" yöntemi kullanılabilir. Bu yöntem ışık koşullarının değişimine karşı kenar eşleme yöntemine göre daha hassas olsa da, gri ölçekli görüntü üzerinde nesneler ile ilgili çok daha fazla bilgi bulunmaktadır. Bu yöntemde görüntü üzerindeki nesneler, piksel pozisyonu ve piksel parlaklık değerleri kullanılarak çıkarılan, piksel uzaklıkları kullanılarak tanımlanmaktadır [17].

Bir diğer görünüş tabanlı nesne tanıma yöntemi olan gradyan eşleme yönteminde, görüntü üzerinde hesaplanan gradyanlar ışık koşullarından etkilenmeyeceğinden, ışık değişimleri performansı çok etkilememektedir. Gradyan eşleme yönteminde nesne tanımlama işlemi "gri ölçekli eşleme" yöntemine benzer şekilde gradyan uzaklıklarının çıkarılmasıyla olur. Bu yönteme alternatif olarak daha basit bir yöntem olan korelasyon ile eşleme gösterilebilir.

2.1.2 Öznitelik Tabanlı Nesne Tanıma Yöntemleri

Öznitelik tabanlı nesne tanıma yöntemlerinde aranan nesnelere ait uygun öznitelikler kullanılmaktadır. Bir nesneyi tanımlayacak özniteliklerin ortam ya da nesne durumu gibi koşullardan etkilenmeyenlerden seçilmesi esastır. Özniteliklerin genel ya da yerel

olarak seçilmesi nesne tanıma uygulamasının türüne göre değişiklik gösterir. Örneğin bir görüntü üzerinde bir insanın varlığı aranırken genel öznitelikleri kullanmak doğru iken, görüntü üzerinde tam olarak bulunmayan bir insan yüzünden kimliği saptamada yerel öznitelikleri kullanmak doğru olacaktır. Nesne bir görüntü üzerinde genel özniteliklerle kolay olarak ayırt edilebiliyorsa genel öznitelikler tercih edilir. Genel özniteliklere nesnenin genel renk yapısı ve dağılımı, doku yapısı, nesnenin kenar durumları ve merkezi gibi öznitelikler örnek olarak verilebilir. Genel özniteliklerin nesneyi ayırt etmede kullanılamadığı durumlarda yerel özniteliklere başvurulur. Bunlara nesnenin köşeleri, nesne üzerindeki özniteliklerin birbirlerine olan uzaklıkları veya uzaklık oranları, nesneyi çevreleyen eğrilerin yön durumları örnek olarak verilebilir. Gerekli durumlarda yerel ve genel öznitelikler birlikte kullanılmaktadır.

Öznitelik tabanlı nesne tanıma yöntemlerinde nesne tanıma işi iki ana bölümden oluşur. Bunlar öznitelik çıkarma ve sınıflandırmadır. Nesne tanımanın iki aşaması olan sınıflandırma modeli oluşturma ve uygulama aşamalarından her ikisi için de bu öznitelik çıkarma ve sınıflandırma işlemleri gerçekleştirilir. Öncelikle aranacak nesnenin öznitelikleri çıkarılır ve uygun sınıflandırıcı ile sınıflandırma modeli oluşturulur. Ardından uygulama aşamasında görüntü üzerinden aynı metotla öznitelikler çıkarılır ve sınıflandırıcıya oluşturulan sınıflandırma modeli ile birlikte verilerek karşılaştırma yapılır.

2.2 Öznitelik Çıkarma Yöntemleri

Bir giriş verisinin bir öznitelik kümesine dönüştürülmesi işlemi,“öznitelik çıkarma” olarak adlandırılır. Bir görüntü verisi üzerinde gerekli gereksiz birçok bilgi barındırmaktadır. Bu verinin bir algoritma tarafından bu haliyle işlenmesi gereğinden çok daha fazla işlem yüküne ve performans kaybına yol açacaktır. Bu problemin giderilmesi için görüntü üzerindeki gereksiz bilgilerin elenip, görüntünün ilgili algoritma tarafından kullanılabilecek boyutta ve anlamlı veri kümesine dönüştürülmesi gereklidir.

Her öznitelik çıkarma yöntemi her uygulamada başarılı sonuçlar vermemektedir. Herhangi bir öznitelik çıkarma yöntemi ile bir görüntü üzerinden elde edilecek öznitelikler, ilgili uygulamaya göre, aranan nesneler için ayırt edici olmalıdır. Bir

görüntü üzerinde nesne tanımada nesnelerin renk, kenar, şekil ve doku gibi özellikleri kullanılarak, bunlardan uygun öznitelikler elde edilir.

Bir görüntü üzerinden öznitelik çıkarma yöntemleri "düşük seviyeli" ve "yüksek seviyeli" yöntemler olmak üzere ikiye ayrılmaktadır. Düşük seviyeli yöntemler "kenar bulma", "köşe bulma", "bölge analizi" olarak sınıflandırılabilir. Yüksek seviyeli yöntemler ise "bağımsız", "şekil tabanlı", "gradyan tabanlı" ve "şekil eşleştirme tabanlı" yöntemler olarak verilebilir [18].

2.2.1 Düşük Seviyeli Öznitelik Çıkarma Yöntemleri

2.2.1.1 Kenar Bulma

Kenar bulma işlemi görüntü üzerindeki piksel değişimlerinin fazla olduğu yerlerin belirlenmesidir. Görüntü üzerinde yer alan nesneler arasındaki sınırları ifade eden kenarların belirlenmesi, nesne bölümleme gibi daha ileri seviyeli görüntü işleme yöntemleri için temel oluşturmaktadır. Kenar bulmaya yönelik olarak çeşitli yöntemler önerilmektedir. Bunlara Sobel, Prewitt, Roberts filtreleri ve Canny Kenar Saptayıcısı [19] örnek olarak gösterilebilir.

2.2.1.2 Köşe Bulma

Köşe bulma yönteminde ana fikir, köşe noktası etrafında belirli bir komşuluk için her yöndeki hareketin kayda değer bir yoğunluk farkı oluşturmasıdır. Köşe bulma temelli yöntemler bilgisayarla görünün pek çok alanı (nesne takibi, imge çakıştırma, boyutlu geri çatım v.b.) için önem taşımaktadır. Moravec [20] özilintiye dayalı bir nokta saptama yöntemi geliştirmiş, Förstner [21] "özilinti matrisi" olarak bilinen, x ve y yönündeki imge türevlerinin çarpımlarından oluşan matrisi köşe bulmak için kullanmıştır. Harris ve Stephens [22] bu matrisin determinantını ve izini kullanarak farklı bir köşelik ölçüsü önermiştir. Tomasi ve Kanade [23] "oto korelasyon matris"inin özdeğerlerini direkt köşelik ölçüsü olarak almışlardır. Kitchen ve Rosenfeld [24] köşelik ölçüsünü yerel gradyan değeri ve kenar çizgisi boyunca gradyan yönündeki değişim değerinin çarpımı olarak belirlemiştir. Yakın zamanda, imge gradyanı tabanlı köşe bulma yöntemlerinin köşelik ölçülerini ve sonuçlarını karşılaştıran çalışmalar

yapılmıştır. Farklı olarak, bölgeyi morfolojik açıdan inceleyerek köşe biçimli şekilleri yakalayan veya köşeleri parametrik modellerle ifade eden ve bu parametrelere uygunluğuna göre seçim yapan köşe bulma yöntemleri de kullanılmaktadır [25].

2.2.1.3 Bölge Analizi

Görüntü üzerinde nesnenin durumuna göre değişmeyen özel bölgelerin bulunması işi bölge analizi yöntemi olarak adlandırılır. Bu özel bölgelerin bulunmasındaki amaç bir görüntüden bir nesne için farklı geometrik ve foto metrik şartlarda elde edilen bu özel bölgelerin birbirleri ile eşleştirilebilmesini sağlamaktır. Bölge analizi ile ilgi noktası bulma yöntemlerine LoG (The Laplacian-of-Gaussian) [26] , DoG (The Difference-of-Gausssian) [26] , DoH (Determinant of Hessian) [27] ve Maksimum Durağan Uç Bölgeler (MSER) [28] algoritmaları örnek olarak verilebilir [29].

2.2.2 Yüksek Seviyeli Öznitelik Çıkarma Yöntemleri

2.2.2.1 Bağımsız Öznitelik Çıkarma Yöntemleri

Nesne tanımaya yönelik olarak mantıksal ya da algoritmik olarak bir öznitelik çıkarım yöntemi bulunamadığı durumlarda Temel Bileşen Analizi (TBA), Bağımsız Bileşen Analizi (BBA), Doğrusal Ayırma Analizi (DAA) ve Dalgacık Dönüşümü gibi öznitelik çıkarma yöntemleri kullanılabilmektedir [18].

Temel Bileşen Analizi (Principal Components Analysis – PCA) aralarında yüksek korelasyon bulunan çok değişkenli verileri, aralarında korelasyon bulunmayan yeni bir koordinat sistemine dönüştüren istatistiksel bir yöntemdir. Bu dönüşüm, farklı disiplinler tarafından çok değişkenli veri analizinde (multivariate analysis) kullanılmaktadır. Özellikle sinyal işleme uygulamalarında sıkça kullanılan bu yöntem, sayısal görüntülerin de sinyal olarak yorumlanabilmesi sayesinde, son zamanlarda görüntü işleme uygulamalarında da sıkça kullanılmaktadır [18].

Bağımsız Bileşen Analizi (Independent Component Analysis) normal dağılıma sahip olmayan çok değişkenli verilerin aralarındaki istatistiksel bağımlılığı en aza indirecek şekilde gösterimini sağlayarak, içindeki saklı bileşenleri bulmaya çalışmaktadır. Bu bileşenler verinin içerisindeki önemli bazı özellikleri ifade etmektedir. BBA yöntemiyle

sağlanan gösterim, veri sıkıştırma, örüntü tanıma gibi birçok alanlarda veri analizinde kullanılmaktadır [18].

Doğrusal Ayırma Analizi (Linear Discriminant Analysis) değişkenleri doğrusal kombinasyon kümelerine ayırarak, grup içerisinde değerlerin birbirine yakın olmasını, gruplar arasında ise olabildiğince farklı olmasını amaçlar. Bu doğrusal kombinasyonlar diskriminant fonksiyonları ile gösterilirler [18].

Dalgacık dönüşümü, Fourier dönüşümünün durağan olmayan sinyallerdeki eksiklerini gidermek için geliştirilmiş dönüşüm yöntemidir. Bu yöntem, gürültüye karşı daha az hassasiyet göstermekte ve durağan olmayan sinyallere rahatlıkla uygulanabilmektedir. Bundan dolayı sinyal işleme uygulamalarında frekans tabanlı Fourier dönüşümünün yerini ölçek tabanlı dalgacık dönüşümü almıştır.

Dalgacık dönüşümü bu işlem için ana dalgacık adı verilen sınırlı süreli, düzensiz ve asimetrik sinyal parçalarının, ölçeklenmiş ve kaydırılmış hallerini kullanmaktadır. Sinyallerdeki kısa süreli ve keskin değişiklikler, bu dönüşümle daha iyi analiz edilebilmektedir. Başka bir ifadeyle, dalgacık dönüşümü daha iyi zaman - frekans lokalizasyonu sağlamaktadır. Dalgacık dönüşümünde yaygın olarak kullanılan ana dalgacık türleri *Haar, Daubechies, Coiflet, Symlet, Morlet ve Meyer*'dir [18].

2.2.2.2 Şekil Tabanlı Öznitelik Çıkarma Yöntemleri

Şekil tabanlı öznitelik çıkarma yöntemlerine Hough dönüşümü örnek olarak verilebilir. Hough dönüşümü görüntü üzerindeki geometrik şekilleri tespit etmek amacıyla kullanılır. Hough dönüşümü ile tespit edilebilecek geometrik şekiller arasında düzgün doğrular, eğriler, çember ve elipsler, rastgele dağılımlı doğrusal olmayan düzensiz şekiller bulunmaktadır [18].

Bir görüntüdeki doğruların tespiti için kullanılan bir Hough dönüşümü sonucunda, görüntünün içindeki doğrusal olmayan daire ve eğri gibi şekiller yeni görüntüde bulunmayacaktır. Çıktı görüntüsünde sadece düzgün doğrular yer alacaktır. Hough dönüşümü kullanılarak şekil tespiti;

- Kaynak görüntü üzerindeki kenarların belirlenmesi,
- Bir eşikleme yöntemi ile görüntünün ikili forma dönüştürülmesi,

- Her kenar pikselin olası şekilleri oylaması adımlarından oluşmaktadır.

Hough dönüşümü yöntemi kenar bilgisi elde edilmiş gri ölçekli görüntüler üzerine uygulanmaktadır. Hough dönüşümü görüntü üzerinde sıfırdan farklı değerler alan her bir (x,y) koordinat çifti için (2.1) doğru denklemi ile ifade edilen doğrunun "y" eksenini kestiği "b" noktasının hesaplanmasını sağlamaktadır.

$$y_n = a.x_n + b \tag{2.1}$$

Her farklı (a,b) çifti için farklı bir doğru çizilebilir. Öncelikle (a,b) katsayılarının minimum ve maksimum değerleri belirlenir. Belirlenen aralıktaki her bir olası (a,b) noktası için bir toplama hücresi oluşturulur. "b" değerinin tüm olası değeri için "a" değerleri hesaplanarak, her bir hesap için toplama hücresinin değeri bir arttırılır. Bu işlem bittiğinde toplama hücrelerinden yüksek değerli olanlar incelenir. Bunlardan aralarındaki boşlukların yüksek olduğu (a,b) noktaları atıldıktan sonra geriye kalanlar, görüntüde yer alan doğruların ifadesi olarak seçilir. Bundan sonra doğruların başlangıç ve bitiş noktalarının belirlenmesi gerekmektedir. Dik doğrularda "a" ya da "b" değerinin sonsuza gitmesi durumu söz konusu olduğundan, kartezyen koordinatlar yerine kutupsal koordinatlar kullanıldığında, (2.1) eşitliği (2.2)'deki gibi ifade edilir [30].

$$q = x.\cos(\theta) + y.\sin(\theta) \tag{2.2}$$

2.2.2.3 Gradyan Tabanlı Öznitelik Çıkarma Yöntemleri

Gradyan temelli öznitelik çıkarma yöntemleri HOG (Histogram of Oriented Gradients), CoHOG (Co-Occurence Histogram of Oriented Gradients), Color CoHOG (Color Co-Occurence Histogram of Oriented Gradients), CoHED (Co-occurence Histograms of pairs of Edge orientations and color Differences), CoHD (Co-Occurrence Histograms of Color Differences) olarak verilebilir.

Yönlü Gradyanların Histogramı (HOG) algoritması bir görüntünün yerel bölgelere ayrılıp, bu bölge içinde yer alan piksellerin doğrultu ve büyüklük değerleri ile histogram oluşturma olarak tanımlanabilir. Bu histogramlar, görüntünün yerel bir bölgesindeki gradyanların yönelimlerinin sayısını içermektedir. Farklı koşullar altında yüksek başarım sağlayan HOG algoritması ilk defa Shashua [31] ve Dalal [6] tarafından önerilmiştir.

HOG yöntemi ile görüntü “tekli gradyan yönelimleri” şeklinde ifade edildiğinden ilgili görüntünün dokusu istenildiği gibi ifade edilememektedir. Bundan dolayı HOG yönteminin türevi olarak gradyan yönelimleri arasındaki uzaysal ilişkiyi ifade eden CoHOG (Co-Occurence Histograms of Oriented Gradients) yöntemi önerilmiştir. Çoklu gradyan yönelimi tabanlı bir öznitelik çıkarma yöntemi olan CoHOG, görüntüyü gradyan çiftleri olarak ifade etmektedir. Bu sebeple CoHOG öznitelik tanımlayıcıları nesnenin şeklini HOG’dan daha detaylı olarak ifade edebilmektedir [18], [32].

HOG ve CoHOG öznitelik vektörlerinin hesaplanmasında giriş görüntüsü gri ölçekli olarak kullanılmaktadır. Ancak, renkli görüntünün gri ölçekli görüntüye dönüştürülmesi sırasında bilgi kaybı yaşanmaktadır. Color-CoHOG yönteminde CoHOG’dan farklı olarak öznitelik tanımlayıcıları hesaplanırken görüntünün renk bilgisi de kullanılmaktadır. Bu yöntemde renk bilgisi iki aşamada kullanılır. Bunlardan ilki renkli görüntüdeki kenar yönelimleri hesaplanması, ikincisi ise önplanla arka plan arasındaki ayrımı dikkate almak üzere renk eşleşmesinin sonuçlarının kullanılmasıdır [18].

CoHED öznitelik çıkarma yönteminde görüntünün kenar yönelimleri ile kenardaki renk değişimi arasındaki ilişkiye bakılmaktadır. CoHD yönteminde ise görüntü içerisinde verilen bir çizgi üzerindeki üç pikselde meydana gelen renk değerindeki değişim, öznitelik çıkarmada kullanılmaktadır. CoHD yöntemi de CoHOG gibi sadece doku bilgisini dikkate almaktadır [18].

2.2.2.4 Şekil Eşleştirme Tabanlı Öznitelik Çıkarma Yöntemleri

Şekil eşleştirme tabanlı yöntemlerin özelliği nesnenin değişen öteleme, dönme, ölçekleme gibi durumlara rağmen tanınmasıdır. Bu yöntemlere Ölçekten Bağımsız Öznitelik Dönüşümü (Scale Invariant Feature Transform – SIFT), Hızlandırılmış Gürbüz Öznitelikler (Speeded Up Robust Features – SURF), Gradyan Konum ve Yön Histogramı (Gradient Location and Orientation Histogram – GLOH) örnek olarak verilebilir [18].

Ölçekten bağımsız öznitelik dönüşümü yönteminde nesne tanıma, nesnenin boyutu, açısı ve ortamın ışık koşullarından bağımsız olarak gerçekleştirilebilir. SIFT algoritması temel olarak 4 aşamadan oluşmaktadır. Bunlar;

- Ölçekli uzaydaki uç değer noktaların tespit edilmesi,

- Anahtar noktaların bulunması,
- Yönlerin belirlenmesi ve
- Anahtar noktalar için tanımlayıcı vektör oluşturulması olarak sıralanabilir.

Hızlandırılmış gürbüz öznitelikler yönteminde (SURF) ise iki boyutlu Haar dalgacıklarının toplamı temel alınmakta ve entegral görüntüleri kullanılmaktadır. SURF, Hessian matrislerini imge özniteliklerini belirlemek için kullanmaktadır. Burada Haar dalgacıklarının kullanımı ile Hessian bölge algılayıcılarının yaklaşık determinantları hesaplanmaktadır. SURF yönteminde kullanılan entegral görüntüleri, görüntü üzerinde belirli bir dikdörtgen içerisinde kalan piksel değerleri toplamının hızlı bir şekilde hesaplanmasını sağlamaktadır [18], [33].

Gradyan konum ve yön histogramı (GLOH) yönteminde ise öznitelik tanımlayıcısı bir ilgi noktası etrafındaki yerel bir konuma bağımlı gradyanların yön histogramıdır. GLOH yönteminin SIFT yönteminden 2 temel farkı bulunmaktadır. Birincisi, kullanılan ızgara konumunun değiştirilerek öznitelik vektör boyutunun artırılmasıdır. Diğer temel fark ise elde edilen yüksek boyutlu öznitelik betimleyicilerinin PCA yöntemi ile boyutunun azaltılmasıdır. GLOH, SIFT yönteminin sağlamlık ve ayırt edici özelliğini artırmak için tasarlanmış özniteliklerini kullanmaktadır [18].

BÖLÜM 3

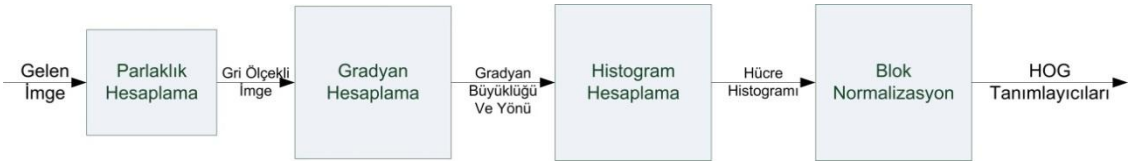
YÖNLÜ GRADYANLARIN HİSTOGRAMI ALGORİTMASI İLE ÖZNİTELİK ÇIKARIMI

Yönlü gradyanların histogramı (HOG) algoritması, gradyan temelli bir öznitelik çıkarım metodudur. Bir imgeyi yerel bölgelere ayırarak bu bölgeler için tanımlayıcılar oluşturur. Oluşturulan HOG tanımlayıcıları imgenin ilgili yerel bölgesindeki imge şeklini kabaca ifade etmektedir. Bu sebeple HOG algoritması başta yaya tanıma ve araç tanıma uygulamaları olmak üzere çeşitli nesne tanıma uygulamalarında kullanılmıştır [6], [9], [34], [12], [13], [35], [14], [36]. Nesne tanımaya yönelik olarak HOG algoritması ile özellik çıkarımı ilk defa Shashua [31] ve Dalal [6] tarafından önerilmiştir.

HOG algoritması bir imgeyi yerel histogramlar dizisi olarak tanımlamaktadır. Her bir yerel histogram, imge üzerinde "hücre" olarak adlandırılan belirli bir bölge için hesaplanan gradyanların, belirlenen yönlerde oluşma sayısının dağılımıdır. Bu gradyanların hesaplanmasında piksel parlaklığı bilgisi kullanılır. HOG algoritmasında öznitelik çıkarımı işlemi temel olarak üç aşamadan meydana gelir.

- Gradyan büyüklük ve yönlerinin hesaplanması
- Her bir hücre için gradyan histogramlarının oluşturulması
- Hücre histogramlarının bloklar halinde birbirleri ile normalize edilmesi

Bir imge üzerinde gradyan hesabı yapılmadan önce imgenin renk sinyallerinden her piksel için parlaklık bilgisi hesaplanarak, gri ölçekli yeni bir imge elde edilir. HOG algoritması gradyan hesabı yaparken bu gri ölçekli imgeyi kullanır.



Şekil 3.1 Yönlü Gradyanların Histogramı (HOG) algoritmasının aşamaları

3.1 Parlaklık Hesaplama

Özniteliği çıkarılacak imge öncelikle gri ölçekli imgeye dönüştürülmelidir. Bunun için imgenin renk sinyallerinden her bir piksel için parlaklık bilgisi hesaplanmalıdır. Gelen imgenin RGB renk formatında olduğu düşünüldüğünde (3.1) ile verilen eşitlik kullanılarak, piksel parlaklık bilgisinden oluşan gri ölçekli imge yaklaşık olarak elde edilebilir.

$$Y = 0.299R + 0.587G + 0.114B \text{ [37]} \tag{3.1}$$

3.2 Gradyan Hesaplama

Gradyan hesaplama aşamasında ilk olarak imge üzerindeki her bir nokta için yatay ve dikey yönde parlaklık değişimleri hesaplanır. Bu değişimler gradyan büyüklüklerinin yatay ve dikey bileşenleridir. Bu bileşenleri hesaplamada genellikle işlem basitliğine rağmen iyi performans sunan Sobel Filtresi tercih edilir [38]. 1 boyutlu Sobel Filtresi kullanımında yatay ve dikey gradyan büyüklükleri (3.2) ile verilen eşitliklerle hesaplanır.

$$\begin{aligned} g_x(x,y) &= i(x+1,y) - i(x-1,y) \\ g_y(x,y) &= i(x,y+1) - i(x,y-1) \end{aligned} \tag{3.2}$$

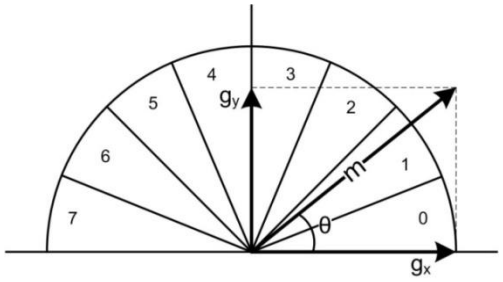
(3.2)eşitliklerinde kullanılan $i(x,y)$ ifadesi imgenin (x,y) noktasındaki parlaklık değerini verir. Gradyan büyüklüğü m ve yönü θ sırasıyla (3.3) ve (3.4) ile verilen eşitliklerle hesaplanır.

$$m(x,y) = \sqrt{g_x(x,y)^2 + g_y(x,y)^2} \tag{3.3}$$

$$\theta(x,y) = \arctan\frac{g_x(x,y)}{g_y(x,y)} \tag{3.4}$$

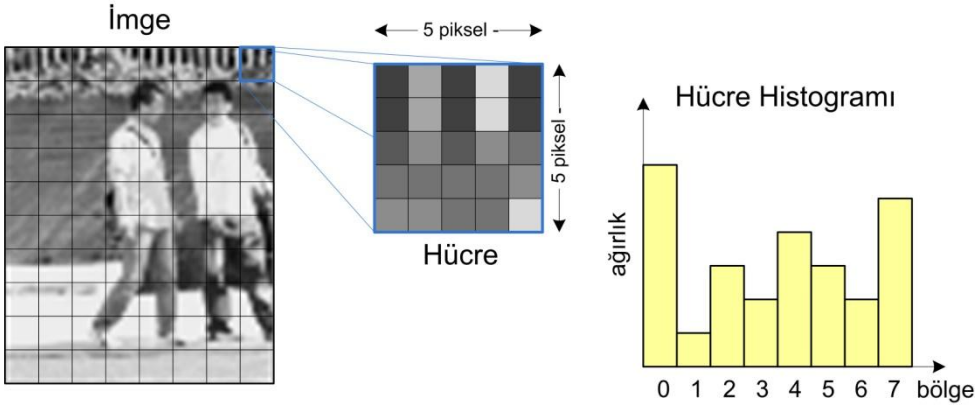
3.3 Histogram Hesaplama

Bu aşamada imge birkaç piksel karelik alanlara ayrılarak incelenir. Bu alanlar "hücre" olarak adlandırılan, uygulamalarda boyutları genellikle 5x5 ya da 8x8 piksel olarak seçilen karelerdir [13], [12]. Seçilen hücre boyutuna göre, her bir hücre için 5x5 ya da 8x8 adet gradyan büyüklüğü "m"ve yönü "θ" bilgisi bulunmaktadır.



Şekil 3.2 Gradyan yön bölgeleri

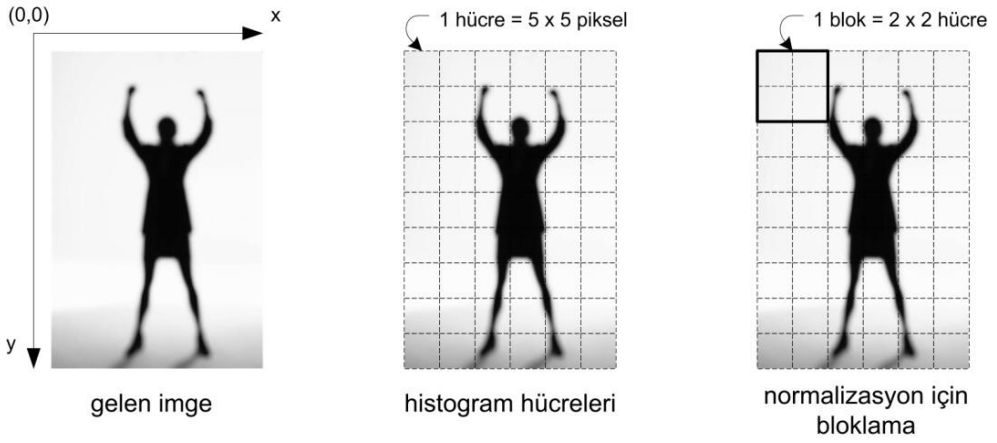
Histogram hesabında gradyan yönleri uygulamaya göre 0 – 180 derece ya da 0 – 360 derece arasında eşit bölgelere ayrılır. Burada gradyan yön bölgeleri 0 – 180 derece arasında sekiz bölge olarak değerlendirilecektir. Şekil 3.2'de gradyan yön bölgeleri verilmektedir.



Şekil 3.3 HOG hücresi ve hücre histogramı gösterimi

Histogram hesabı yapılırken hücre içerisindeki her gradyan büyüklüğü, sahip olduğu açıya göre histogramda ilgili bölgelere belirli bir metoda göre dağıtılır. Hesaplamada sağladığı kolaylık nedeniyle gradyan büyüklüğü dağıtımında genellikle lineer interpolasyon kullanılmaktadır. Bu metotta hesaplanan gradyana komşu iki gradyan yön bölgesinin histogramdaki değerleri (3.5) ile verilen eşitlikler ile arttırılır. Burada

$h(n+1)$ ve $h(n+1)$ histogramda komşu iki gradyan yön bölgesinin değeridir. "α" ile verilen değer büyüklük dağıtım çarpanı, "b" ise gradyan yön bölgelerinin açı değeridir.



Şekil 3.4 HOG algoritmasında kullanılan hücre ve blok yapıları

Gradyan büyüklüğü dağıtım metodu olarak Dalal [38] trilineer interpolasyonu kullanmıştır. Bu metotta imgenin bir noktası için hesaplanan gradyanlar, sınır bölgelerinde çevre hücrelerin histogramlarını da etkiler.

$$\begin{aligned} h(n) &= h(n) + (1-\alpha)m(x,y) \\ h(n+1) &= h(n+1) + \alpha m(x,y) \\ \alpha &= \frac{\theta(x,y) - \theta_n}{b} \end{aligned} \tag{3.5}$$

3.4 Blok Normalizasyon

Bir önceki aşamada elde edilen hücre histogramları bloklar halinde gruplanıp, blok içerisindeki hücreler birbirleri ile normalize edilir. Blok boyutları genellikle 2x2 ya da 3x3 hücre olarak alınır [6], [12], [14]. Burada blok boyutları 2x2 hücre olarak alınacaktır. Dalal ve Triggs blok normalizasyon için dört farklı normalizasyon yöntemini denemişlerdir. Bu normalizasyon yöntemleri $L_1 - norm$, $L_1 - sqrt$, $L_2 - norm$ ve $L_2 - hys$ olarak (3.6), (3.7), (3.8), (3.9) eşitlikleri ile verilmiştir.

$$L_1 - norm: f = \frac{v}{(\|v\|_1 + e)} \tag{3.6}$$

$$L_1 - sqrt : f = \sqrt{\frac{v}{(\|v\|_1 + e)}} \tag{3.7}$$

$$L_2 - norm : f = \frac{v}{\sqrt{(\|v\|_2^2 + e^2)}} \tag{3.8}$$

$$L_2 - hys : f = \frac{v}{\sqrt{(\|v\|_2^2 + e^2)}} \quad , \quad f_{\max} = 0.2 \tag{3.9}$$

Yukarıda verilen normalizasyon metotlarında bir blok içerisinde yer alan tüm histogram değerleri kaskad eklenerek " v " vektörü oluşturulmuştur. " $\|v\|_n$ " gösterimi n. dereceden normu ifade etmektedir ve (3.10) eşitliği ile verilmektedir. " f " vektörü, " v " vektörünün normalize edilmiş halidir. (3.9) eşitliği ile verilen $L_2 - hys$ normalizasyonu $L_2 - norm$ normunun çıkış vektörü " f " nin elemanlarının maksimum 0,2 değeriyle sınırlandırılmasıyla hesaplanır. $L_1 - sqrt$ normalizasyonu ise $L_1 - norm$ normalizasyonunun çıkış vektörü " f " nin elemanlarının karekökü alınarak hesaplanır.

$$\|v\|_n = \sqrt[n]{\sum_i |x_i|^n} \tag{3.10}$$

Blok normalizasyonda normalizasyon bloğu, histogram hücreleri üzerinde kayar pencere olarak gezdirilir. Dolayısıyla mxn hücrelik bir imge için kxk blok normalizasyon yapıldığında; $(m-(k-1)).(n-(k-1))$ tane blok normalizasyon işlemi yapılır. Gradyan yön bölgesi sayısı " p "olarak alındığında blok normalizasyon sonucu oluşacak HOG tanımlayıcısı sayısı (3.11) eşitliğiyle hesaplanır.

$$Ht = p.k^2.(m-(k-1)).(n-(k-1)) \tag{3.11}$$

BÖLÜM 4

DESTEK VEKTÖR MAKİNELERİ

Destek Vektör Makineleri (Support Vector Machines – SVM) ilk defa Vapnik [39] tarafından önerilen, istatistiksel öğrenme teorisinde dayalı kontrollü bir sınıflandırma algoritmasıdır. SVM'nin temelini oluşturan matematiksel algoritmalar başlangıçta iki sınıflı doğrusal verilerin sınıflandırılması problemi için tasarlanmış, daha sonraki çalışmalarla çok sınıflı ve doğrusal olmayan verilerin sınıflandırılmasından kullanılmak üzere genelleştirilmiştir. Temel olarak SVM'nin çalışma mantığı iki farklı sınıftan oluşan bir veri öbeğini birbirinden ayırabilecek en uygun karar fonksiyonun bulunması ilkesine dayanır. Bu karar fonksiyonu çok boyutlu bir veri uzayında bir hiper-düzlem'e karşılık düşmektedir [40]. Destek Vektör Makineleri el yazısı, ses, yüz tanıma, gen, protein, kanser hücresi sınıflandırma, uzaysal veri analizi gibi birçok alanda yapılan çalışmalarda kullanılmaktadır [41], [42], [43], [44], [45].

SVM'ler doğrusal ve doğrusal olmayan destek vektör makineleri olmak üzere iki sınıfa ayrılmaktadır. Bu çalışmada doğrusal destek vektör makineleri kullanılmıştır, dolayısıyla bu konu üzerinde durulacaktır.

4.1 Doğrusal Destek Vektör Makineleri

Doğrusal destek vektör makineleri kullanılan verilerin doğrusal ayrılabilme ya da ayrılamama durumuna göre incelenecektir.

4.1.1 Doğrusal Olarak Ayrılabilen Veriler İçin SVM

Destek vektör makineleri kullanımı ile yapılan sınıflandırma işlemlerinde iki sınıfa ait örneklerin, eğitim verisi ile elde edilen bir karar fonksiyonu ile birbirinden ayrılması amaçlanır. Bu iki sınıf genellikle (-1,+1) sınıf etiketleri ile gösterilmektedir. Eğitim sonucu elde edilen karar fonksiyonu kullanılarak eğitim öbeğini en uygun şekilde ayırabilecek hiper-düzlem belirlenir.

$x_i \in R^N$ eğitim kümesindeki örnekler için öznitelikler vektörü, $y_i \in \{-1,+1\}$ ise sınıfı etiketlerini göstermek üzere pozitif ve negatif etiketli örnekleri birbirinden ayırabilen birçok hiper-düzlem çizilebilir. Burada SVM'nin amacı kendisine en yakın noktalar arasındaki uzaklığı maksimum yapan hiper-düzlemi bulmaktır. Bulunan bu hiper-düzleme optimum hiper-düzlem ve bu düzleme komşu olan, sınır genişliğini sınırlandıran noktalar ise destek vektörleri adı verilir. Bu hiper düzlem üzerindeki herhangi bir x noktası (4.1) eşitliğini sağlamaktadır. Burada "w" hiper-düzlemin normal vektörü (ağırlık vektörü), "b" eğilim değerini göstermektedir.

$$w.x_i + b = 0 \tag{4.1}$$

SVM eğitimi için kullanılacak k elemanlı bir veri aşağıdaki gibi kabul edilsin;

$$E = \{x_i, y_i\} \ , \quad i = 1,2,...,k$$

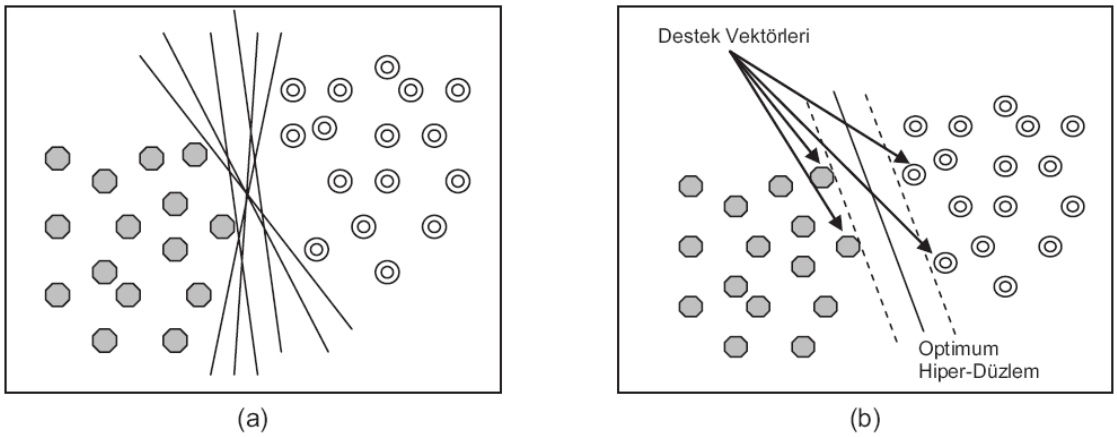
Optimum hiper-düzleme ait eşitsizlikler bu durumda (4.2) ve (4.3) eşitsizlikleri ile yazılırlar.

$$y_i = +1 \quad \text{için,} \ w.x_i + b \geq +1 \tag{4.2}$$

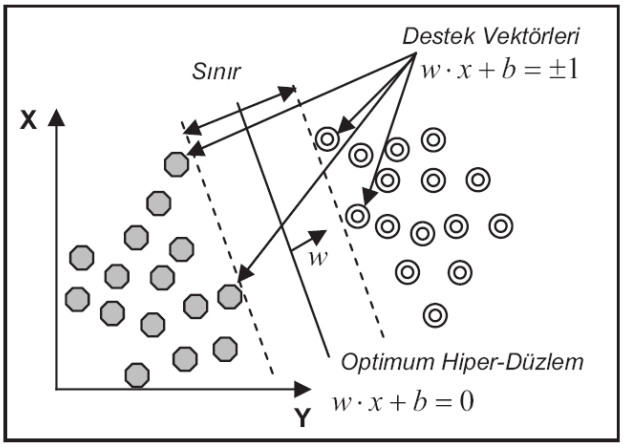
$$y_i = -1 \quad \text{için,} \ w.x_i + b \leq -1 \tag{4.3}$$

Bu eşitsizlikleri sağlayan hiper-düzlemin iki tarafındaki noktalara olan diz uzaklıkları toplamı sınır olarak adlandırılır. Bu sınırı maksimum yapan hiper-düzlem optimum hiper-düzlemi verir. Optimum hiper-düzlemin belirlenebilmesi için bu düzleme paralel olan ve sınırları oluşturan iki hiper-düzlemin belirlenmesi gerekir. Bu hiper-düzlemler destek vektörleri ile oluşturulmakta, eşitlik (4.4) ile ifade edilmektedirler.

$$w.x_i + b = \pm 1 \tag{4.4}$$



Şekil 4.1 İki sınıflı bir veri öbeğini ayırmada kullanılacak hiper düzlemler (a) ve optimum hiper düzlem (b) [40]



Şekil 4.2 Doğrusal olarak ayrılabilen veri öbekleri için optimum hiper-düzlemin belirlenmesi [40]

Optimum hiper-düzlemin bulunması için uygun "w" ve "b" değerleri hesaplanmalıdır. Eşitlik (4.4) ile ifade edilen sınır hiper-düzlemleri'nin orijine olan uzaklıkları sırasıyla $|1-b|/\|w\|_1$ ve $|-1-b|/\|w\|_1$ olarak hesaplanır. Bu iki hiper-düzlem arası uzaklık ise $2/\|w\|$ kadardır. Optimum hiper-düzlem bulunurken, bu düzlemin sınıra olan uzaklığı maksimuma çıkarılmaya çalışılır. Bunun için ise $\|w\|_1$ ifadesinin minimum hale getirilmesi gerekmektedir. Bu durumda maksimum sınırın bulunması için $\min\left[\frac{1}{2}\|w\|_1^2\right]$ ifadesi (4.5) koşuluna bağlı olarak hesaplanmalıdır [39].

$$y_i(w.x_i + b) - 1 \geq 0 \ , \ y_i \in \{1, -1\} \tag{4.5}$$

Bu optimizasyon probleminin çözümünde Lagrange denklemleri kullanılır. Bu problem Lagrange denklemleri kullanılarak eşitlik (4.6)'daki gibi yazılır. Burada verilen "α_i" değerleri pozitif Lagrange çarpanlarıdır.

$$L_p = \frac{1}{2}\|w\|_1^2 - \sum_{i=1}^{k} \alpha_i y_i (w.x_i + b) + \sum_{i=1}^{k} \alpha_i \tag{4.6}$$

Eşitlik (4.6)'daki verilen ifadenin çözümü oldukça karmaşıktır. Bu yüzden Karush-Kuhn-Tucker (KKT) koşulları kullanılarak sadeleştirilir ve (4.9) eşitliği elde edilir. Bu problem için gerekli KKT koşulları (4.7) ve (4.8) ile verilmiştir.

$$\frac{\partial L_p}{\partial w} = 0 \Rightarrow w = \sum_i \alpha_i y_i x_i \tag{4.7}$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_i \alpha_i y_i = 0 \tag{4.8}$$

$$L_d = \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \ , \quad \alpha_i > 0, \forall_i \tag{4.9}$$

Eğitim öbeğinde bulunan her örnek için bir tane Lagrange çarpanı bulunmaktadır. Bu Lagrange çarpanlarının birçoğunun değeri çözüm sırasında sıfır olacak, geriye kalan pozitif "α_i" değerli "x_i" vektörleri ise destek vektörlerini oluşturacaktır.

4.1.2 Doğrusal Olarak Ayrılamayan Veriler İçin SVM

Birçok sınıflandırma uygulamasında veri öbeklerinin doğrusal olarak ayrılması mümkün değildir. Eğitim verilerinin bir kısmının optimum hiper-düzlemin diğer tarafında kalmasından kaynaklanan bu problem bir hata değişkeninin tanımlanması ile çözülmektedir. Optimum hiper düzlemin hesaplanabilmesi için (4.2) ve (4.3) eşitsizlikleri eğitim hatası sapması değeri "ξ_i" ile birlikte tekrar yazılarak (4.10), (4.11) ve (4.12) eşitsizlikleri elde edilir.

$$y_i = +1 \quad \text{için,} \ w.x_i + b \geq +1 - \xi_i \tag{4.10}$$

$y_i = -1$ için, $w.x_i + b \le -1 + \xi_i$ (4.11)

$\xi_i \ge 0, \forall_i$ (4.12)

Bu durumda bir "x_i" örneğinin yanlış sınıflandırılmış olması için "$\xi_i < 0$" olmalıdır. Doğru sınıflandırılmış bir "x_i"örneği "$0 < \xi_i < 1$" arasında olması bu verinin aslında iki sınır hiper-düzlemi arasında yer aldığı anlamına gelmektedir. Doğrusal olarak ayrılamama durumu için veri öbeği uzayına bir "C" üst sınırı eklenir. Bu üst sınır Lagrange çarpanlarının "$0 < \xi_i < C$" arasında kalmasını sağlar [39]. Eşitlik (4.6) bu durum için tekrar yazıldığında (4.13) eşitliği elde edilir. Bu ifadedeki μ_i değeri, ξ_i değerinin pozitif olmasını sağlayan Lagrange parametresidir.

$$L_p = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{k}\xi_i - \sum_{i=1}^{k}\alpha_i\{y_i(w.x_i + b) - 1 + \xi_i\} - \sum_{i=1}^{k}\mu_i\xi_i \quad (4.13)$$

Eşitlik (4.6)'da olduğu gibi (4.13) eşitliği ile verilen ifadenin de çözümü oldukça karmaşıktır. Bu yüzden Karush-Kuhn-Tucker (KKT) koşulları güncellenerek tekrar uygulanır ve ifade sadeleştirilerek (4.17) eşitliği elde edilir. Bu problem için gerekli KKT koşulları (4.14), (4.15) ve (4.16) ile verilmiştir.

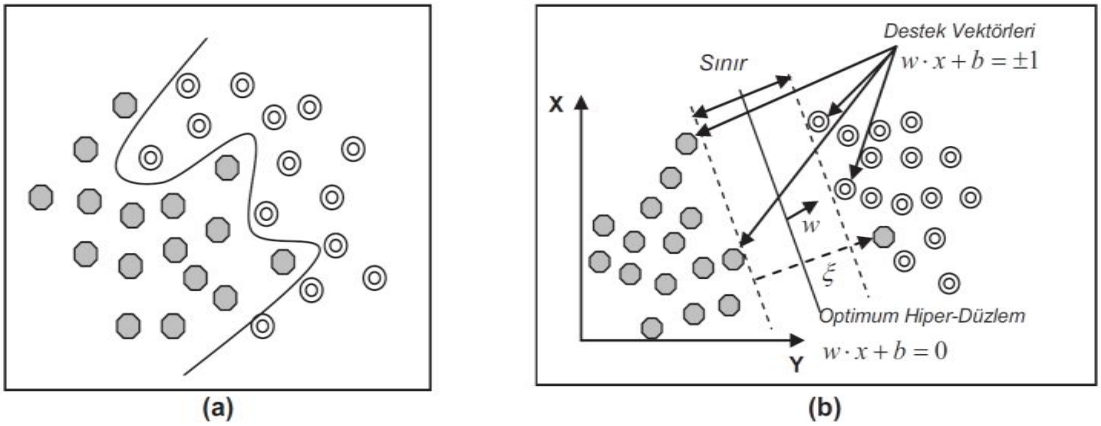
$$\frac{\partial L_p}{\partial w} = 0 \Rightarrow w = \sum_i \alpha_i y_i x_i \quad (4.14)$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_i \alpha_i y_i = 0 \quad (4.15)$$

$$\frac{\partial L_p}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \quad (4.16)$$

$$L_d = \sum_{i=1}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i x_j \ , \quad C > \alpha_i > 0, \forall_i \quad (4.17)$$

Burada $C > \alpha_i > 0, \forall_i$ aralığında yer alan pozitif "α_i" değerli "x_i" vektörleri destek vektörlerini oluşturacaktır.



Şekil 4.3 Doğrusal olarak ayrılamayan veri öbekleri (a) için optimum hiper-düzlemin (b) belirlenmesi [40]

BÖLÜM 5

YÖNLÜ GRADYANLARIN HİSTOGRAMI ALGORİTMASININ FPGA ÜZERİNDE GERÇEKLENMESİ

Bu bölümde Yönlü Gradyanların Histogramı (Histogram of Oriented Gradients – HOG) algoritmasının FPGA üzerinde donanımsal gerçeklenmesi aşamaları anlatılacaktır. İlk olarak HOG algoritmasının donanımsal gerçeklenmesi konusunda daha önce yapılmış olan çalışmalardan bahsedilecektir. Ardından tasarlanan sistem hakkında genel bilgi verilip, tasarım aşamasında dikkat edilen hususlar belirtilecektir. Son olarak algoritmanın bölümlerinin nasıl gerçeklendiği anlatılacaktır.

HOG algoritmasının FPGA üzerinde gerçeklemesinde VHDL donanım tanımlama dili kullanılmıştır. Geliştirme ortamı olarak “Altera Quartus II” [46], “Xilinx ISE” [47], “Modelsim” [48] ve “Sigasi” [49] yazılımları kullanılmıştır.

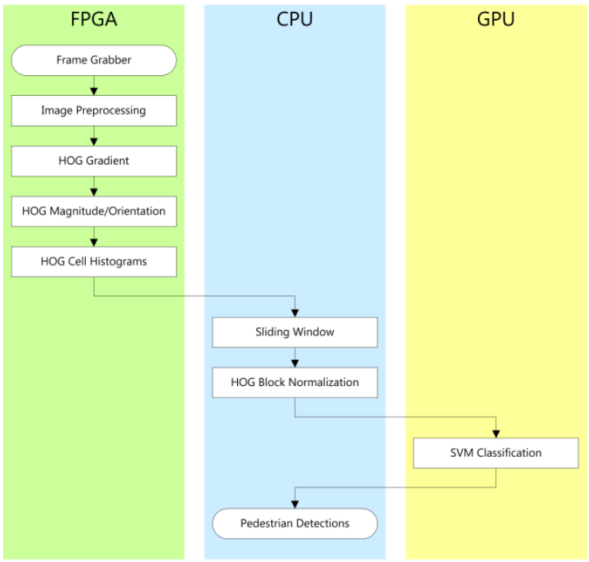
5.1 HOG Algoritmasının Donanımsal Gerçeklemesi Üzerine Yapılan Çalışmalar

Literatürde HOG algoritmasının birçok donanımsal ya da yazılım – donanım melez gerçeklenmesi üzerine çalışmalar bulunmaktadır [34], [12], [13], [35], [14], [50]. Bu çalışmalardaki ortak amaç, HOG algoritmasının ihtiyaç duyduğu yoğun matematiksel işlemlerin donanımsal olarak gerçeklenerek hızlandırılması ve sistemin performansını arttırılmasıdır. Bu sayede HOG algoritması ile gerçek zamanlı çalışabilen sistemler mümkün olmaktadır.

Cao ve Deng, HOG tanımlayıcıları ve entegral haritasını kullanarak, trafik levhalarını tanıyan FPGA tabanlı gerçek zamanlı bir sistem tasarlamıştır [34]. Sistemde bir araç kamerasından alınan görüntü FPGA üzerinde işlenip, dur levhaları tespit edildiğinde

uyarı verilmektedir. *Xilinx Virtex-4* FGPA'sı üzerinde gerçeklenen bu tasarımda 752x480 piksel çözünürlüğünde saniyede 60 kare işlenebilmektedir.

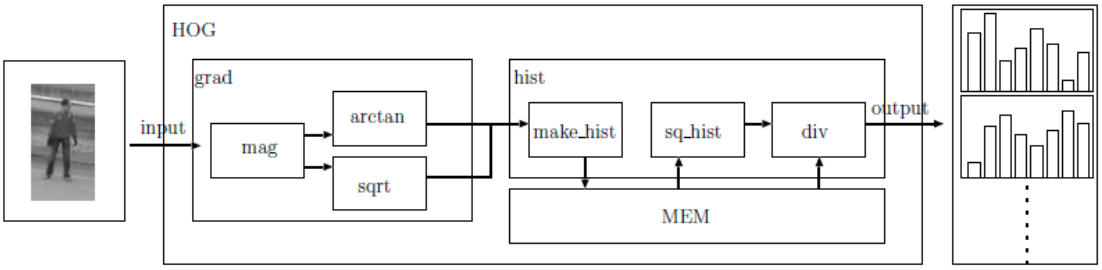
Bauer vd. FPGA, mikroişlemci, grafik kartı tabanlı melez bir sistem oluşturarak HOG algoritması ile yaya tanıma uygulaması yapmıştır [12]. Tasarlanan sistemde HOG algoritmasının gradyan hesaplama ve histogram oluşturma blokları FPGA üzerinde, blok normalizasyon bloğu mikroişlemci üzerinde gerçeklenmektedir. HOG tanımlayıcılarını kullanarak sınıflandırma işlemi grafik kartında SVM ile yapılmaktadır. Son olarak yaya tanıma işi mikroişlemci üzerinde yapılır. HOG algoritması gradyan hesaplama bloğunda hücre boyutu 8 piksel, normalizasyonda kullanılan blok boyutu 2 hücre olarak seçilmiştir. Histogram hesabında $0-180$ derece arasında 9 gradyan yön bölgesi tayin edilmiştir. Tasarlanan bu melez sistemde 800x600 piksel çözünürlüğünde saniyede 20 kare işlenebilmektedir. INRIA yaya veritabanı [8] ile yapılan testler sonucu Dalal ve Triggs'in [6] çalışmasına yakın sonuçlar elde edilmiştir.



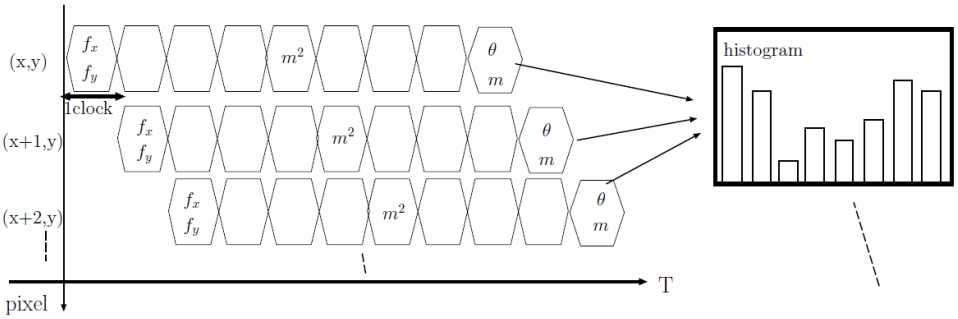
Şekil 5.1 FPGA, mikroişlemci, grafik kartı tabanlı melez HOG gerçeklemesi [12].

Kadota vd. HOG algoritmasını donanımsal olarak gerçekleyerek bir yaya tanıma uygulaması yapmıştır [13]. Yapılan çalışmada HOG algoritmasının aşamaları sırasında karşılaşılan işlemsel güçlükler için yeni yaklaşımlar önerilmiştir. HOG algoritması gradyan hesaplama bloğunda hücre boyutu 5 piksel, normalizasyonda kullanılan blok boyutu 3 hücre olarak seçilmiştir. Histogram hesabında $0-180$ derece arasında 8 gradyan yön bölgesi tayin edilmiştir. *Altera Starix II* FPGA'sı üzerinde gerçeklenen

sistemde 10 HOG işlem birimi paralel çalışarak 640x480 çözünürlüğünde saniyede 30 kare işlenebilmektedir.



Şekil 5.2 Kadota vd.'nin donanımsal HOG gerçeklemesi blok mimarisi [13].



Şekil 5.3 Paralel çalışan HOG işlem birimleri ile histogram üretimi [13].

Negi vd. FPGA üzerinde gerçek zamanlı bir yaya tanıma uygulaması yapmıştır [14]. Yapılan çalışmada Kadota vd.'nin [13] HOG algoritmasında karşılaşılan işlemsel güçlükler için önerdiği yaklaşımlardan yararlanılmıştır. Çıkarılan HOG tanımlayıcıları yine FPGA üzerinde gerçeklenen AdaBoost sınıflandırıcısında kullanılmaktadır. HOG algoritması gradyan hesaplama bloğunda hücre boyutu 5 piksel, normalizasyonda kullanılan blok boyutu 3 hücre olarak seçilmiştir. Histogram hesabında $0-180$ derece arasında 8 gradyan yön bölgesi tayin edilmiştir. Normalizasyon sırasında işlem yükünü azaltmak için histogram değerleri bir eşik değerine göre ikileme (binarization) işlemine sokulmaktadır. *Xilinx Virtex-5* FPGA'sı üzerinde gerçeklenen sistemde 320x240 çözünürlüğünde, saniyede 62,5 kare işlenebilmektedir. NICTA yaya veritabanında [51] 64x80 piksellik örnekler kullanılarak yapılan başarım ölçümünde %20,7 yanlış pozitif oranına karşı %96,6 tanıma oranı elde edilmiştir.

5.2 Tasarlanan Sistem Hakkında Bilgi

Bu projede HOG algoritmasının donanımsal gerçeklemesi tasarlanırken bazı kıstaslar belirlenmiştir. Bunlardan ilki platformdan bağımsızlıktır. VHDL dili ile tasarlanan sistemde platforma bağımlı hazır bloklar kullanılmamıştır. Böylece tasarlanan sistem *Xilinx*, *Altera*, *Actel* gibi farklı firmalar tarafından üretilen birçok platform üzerinde gerçeklenebilir. Dikkat edilen diğer bir husus, tasarlanan sistemin mümkün oldukça modüler ve parametrik bir yapıda olmasıdır. Tasarlanan sistemde kullanılan bloklar modüler ve parametrik bir yapıya sahiptir. Sistemde en üst seviyeden yapılan birkaç parametre değişikliği ile sistemin çözünürlüğü, gradyan hücre boyutu, normalizasyon blok boyutu gibi özellikleri değiştirilebilmektedir. Son olarak sistem gerçek zamanlı çalışabilmelidir. Bunun için tasarım sırasında sistem frekansını düşürecek devre yapılarından kaçınılmış, HOG algoritmasının aşamaları sırasında karşılaşılan bazı karmaşık işlemler için çeşitli yaklaşımlar uygulanmıştır [13].

Tasarlanan HOG bloğu *Xilinx ISE* ve *Altera Quartus II* yazılımları ile sentezlenmiş, *Modelsim* yazılımı ile benzetimleri yapılmıştır. Sentezlenen tasarım *Altera DE0 Nano* [52] geliştirme kartı üzerinde test edilmiştir.

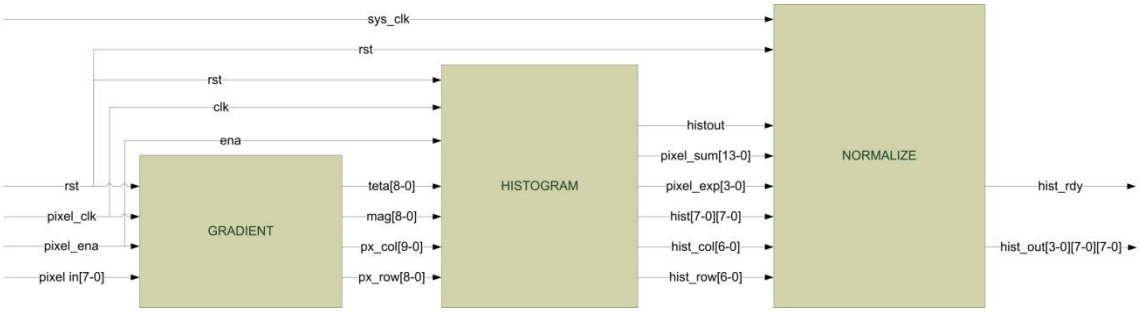
5.3 Tasarlanan Sistemin Blokları

Gerçeklenen sistemin blok şeması Şekil 5.4'te verilmektedir. Tasarım "gradyan hesaplama", "histogram çıkarma" ve"normalizasyon" bloklarından oluşmaktadır. Gradyan hesaplama bloğunda gelen her piksel için bir gradyan büyüklüğü ve yönü bilgisi hesaplanır. Bu piksel için hesaplanan gradyan büyüklüğü ve yönü, pikselin imge üzerindeki konumu bilgisi ile birlikte histogram çıkarma bloğuna verilir. Histogram çıkarma bloğu her hücre için bir histogram bilgisi üretir. Üretilen histogram değerleri ve hücre konumu bilgisini, blok normalizasyon işlemi için normalizasyon bloğuna verir. Normalizasyon bloğu gelen hücre histogramlarını bir blok içinde birbirleri arasında normalize eder ve normalize edilmiş bilgiyi bir hazır (ready) sinyali ile birlikte dışarı verir.

Sistemde iki saat sinyali kullanılır, bunlar "piksel saati" ve "sistem saati" dir. Sistem saati tasarım gereği piksel saatinin iki katı veya üzeri bir frekansa sahip olmalıdır. Her

blokta aktif (enable) ve sıfırla (reset) sinyal girişleri bulunmaktadır. Sistem reset edildiğinde bloklar içerisindeki sayıcılar, FIFO'lar, kaydediciler sıfırlanır. Yeni bir görüntü akışı başlatılmadan sistem sıfırlanmalıdır. Gradyan hesaplama ve histogram çıkarma blokları bir "piksel mevcut" (pixel enable) sinyali ile eş zamanlı olarak aktif olmaktadır. Bu bloklar piksel saati ile çalışırlar. Normalizasyon bloğu ise bu iki bloktan daha yüksek frekansta sistem saati ile çalışır ve "histogram hazır" (histogram ready) sinyali ile aktif olur.

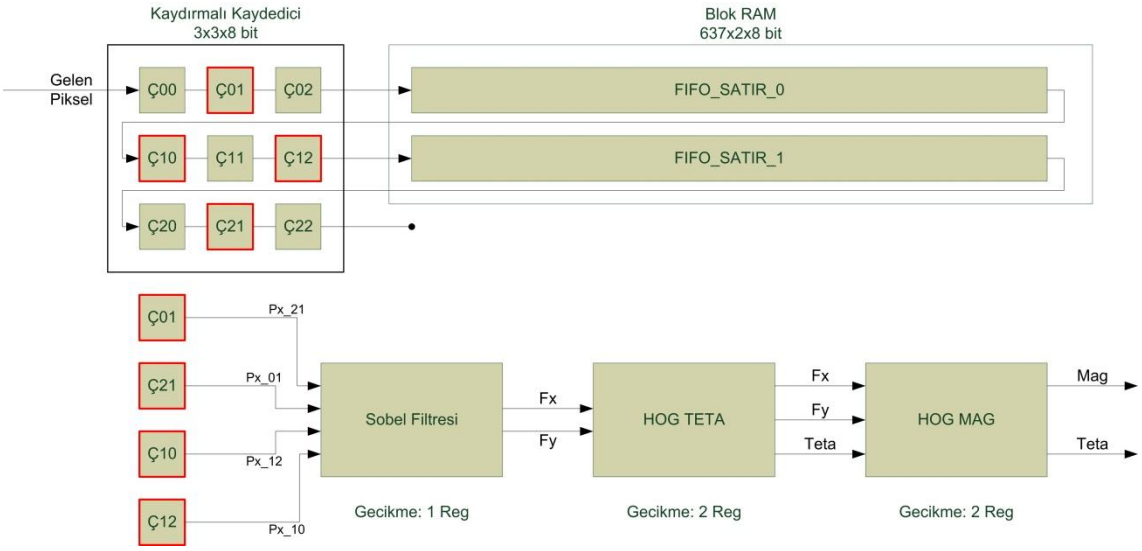
Sistem çözünürlüğü parametrik olmakla birlikte varsayılan olarak 640x480 seçilmiştir. HOG hücre genişliği 5 piksel, blok genişliği 2 hücre olarak kabul edilmiştir.



Şekil 5.4 Tasarlanan HOG yapısının blok şeması

5.4 Gradyan Hesaplama

Yatay ve dikey yöndeki gradyan büyüklerini hesaplamada Sobel Filtresi kullanılmaktadır. Bunun için gelen pikseller 3x3 piksel boyutunda kaydedici ve 637 piksel boyutunda 2 FIFO'dan oluşan yapıya gönderilir. Bu yapı Şekil 5.5'de verilmektedir. Sobel Filtresi bloğu, kaydedici üzerinden Sobel maskesi için gerekli olan piksel değerlerini alarak yatay ve dikey gradyan büyüklüklerini eşitlik (3.2) de verildiği gibi hesaplar.



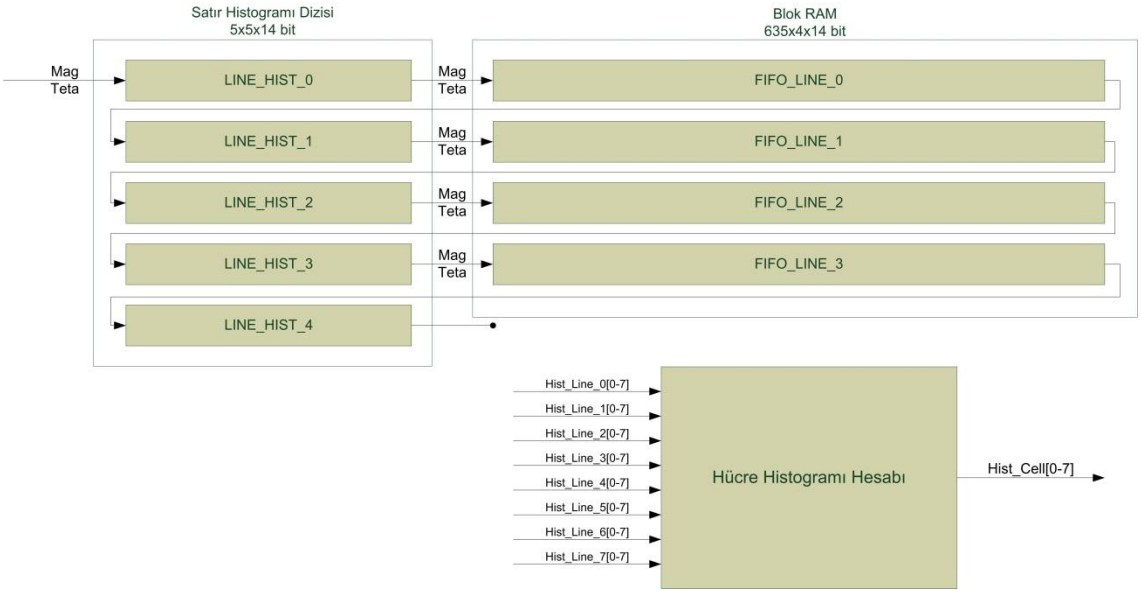
Şekil 5.5 HOG gradyan hesaplama bloğu yapısı

Gradyan büyüklüğü "m" ve yönü "θ" değerlerini sırasıyla (3.3) ve (3.4) eşitlikleri vermektedir. Yalnız bu eşitlikleri donanımsal olarak gerçeklemek oldukça zahmetlidir. Bu zahmetli işlemleri yapmak yerine burada Kadota vd.'nin yaklaşımına [13] benzer bir yaklaşım kullanılmıştır. Gradyan büyüklüğü "m" ve yönü "θ" değerlerinin hesaplamada doğruluk tabloları (look-up-table) kullanılır. Öncelikle yatay ve dikey gradyan büyüklüklerinden, gradyan yönü "θ" hesaplanır. Bunun için "arctan" işlemi için hazırlanan doğruluk tablosu kullanılır. İleriki işlemlerde kolaylık sağlaması açısından 180 derecelik alan 256 bit ile ifade edilir. Hesaplanan gradyan yönü "θ" ve gradyan büyüklüğü bileşenleri kullanılarak gradyan büyüklüğü "m" hesaplanır. Burada gradyan büyüklüğü ile gradyan açısı arasında (5.18)ile verilen eşitlikler kullanılır. Gradyan büyüklüğünün hesabında hazırlanan "$0^0 - 45^0$" arası $1/\cos(\theta)$ ve "$45^0 - 90^0$" arası $1/\sin(\theta)$ doğruluk tabloları kullanılmaktadır. Gradyan hesaplama bloğunda gradyan büyüklüğü "m" ve yönü "θ" değerleri 5 saat darbesi gecikmeli olarak hesaplanır.

$$m = gx.\frac{1}{\cos(\theta)}, \quad m = gy.\frac{1}{\sin(\theta)} \tag{5.18}$$

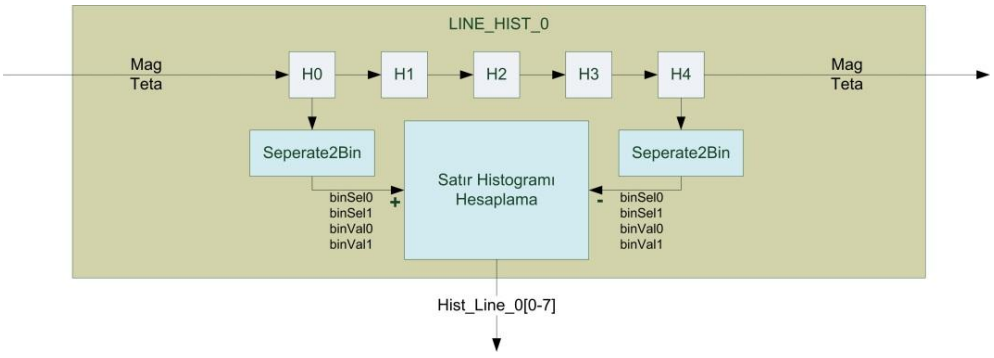
5.5 Histogram Hesaplama

Histogram hesabında hücre boyutu 5 piksel olarak belirlenmiştir. Histogram hesaplama bloğu, 5x5 piksel boyutunda "satır histogramı" dizisi, 635 piksel boyutunda FIFO ve "hücre histogramı hesaplama" bloklarından oluşmaktadır. Bu bloklar Şekil 5.6'da verilmektedir. Satır histogramı hesaplama bloğunda her gelen gradyan bilgisi için histogram değeri hesaplanmaktadır. Daha sonra hesaplanan histogram değerleri hücre histogramı hesaplama bloğunda toplanarak, hücre histogram değeri elde edilir.



Şekil 5.6 HOG histogram hesaplama bloğu yapısı

Satır histogramı hesaplama bloğunda her gelen gradyan değeri için bir toplama ve bir çıkarma işlemi yapılır. Gelen gradyan değeri satır histogramına eklenirken, çıkan gradyan değeri satır histogramından çıkarılır. Böylece her "t" anı için satır histogramı doğru şekilde hesaplanmış olur. Bu bloğun yapısı Şekil 5.7'de verilmektedir.



Şekil 5.7 HOG satır histogramı hesaplama

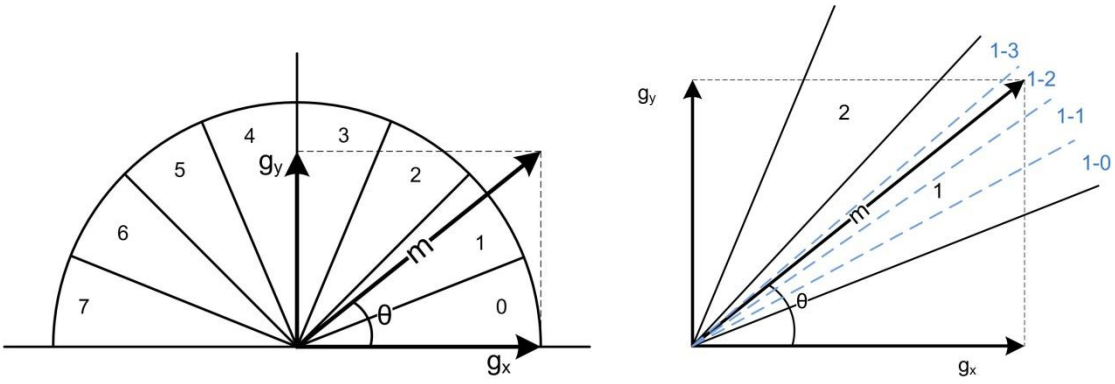
Gradyan büyüklüğü ” m ” ve yönü “ θ ” değerlerinden histogram gradyan yön bölgelerine dağıtma işlemi bu blok içerisinde yapılmaktadır. Tasarımda $0^0 - 180^0$ arasındaki bölge 8 gradyan yön bölgesine ayrılmıştır. Her bir gradyan yön bölgesi içerisinde ise 4 alt bölge bulunur. Gradyan yön bölgeleri ve alt bölgeleri Şekil 5.8'de verilmektedir. Burada gradyan büyüklüğü iki komşu bölge arasında dağıtılır. Dağıtım yapılırken gradyanın bulunduğu alt bölgeye göre (5.19)'de verilen koşullu ifade kullanılır.

$$
\begin{aligned}
&h(n) = h(n) + m(x,y) \\
&h(n+1) = h(n+1)
\end{aligned}
\qquad \left\{ n.b < \theta < n.b + \frac{b}{4} \right\} \text{ için}
$$

$$
\begin{aligned}
&h(n) = h(n) + 0.75m(x,y) \\
&h(n+1) = h(n+1) + 0.25m(x,y)
\end{aligned}
\qquad \left\{ n.b + \frac{b}{4} < \theta < n.b + \frac{b}{2} \right\} \text{ için}
$$

$$
\begin{aligned}
&h(n) = h(n) + 0.5m(x,y) \\
&h(n+1) = h(n+1) + 0.5m(x,y)
\end{aligned}
\qquad \left\{ n.b + \frac{b}{2} < \theta < n.b + \frac{3.b}{4} \right\} \text{ için}
$$

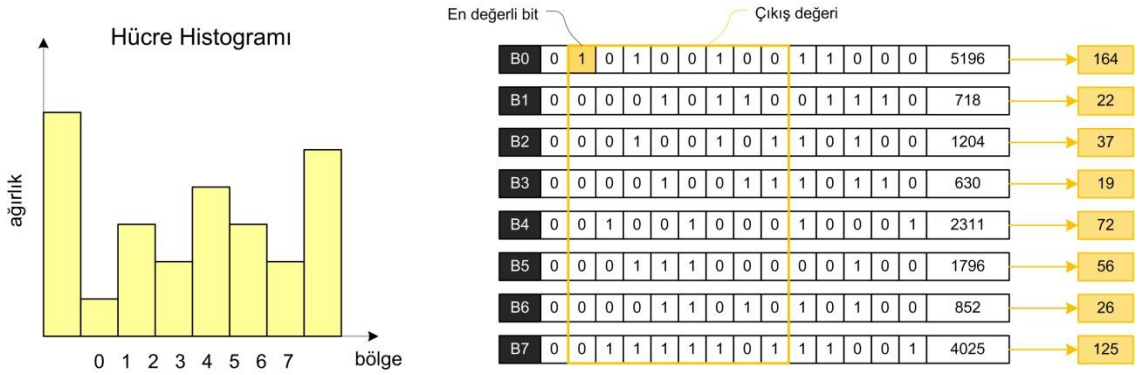
$$
\begin{aligned}
&h(n) = h(n) + 0.25m(x,y) \\
&h(n+1) = h(n+1) + 0.75m(x,y)
\end{aligned}
\qquad \left\{ n.b + \frac{3.b}{4} < \theta < (n+1).b \right\} \text{ için}
\qquad (5.19)
$$



Şekil 5.8 HOG gradyan yön bölgeleri

Histogram hesaplama bloğunda her hücre için 8 gradyan yönü bilgileriyle histogram hesaplanır. Çözünürlüğü (wxh) olan bir imge için hücre boyu 5x5 piksel seçildiğinden toplam $(\frac{w}{5} x \frac{h}{5})$ hücre histogramı hesaplanır. Dolayısıyla 640x480 çözünürlüğünde bir görüntü için, histogram hesaplama bloğunda 128x96 adet hücre histogramı hesaplanır. Histogram hesaplama bloğuna gelen gradyan büyüklüğü 9 bit ile ifade edilir. Bir hücre histogramında tüm gradyanların aynı yönde olması ihtimali göz önünde

bulundurulduğunda, 25x9 bitlik değer için 14 bitlik bir alan bulunmalıdır. Dolayısıyla hücre histogramı hesaplama bloğunda her gradyan yön bölgesi en kötü durumu karşılamak için 14 bit ile ifade edilmektedir. Hücre histogramı hesaplama bloğu histogram değerini kendi içerisinde normalize eder ve dışarıya histogram değerini "8 bölge x 8 bit" formunda verir. Bunun için hücre histogramındaki 8 bölgenin değerleri incelenir. 8 bölgeden en yüksek değerlisi bulunur ve bu bölge değerinin en değerli biti, çıkış değerinin en değerli biti olarak kabul edilir. Bu bit değeri hücre ön normalizasyon çarpanı olarak adlandırılır. Her histogram bölgesi için bu bit değerinden itibaren 8 bit değeri blok dışarısına verilir. Ayrıca blok dışına normalizasyon bloğunda kullanılmak üzere "hücre ön normalizasyon çarpanı" ve tüm histogram bileşenlerinin toplam değeri verilmektedir. Histogram bloğu içerisinde yapılan bu ön normalizasyon işlemi Şekil 5.9'da verilmektedir.



Şekil 5.9 HOG histogram bloğu çıkış normalizasyonu

5.6 Blok Normalizasyon

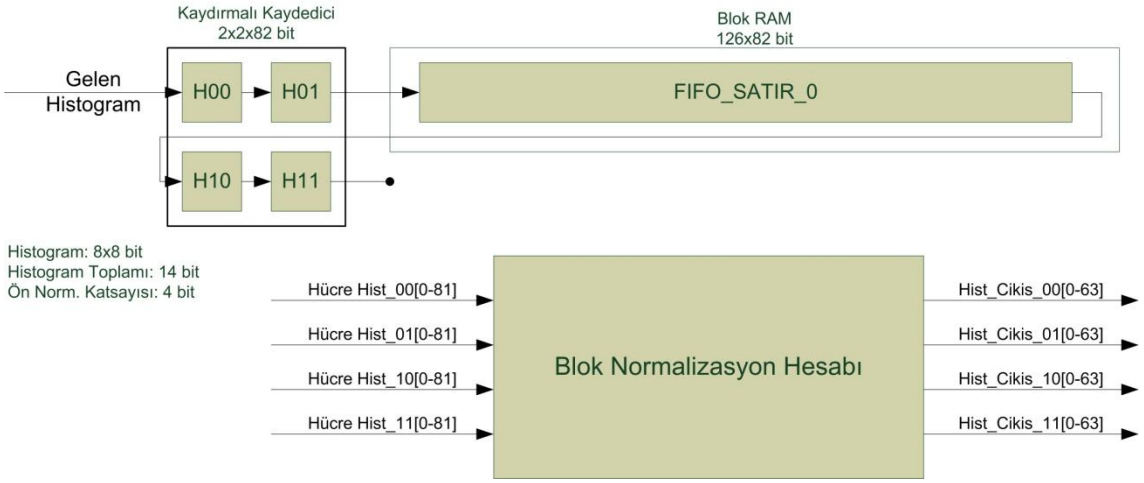
Bir önceki aşamada elde edilen hücre histogramları bloklar halinde gruplanıp, blok içerisindeki hücreler birbirleri ile normalize edilir. Blok boyutu 2x2 hücre olarak belirlenmiştir. Blok normalizasyon yapılırken hücre histogram bileşenleri değerleri, hücre histogram bileşenlerinin toplamı değeri ve "hücre ön normalizasyon çarpanı" kullanılmaktadır. Normalizasyonda diğer karmaşık yöntemlerin donanımsal gerçeklemesinde karşılaşılacak güçlük nedeniyle $L_1 - norm$ yöntemi kullanılmıştır. Bu yöntem (5.20) eşitliği ile verilmektedir.

$$L_1 - norm: f = \frac{v}{(\|v\|_1 + e)} \tag{5.20}$$

Normalizasyon bloğu içerisinde “2 x 2 x 82 bit” boyunda kaydırmalı kaydedici, “126 x 82 bit” boyunda FIFO ve blok normalizasyon hesaplama blokları bulunur. Normalizasyon bloğu yapısı Şekil 5.10’da verilmektedir. Her t anı için 4 hücre histogramı değeri alınarak “blok normalizasyon hesabı” bloğunda normalizasyon işlemi yapılır. Bir histogram kaydedici bloğu “8 bölge x 8 bit” histogram bileşenleri değerinden, 14 bit histogram bileşenleri toplamı değerinden ve 4 bit hücre ön normalizasyon çarpanı değerinden oluşur. Blok normalizasyon hesaplama bloğu kendisine gelen histogram değerlerini, 14 bit histogram bileşenleri toplamı değeri ve 4 bit hücre ön normalizasyon çarpanı değerini kullanarak normalize eder. Her t anı için 4 hücre histogramı değeri de dışarı verilir. Dolayısıyla 640x480 çözünürlüğünde bir görüntü için ile verilen eşitlikte değerler yerine yazıldığında 264160 adet HOG tanımlayıcısı oluşturulur. Burada “g” gradyan yön bölgesi sayısı, “m” ve “n” görüntünün çözünürlük değerleri, “h” hücre boyu ve “k” blok normalizasyon boyudur.

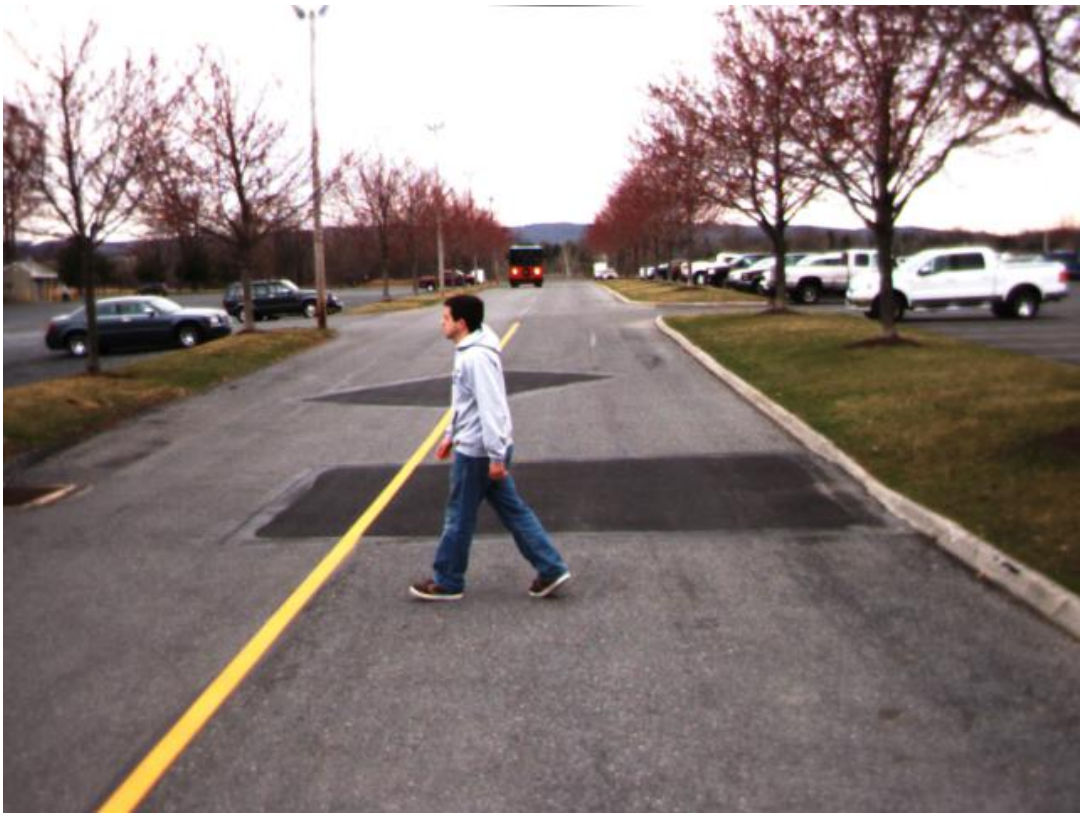
$$Ht = g.k^2.(\frac{m}{h} - (k-1)).(\frac{n}{h} - (k-1)) \tag{5.21}$$

$$Ht = 8.2^2.(\frac{640}{5} - (2-1)).(\frac{480}{5} - (2-1)) = 264160 \tag{5.22}$$

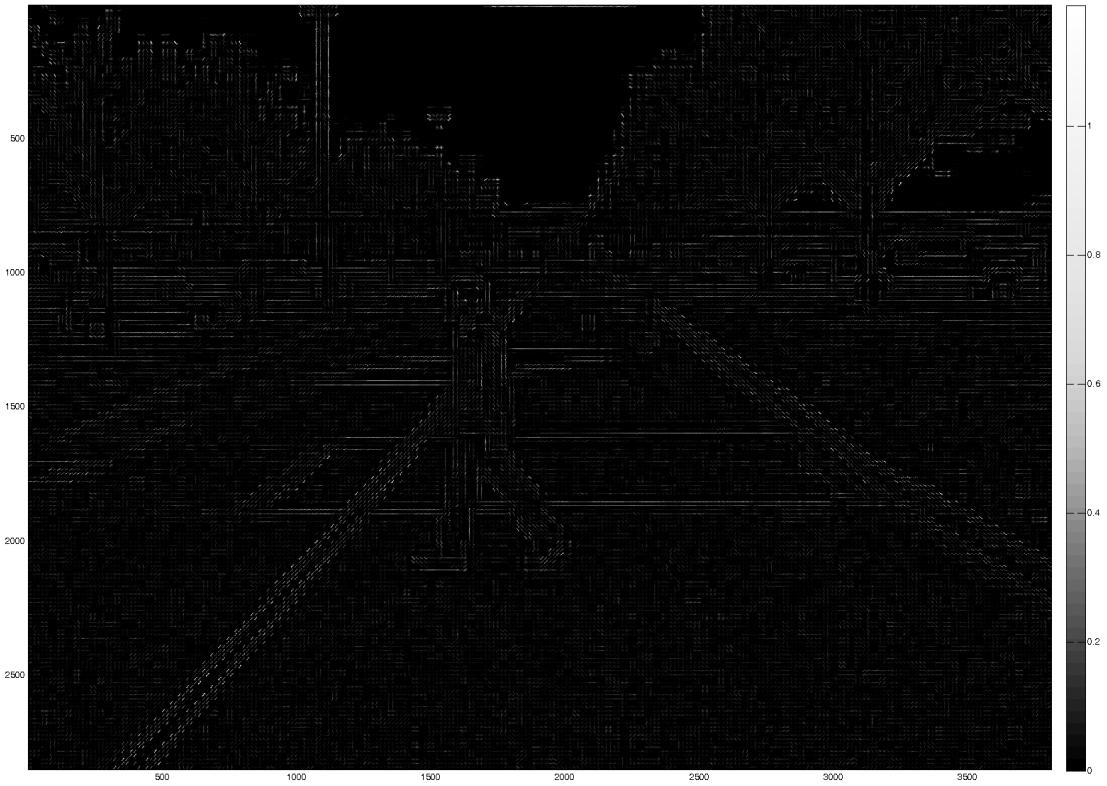


Şekil 5.10 HOG normalizasyon bloğu

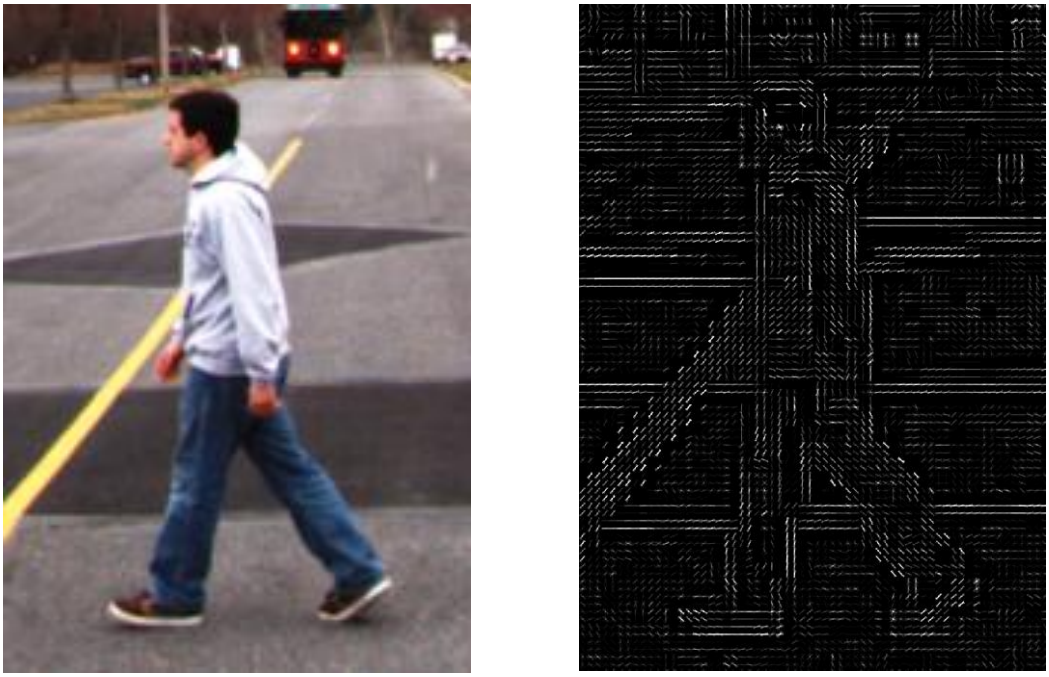
Aşağıda tasarlanan sisteme gönderilen görüntü ve oluşturulan özniteliklerin yön haritası Şekil 5.11, Şekil 5.12 ve Şekil 5.13'de verilmektedir. Çıkarılan özniteliklerle *MATLAB* programında yön haritası oluşturulmuştur.



Şekil 5.11 HOG test giriş görüntüsü



Şekil 5.12 HOG test çıkış öznitelikleri yön haritası



Şekil 5.13 HOG test görüntüsünde yaya ve çıkarılan özniteliklerin yön haritası

BÖLÜM 6

DESTEK VEKTÖR MAKİNESİ İLE GÖMÜLÜ MİKROİŞLEMCİ ÜZERİNDE SINIFLANDIRMA

Bu bölümde bir yazılımsal SVM gerçeklemesi olan *SVM*^{light} [53] kütüphanesinin bir gömülü mikroişlemcili sistem üzerinde çalıştırılmasının aşamaları anlatılacaktır. İlk olarak kullanılacak mikroişlemcili sistem hakkında bilgi verilecek, ardından bu sistemin yazılımsal altyapısı oluşturulacaktır. Altyapı oluşturulduktan sonra *SVM*^{light} kütüphanesi hazırlanan mikroişlemcili sisteme göre derlenecektir. Son olarak hazırlanan bu sistemde bir sınıflandırma uygulaması yapılacaktır.

Gömülü mikroişlemcili sistemin yazılımsal altyapısını oluşturmada ve kütüphanelerin derlenmesinde geliştirme ortamı olarak, bir *Linux* dağıtımı olan *Ubuntu* [54] işletim sistemi kullanılmıştır.

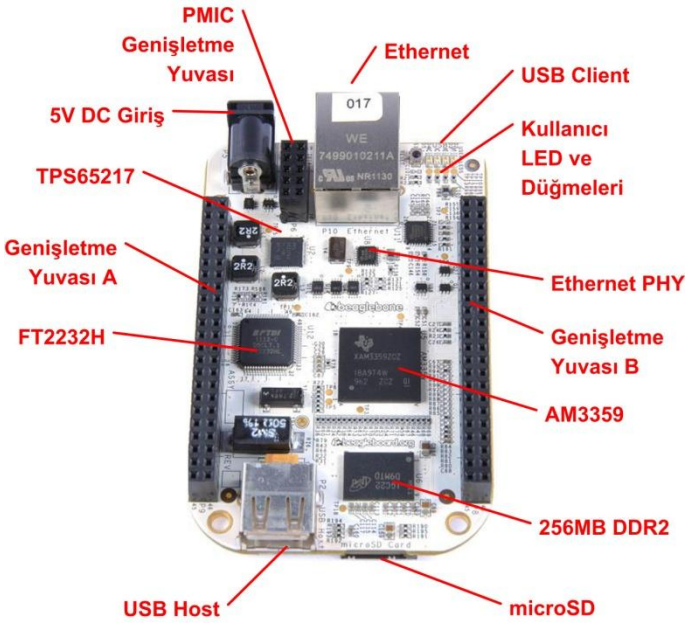
6.1 Gömülü Mikroişlemcili Sistemin Hazırlanması

Bu çalışmada nesne tanımaya yönelik olarak donanım ve yazılım temelli melez bir sistem oluşturulmuştur. Oluşturulan bu sistemin gerekleri belirlenirken, bu sistemin daha sonra tasarlanabilecek ürünler için altyapı oluşturabileceği göz önünde bulundurulmuştur. Bu nedenle sistemin yazılımsal kısmını gerçeklemede standart PC yerine, güç, maliyet, taşınabilirlik avantajları nedeniyle bir gömülü mikroişlemcili sistem kullanılması kararlaştırılmıştır.

6.1.1 Kullanılan Mikroişlemcili Sistem

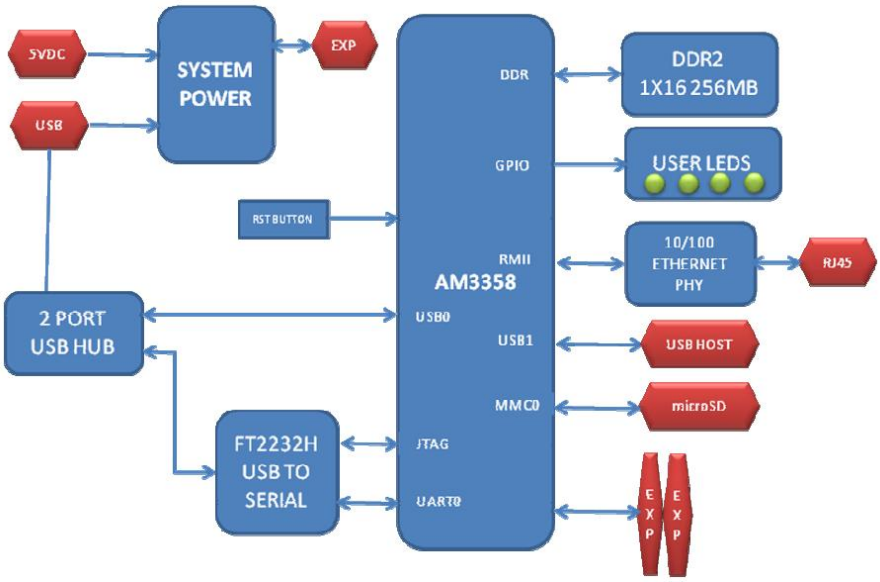
Sistemin yazılımsal kısmını gerçeklemede mikroişlemcili sistem olarak *BeagleBone* tercih edilmiştir. BeagleBone Texas Instruments tarafından üretilen düşük güç tüketimli *ARM* mikroişlemcili bir tek kart bilgisayardır. Barındırdığı (*SPI, I2C, GPIO(65), GPMC, RS232, CANBUS, LVDS, MMC vb.*) giriş – çıkış arayüzleri ile uygulama geliştirmede esneklik sağlar. Yaklaşık kredi kartı büyüklüğünde boyutlara ve 40 gram ağırlığa sahip olması taşınabilirlik konusunda büyük avantaj oluşturur. Tüm sistemin güç tüketimi yük altında 2,5 watt'tır. Üzerinde barındırdığı *ARM Cortex A8* mimarisine sahip mikroişlemci ile *Linux*, *Windows Embedded* gibi standart işletim sistemlerini çalıştırabilir ve bu nedenle bu işletim sistemleri için oluşturulmuş tüm uygulama ve kütüphanelerin kullanımına olanak sağlar [55]. Kısaca özetlenecek olursa;

- Giriş – çıkış arayüzleri,
- Boyut ve ağırlık,
- Güç tüketimi,
- Performans,
- Standart işletim sistemi desteği özellikleri için sahip olduğu avantajlar nedeniyle mikroişlemcili sistem olarak *BeagleBone* [56] tercih edilmiştir.



Şekil 6.1 *BeagleBone*gömülü mikroişlemcili sistemi [55]

BeagleBone Texas Instruments tarafından üretilen *ARM Cortex A8* mimarisine sahip *AM3359 Sitara* [57] mikroişlemcisini kullanmaktadır. Kart üzerinde bu mikroişlemcinin kullanımına sunulmuş 256 MB boyutunda *DDR2 RAM* bellek bulunmaktadır. Mikroişlemcinin hata ayıklama arayüzleri *FTDI FT2232H* entegresi kullanılarak *USB* arayüzü üzerinden kullanıcıya sunulmaktadır. Kart üzerinde bir kalıcı bellek bulunmamaktadır. Önyükleyici, çekirdek, dosya sistemi ve kullanıcı bilgileri *microSD* kart üzerinde tutulmalıdır. Bunun için kart üzerinde *microSD* kart bağlantısı bulunmaktadır. Bunların dışında BeagleBone üzerinde mikroişlemcinin ethernet arayüzü için fiziksel katman entegresi ve ethernet bağlantısı, *USB* bağlantıları, kullanıcı *LED*'leri ve düğmeler ve diğer arayüzler için genişletme bağlantıları bulunur. Kart üzerindeki elemanların ihtiyaç duyduğu tüm gerilimler Texas Instruments tarafından üretilen *TPS65217* [58] entegresi tarafından karşılanır. Bu entegreye güç girişi 5V DC güç kaynağı ya da *USB* arayüzü üzerinden sağlanabilir. Bunun için kart üzerinde *USB* ve standart adaptör bağlantıları da bulunmaktadır.



Şekil 6.2 *Beaglebone*sistem blok şeması [55]

6.1.1.1 ARM Mimarisi

ARM (Acorn RISC Machine) mimarisi *RISC* tabanlı mikroişlemci ailesinin en yaygın türüdür. Günümüzde yeryüzündeki 32-bit gömülü işlemcilerin %75'i *ARM* mimarisini kullanmaktadır [59]. *ARM* mimarisi sahip olduğu düşük güç tüketimi özelliğiyle öne

çıkmaktadır. Bu nedenle günümüzde taşınabilir elektronik cihazların hemen hepsi *ARM* mimarisine sahip bir mikrodenetleyici ya da mikroişlemciye sahiptir. İlk olarak 1980'li yıllarda Acorn Computer şirketi tarafından geliştirilmiştir [60]. Halen *ARM* şirketi tarafından tasarımı yapılmakta olan *ARM* mimarileri lojik yapı olarak pazarlanmakta, bu lojik yapıları birçok firma kendi ürünlerine entegre ederek kullanmaktadır. *ARM* şirketi fiziksel olarak işlemci üretimi yapmamaktadır.

ARM mimarisi *RISC* temelli olması nedeniyle benzer yapılar için *x86* türü mikroişlemcilere oranla daha düşük transistöre ihtiyaç duymaktadır. Bu durum daha az güç tüketimini ve daha düşük üretim maliyetlerini mümkün kılmaktadır. *ARM* mikroişlemcili yongalarda genellikle merkezi işlem birimine ek olarak birçok çevresel birim de bulunmaktadır. Bu da *ARM* mikroişlemcili sistemlerde üretim maliyetlerini düşürmekte, baskı devre tasarımlarını kolaylaştırıp, devre boyutlarını küçülmesini mümkün kılmaktadır.

ARM firması tarafından kullanım alanları ve performans gereklerine göre yıllar içerisinde çeşitli mimariler geliştirilmiştir. Çizelge 6.1'de *ARM* mikroişlemci mimarileri ve işlemci aileleri verilmektedir.

Çizelge 6.1 *ARM* mikroişlemci mimarileri ve işlemci aileleri

Mimari	**İşlemci Ailesi**
ARMv1	ARM1
ARMv2	ARM2, ARM3, Amber
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM8, ARM9TDMI, FA526
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale, FA626TE, Feroceon,
ARMv6	ARM11
ARMv6-M	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1
ARMv7-A	ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-
	Scorpion, Krait, PJ4/Sheeva, Swift
ARMv7-M	ARM Cortex-M3
ARMv7-R	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7
ARMv7E-M	ARM Cortex-M4
ARMv8-A	ARM Cortex-A53, ARM Cortex-A57, X-Gene, Denver

ARM işlemcilerinin mimari, komut kümesi ve birimler arası arayüz tasarımlarında 1995 yılında yayınlanan "*ARM* Referans Kılavuzu" temel alınır. *ARM* mimarili işlemciler için günümüzde üç profil tanımlanmıştır.

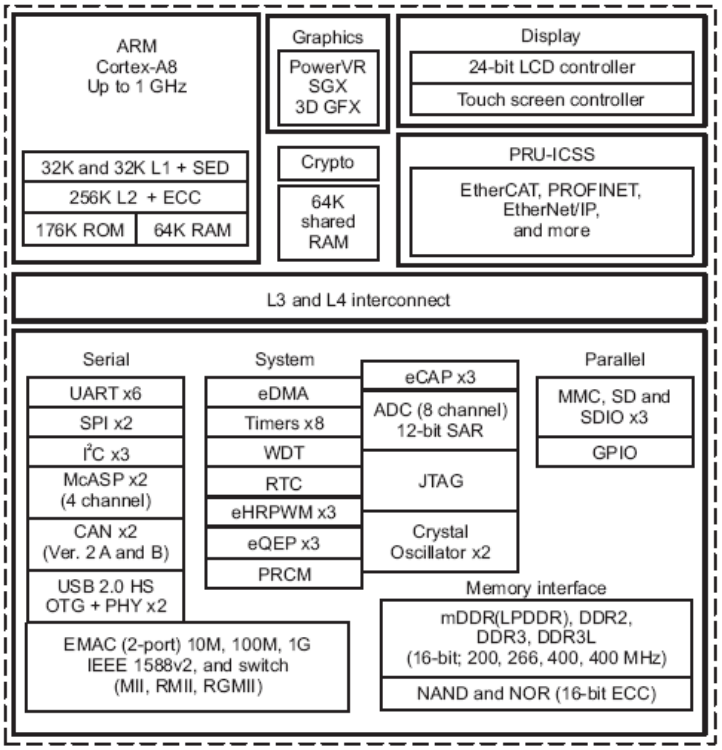
- Uygulama profili: Yüksek performanslı *Cortex-A* serisi işlemciler.
- Gerçek-zamanlı profil: Gerçek-zamanlı uygulamalara yönelik *Cortex-R* serisi işlemciler.
- Mikrodenetleyici profili: *Cortex-M* serisi mikrodenetleyiciler.

6.1.1.2 AM3359 Mikroişlemcisi

AM3359 mikroişlemcisi Texas Instruments tarafından üretilmiş yüksek performanslı ARM Cortex-A8 mimarisine sahip bir gömülü mikroişlemcidir. Mikroişlemci üzerinde ARM merkezi işlem birimi dışında birçok çevresel birim bulunmaktadır. Bu çevresel birimler;

- *NEON SIMD* yardımcı işlemcisi,
- *SGX530 3D* grafik işlemcisi,
- Programlanabilir gerçek-zaman ünitesi,
- *LCD* ekran ve dokunmatik arayüz kontrol birimi,
- *USB* arayüz modülü,
- Gigabit ethernet modülü,
- *CANBUS, UART, SPI, I2C* seri iletişim modülleri,
- 12-bit *ADC* ve *PWM* modülleri,
- *AES, SHA, PKA, RNG* donanımsal kripto hızlandırıcıları,
- *mDDR, DRR2, DDR3* hafızalar için harici hafıza kontrolörü (EMIF),
- Genel amaçlı hafıza kontrolörü (*GPMC*) olarak verilebilir.

Bu işlemcinin sahip olduğu *GPMC* arayüzü, projede mikroişlemcili sistem ile FPGA modülünü haberleştirmede kullanılmakta ve kilit bir görevi üstlenmektedir.



Şekil 6.3 *AM3359* mikroişlemcisi blok şeması [61]

6.1.2 Çapraz Derleme

C, C++, BASIC gibi üst seviye dillerle yazılmış olan kaynak kodların makinenin anlayacağı dile çevrilmesi işlemine derleme denilir. Derleme işlemi kullanılan programlama dili ve mimari için özel olarak yazılmış derleyici programlar ile yapılır. Eğer derleyici program, üzerinde koştuğu platformdan farklı bir platformda koşacak bir program kodu üretiyorsa, bu işlem "çapraz derleme" olarak adlandırılır. Çapraz derleyiciler, hedef platformun derleme işlemi için yeterli performansa ya da belleğe sahip olmaması durumlarında kullanılır. Bir yazılımın, hedef donanıma ihtiyaç duyulmadan, birçok platform için kolaylıkla üretilebilmesini sağlar.

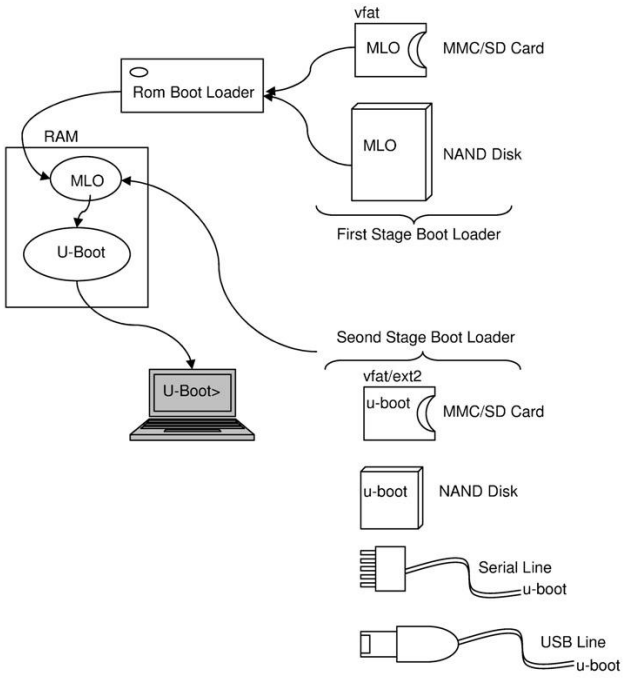
Projede geliştirme ortamı *x86* mimarisine sahip bir bilgisayardır. Dolayısıyla *ARM* mimarili gömülü mikroişlemcili sistem için derleme işlemleri çapraz derleyici ile yapılmıştır. *ARM* mimarisi için açık kaynak *GCC* temelli birçok hazır çapraz derleyici mevcuttur. Bu projede *Texas Instruments* firması tarafından sunulan yazılım geliştirme paketiyle beraber gelen "ARM Arago Linux" çapraz derleyici seti kullanılmıştır.

6.1.3 Önyükleyicinin Derlenmesi

Bir mikroişlemcide işletim sistemi açılmadan önce koşturulan, kullanılan mikroişlemcinin mimarisine göre değişmekle birlikte adres uzayında ilgili alanları değiştirip mikroişlemciyi ve gerekirse çevresel birimleri konfigüre eden kod parçasına "ön yükleyici" denir. Önyükleyiciler işleri bittiğinde işletim sistemini çağırarak, işletim sisteminin belleğe yüklenmesini sağlarlar.

Mikroişlemciler içerisinde gömülü olarak gelen daha basit bir ön yükleyici de bulunmaktadır. Bu önyüklecinin görevi belirli bir adresten birinci seviye önyükleyiciyi (First Stage Boot Loader) yüklemektir [62].

BeagleBone'da birinci seviye önyükleyici *x-loader* veya *MLO* olarak adlandırılır. *MLO* donanımsal önyükleyici tarafından belleğe yüklenir. *MLO* donanımsal önyükleyiciye göre daha yetenekli olup seri kanaldan, *USB*'den, *SD* bellekten veya mikroişlemcinin destek verdiği diğer arayüzler üzerinden başka bir önyükleyici yükleyebilir. *MLO*'nun yüklediği bu önyükleyiciye ikinci seviye önyükleyici (Second Stage Boot Loader) denir.

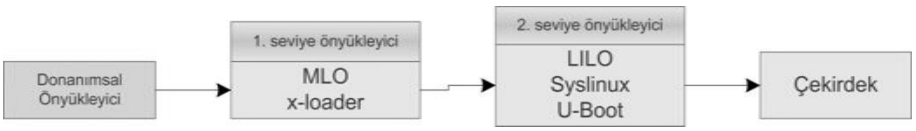


Şekil 6.4 Önyükleyiciler [62]

Linux işletim sistemleri için kullanılan *LILO, Syslinux, U-Boot* gibi birçok ikinci seviye önyükleyici bulunmaktadır. *ARM* mimarisine sahip mikroişlemcili sistemlerde daha çok

U-Boot kullanılmaktadır. Bu çalışmada ikinci seviye önyükleyici olarak *U-Boot* kullanılmıştır.

U-boot geliştirici tarafından konfigüre edilebilen, yüzlerce mikroişlemci ve geliştirme kartı için temel konfigürasyonu üzerinde hazır gelen, betik desteği olan, *GPL* lisansına sahip açık kaynak bir yazılımdır. *U-Boot*'un görevi mikroişlemciyi konfigüre ettikten sonra *Linux* çekirdeğini belleğe yüklemek ve bu *Linux* çekirdeğini ilgili bellek adresinden başlatmaktır. Yürütmeyi çekirdek aldıktan sonra *U-boot*'un görevi son bulur. Yüklenen *Linux* çekirdeği "kök dosya sistem"ini bağlar ve burada açılış betiğini çalıştırır. Açılış betiği mikroişlemcili sistemi hazır hale getirerek kullanıcı arayüz programlarını ya da kullanıcı tarafından istenilen programları çalıştırır [62].



Şekil 6.5 Önyükleyicilerin çalışma sıralaması

6.1.3.1 U-Boot ve MLO'nun Derlenmesi

Öncelikle *DENX* firması web sitesinden *U-Boot* [61] yazılımının en güncel sürümünün kaynak kodu indirilir. Sıkıştırılmış haldeki kaynak kodu geliştirme ortamı üzerinde uygun bir yere açılır. Yeni bir kabuk penceresi (shell window) açılarak kaynak kodunun bulunduğu dizine girilir. Aşağıda verilen betik ile hedef platform *Beaglebone* için *U-boot* ve *MLO* derlenir. *U-Boot* kendisini yükleyecek birinci seviye önyükleyici olan *MLO*'yu kendisi derleyebilmektedir. Burada "O" parametresi ile çıkış dizini, "CROSS_COMPILE" parametresi ile çapraz derleme işlemi yapılacağı belirtilerek derleyici öneki, "ARCH" parametresi ile hedef mimari verilmektedir. Kullanılan son parametre "am335x_evm" ile *BeagleBone* için *U-Boot* içerisinde hazır gelen konfigürasyonun kullanılacağı belirtilmektedir. Derleme işlemi sonrasında çıkış dizininde "*MLO*" ve "*u-boot.img"* isminde iki çalıştırılabilir kod üretilir. Bunlar *BeagleBone* için derlenen sırasıyla birinci ve ikinci seviye önyükleyicilerdir.

```
$ make O=beaglebone CROSS_COMPILE=arm-arago-linux-gnueabi-
ARCH=arm am335x_evm
```

6.1.4 Linux Çekirdeği'nin Derlenmesi

Linux 1990'lı yılların başında *Unix* temel alınarak geliştirilmeye başlanan açık kaynak kodlu, ücretsiz bir işletim sistemi çekirdeğidir. Dünya üzerindeki birçok yazılımcı tarafından geliştirilmeye devam eden *Linux* çekirdeği, açık kaynak kodlu olmasının bir sonucu olarak birçok platformda çalıştırılmıştır. Günümüzde *Linux* çekirdeği resmi olarak; *x86, ARM, Alpha, AVR32, C6X, 68k, H68, PowerPC, SPARC* gibi onlarca mimariye destek vermektedir [63]. *Linux* çekirdeği tek başına bir işletim sistemi değildir. Son kullanıcı tarafından kullanılabilmesi için, "kök dosya sistem"ine ve kullanıcı uygulamalarına ihtiyaç duymaktadır. Son kullanıcılar için *Linux* çekirdeği ve uygun bir dosya sistemi kullanan, buna ek olarak kullanıcı kitleye göre çeşitli kullanıcı yazılımları içeren dağıtımlar oluşturulmuştur. Bu dağıtımlara *Ubuntu, Fedora, Debian, Slackware, Pardus* örnek olarak verilebilir. *Linux* kullanılan dağıtımlar *Linux İşletim Sistemi* olarak tanımlansa da aslında *Linux* bir işletim sistemi değil, işletim sistemi çekirdeğidir.

Bir işletim sistemi çekirdeğinin temel görevleri donanım ile haberleşme, süreç kontrolü, hafıza yönetimi olarak sıralanabilir. Kullanıcı arayüzleri, multimedya ve ofis programları gibi diğer tüm uygulamalar kullanıcı uzayında (user space) çalışan yazılımlardır. *Linux* çekirdeği kullanan tüm işletim sistemleri "tekil hiyerarşik klasör" yapısını kullanır. İşletim sisteminde her şey "/" simgesiyle gösterilen kök klasöründen başlayarak aşağıya inen klasörler ve içerisinde yer alan dosyalardan oluşmaktadır. *Linux* çekirdeği bellekte çalışmaya başladığında önem sırasına göre dosya sistemi içerisinde belirli klasörler içerisinde dosyalar oluşturur. Oluşturulan bu dosyalar *Linux* çekirdeği ile dosya sistemi arasındaki arayüzlerdir. Dosya sisteminde çalışan yazılımlar, donanımsal aygıtlara, arayüzlere, süreç bilgilerine bu arayüzler üzerinden ulaşır. Örnek verilecek olursa *Linux* çekirdeği, üzerinde çalıştığı platform üzerinde bulunan tüm donanımlar için *"/dev*" klasörü altında bir dosya oluşturur. Bu dosyalara okuma veya yazma yapıldığında, aslında ilgili donanıma okuma ve yazma yapılmış olunur.

Linux işletim sistemlerinde dosya sisteminin dizin yapısı dağıtımdan dağıtıma farklılık gösterse de Çizelge 6.2'de verilen klasör yapısı temel alınır. Bu yapı *Linux Vakfı* tarafından 1994 yılında "Linux Dosya Sistemi Hiyerarşisi" (FSSTND) adı altında standartlaştırılmıştır [63].

Çizelge 6.2 *Linux* işletim sistemlerinde temel dosya sistemi hiyerarşisi [64]

Dizin	İçeriği
/bin	Olması zorunlu temel komut dosyalarını içerir (cp, mv, ls gibi). Sistemde bir sorun meydana geldiğinde /bin klasörü altındaki komutlar kullanarak sistem onarılabilmektedir.
/boot	Açılış işlemi sırasında kullanılan dosyaları (çekirdek görüntüsü, sistem haritası, önyükleyici yapılandırması gibi) içerir.
/dev	Bilgisayardaki donanımlarla (sabit diskler, fare gibi) iletişim kurulabilmesi için gereken özel aygıt dosyalarını içerir.
/etc	Sistem ayarlarını barındırır, bulunduğu bilgisayara özel birçok yapılandırma bilgisini içerir. Durağandır ve çalıştırılmak için değildir. Bu dizinde çalıştırılabilir dosyalar bulunmamalıdır.
/home	Bu dizin altında, kullanıcıların kişisel verilerini, yapılandırmalarını kaydettikleri çalışma alanları olan ev dizinleri bulunur. /home dizini altında her kullanıcı için ayrı tahsis edilmiş kullanıcı (ev) dizinleri mevcuttur.(/home/ftpadm , /home/ogrenci gibi).
/lib	Çekirdek modülleri ve paylaşılan kod kütüphanelerini içerir.
/media	Sistem açılışında otomatik olarak bağlanmayan sabit disk bölümleri ile kaldırılabilir aygıtlar (CD-Rom, USB bellek, vb.) bu dizin altında bir dizin açılarak bağlanmaktadır.
/mnt	Sistem açılışında otomatik olarak bağlanan sabit disk bölümleri bu dizin altında bir dizin açılarak bağlanmaktadır.
/opt	Sistem için zorunlu olmayan 3. parti kullanıcı programları bulunur.
/proc	Süreçler, sistem belleği, bağlı aygıtlar, donanım yapılandırmalarıyla ilgili bilgileri içeren özel bir “sanal” dosya sistemidir. Fiziksel dosyalar bulunmaz. Bir bilgi alma merkezi olarak görülebilir, birçok uygulama buradaki bilgilerden yararlanmaktadır.
/root	Sistem yöneticisinin (yani “root” kullanıcısının) ev dizinidir.
/usr	Tüm kullanıcılarca paylaşılan verileri (örneğin programlar, komutlar, kütüphaneler, dokümanlar gibi) içeren dizindir.
/var	Değişken verileri içerir (örneğin rapor dosyaları, veritabanları, kuyrukta bekleyen yazdırılacak dokümanlar gibi)
/tmp	Geçici dosyaları içerir.

Linux çekirdeğinin açık kaynak kodlu olması üzerinde uygulama geliştirmeyi kolaylaştırmaktadır. *Linux* çekirdeğinde birçok ara yüzün, çevresel donanımın sürücü yazılımı (driver) hazır olarak gelmektedir ve gün geçtikçe desteklenen çevresel donanım sayısı artmaktadır. Bunun yanında isteyen bir yazılımcı henüz desteklenmeyen herhangi bir donanımın sürücünü yazıp çekirdeğe yama olarak ekleyebilir ya da bu sürücüyü çekirdek modülü (kernel module) olarak derleyip işletim sistemi çalışırken çekirdeğe tanıtabilir. Açık kaynak kodlu olması nedeniyle *Linux*'da bir donanım için sürücü yazılım oluşturmak *Windows, MacOs* gibi diğer kapalı sistemlere göre daha kolaydır.

Günümüzde *Linux* çekirdeği çapraz derleyiciler ile birçok platform üzerinde çalıştırılabilmektedir. Yine çapraz derleyiciler kullanılarak *C, C++, BASIC* gibi herhangi bir dille yazılmış bir uygulama *Linux*'un çalıştığı her platform için derlenebilir. Bu yazılımcıya yazdığı bir kodu üzerinde değişiklik yapmadan, ya da ufak değişikliklerle birçok platformda çalıştırma özgürlüğü sunar. Yukarıda anlatılan nedenlerden dolayı bu projede gömülü mikroişlemcide işletim sisteminin Linux temelli olması tercih edilmiştir.

Linux çekirdeğinde birçok platformun konfigürasyonu hazır olarak bulunmaktadır. BeagleBone için derlenecek Linux çekirdeğinde BeagleBone için mevcut bulunan konfigürasyonda bazı değişiklikler yapılacaktır. Öncelikle *Linux* çekirdeğinin son sürümünün kaynak kodu [64] indirilerek, sıkıştırılmış haldeki bu kaynak kodu geliştirme ortamı üzerinde uygun bir dizine açılır. Yeni bir kabuk penceresi açılarak kaynak kodunun bulunduğu dizine girilir. Aşağıda verilen betik ile hedef platform olarak *BeagleBone* seçilir. Kaynak kodunun içerisinde derleme betiklerinin bulunduğu "*Makefile"* adı verilen bir dosya bulunur. Bir kabuk penceresi üzerinde *make* programı çağırıldığında, bu program ilgili dizinde *"Makefile"* dosyasını arar ve içinde bulunan betikleri çalıştırır. Aşağıda verilen betikte "CROSS_COMPILE" parametresi ile çapraz derleme işlemi yapılacağı belirtilerek derleyici öneki, "ARCH" parametresi ile hedef mimari verilmektedir. Kullanılan son parametre"am335x_evm" ile *BeagleBone* için çekirdek içerisinde hazır gelen konfigürasyonun kullanılacağı belirtilir. Bu komutla sadece konfigürasyon seçilmiş olur, derleme işlemine başlanmaz.

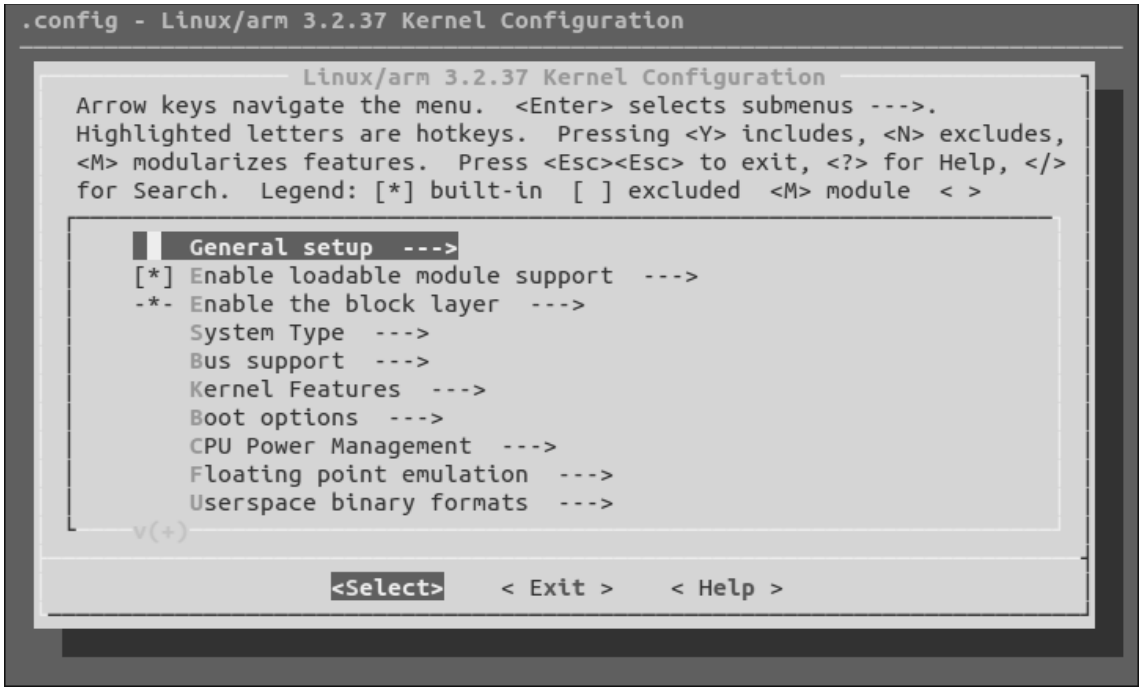
```
$ make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
am335x_evm_ defconfig
```

Şimdi "menuconfig" komutu ile bu konfigürasyonu isteğimize göre düzenleyebiliriz. Aşağıdaki komut işletildiğinde seçilen konfigürasyon baz alınarak *Menuconfig* programı başlatılır.

```
$ make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
menuconfig
```

Menuconfig yazılımının kabuk penceresi üzerinde çalışan Şekil 6.6'da da verilen bir grafik arayüzü bulunmaktadır. Bu arayüz üzerinden yapılan konfigürasyonla istenilen aygıtın sürücüsü çekirdeğe dâhil edilebilir ya da çıkarılabilir. Çekirdek tarafından desteklenecek dosya sistemi tipleri, önyükleme seçenekleri, güvenlik özellikleri, süreç

yönetimi ayarları, güç yönetimi gibi birçok konfigürasyon bu yazılım üzerinden değiştirilebilmektedir. Bu projede gerekli olan “ext3 dosya sistemi” desteği ve “debugfs” özelliği *Menuconfig* programı üzerinden açılacaktır. Çekirdekte “debugfs” özelliği aktif edildiğinde *BeagleBone*’un birçok arayüzüne kullanıcı uzayından erişim hakkına sahip olacağız. Bu özellik geliştirme sırasında yazılım geliştiriciye çevresel donanımlara erişmede kolaylık sunmaktadır.



Şekil 6.6 *Menuconfig*konfigürasyon yazılımı

Bu seçenekler *Menuconfig* yazılımında “Filesystem” ve “Kernel Hacking” sekmeleri altında bulunmaktadır. İlgili seçenekler aktif hale getirildikten sonra *Menuconfig* yazılımından yapılan değişiklikler kaydedilerek çıkılır ve aşağıdaki betik ile derleme işlemi başlatılır.

```
$ make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- uImage
modules
```

Yukarıdaki komutla birlikte çekirdek derlenerek “*uImage”* adında bir dosya ve çekirdek modülleri oluşturulur. Bu dosyaya “çekirdek görüntüsü”(kernel image) adı verilir. Burada çekirdek görüntüsü *U-Boot* önyükleyicisine göre oluşturulmuştur. Oluşturulan bu dosya “arch/arm/boot” klasörü altında bulunmaktadır. Bu çekirdekle çalışacak dosya sistemi oluşturulduktan sonra aşağıda verilen betik çalıştırılarak derlenen çekirdek modüllerinin hedef dosya sisteminde gerekli yerlere kopyalanması sağlanır.

```
$ make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
INSTALL_MOD_PATH=<kök dosya sistemi yeri> modules_install
```

6.1.5 Kök Dosya Sisteminin Hazırlanması

Kök dosya sistemi işletim sistemini çalışması için gerekli olan yazılım ve aygıtların, kullanıcı uygulamalarının bir hiyerarşi içerisinde bulunduğu dizin yapısıdır. Bu yapı içerisinde kök dizin altında bulunan her bir klasör farklı görevleri yerine getiren dosya kümelerini barındırır. Kök dosya sisteminin hiyerarşisi Çizelge 6.2'de verilmektedir.

Basit bir kök dosya sistemi içeriği kullanıcı tarafından ilgili klasörleri tanımlamak suretiyle oluşturulabilir. Ardından gerekli kullanıcı uzayı uygulamalarını derlemek için *Busybox* yazılımı kullanılabilir. *Busybox* yazılımı özellikle sistem kaynaklarının sınırlı olduğu gömülü sistemler için temel kullanıcı uzayı uygulamalarını oluşturmaktadır.

Bu projede sistem kaynakları yeterli olduğundan, yeni bir kök dosya sistemi oluşturmak yerine *BeagleBone* platformu ile birlikte gelen kök dosya sistemi kullanılacaktır.

Beaglebone üzerinde herhangi bir kalıcı bellek bulunmadığından önyükleyiciler, çekirdek görüntüsü ve kök dosya sistemi bir *microSD* belleğe konulacak ve sistem bu bellek üzerinden çalıştırılacaktır. *MicroSD* bellek üzerinde "boot" ve "root" adında iki bölüm oluşturulacak, önyükleyiciler ve çekirdek görüntüsü "boot" bölümü altına, kök dosya sistemi is "root" bölümü altına kopyalanacaktır.

MicroSD bellek üzerinde bölümler oluşturmak için *Fdisk* uygulaması kullanılır. Bu yazılım çalıştırılmadan önce kullanılacak *microSD* belleğin konfigürasyonu temizlenmelidir. Bunun için aşağıda verilen betik yeterlidir. Bu betik *microSD* bellek geliştirme ortamında hangi aygıt adıyla (sda, sdb, sdc vb.) tanınmışsa ona göre düzenlenmelidir.

```
$ sudo dd if=/dev/zero of=/dev/sdc bs=1024 count=10240
```

Fdisk uygulaması ilgili aygıt adıyla açıldıktan sonra iki bölüm oluşturulur. İlk bölüm önyükleme bölümü olacaktır. İkincisine ise kök dosya sistemi oturacaktır. Burada ilk bölüm için 100MB'lık alan ayrılmaktadır diğer bölüm için istenildiği kadar alan tahsis edilebilir.

```
$ sudo fdisk /dev/sdc
Command (m for help): n
```

```
Command (m for help): p
Partition number (1-4, default 1): 1
First sector (15430560-15433727, default 15430560): (enter ile
geçilir)
Last sector, +sectors or +size(K,M,G) (15430560-15433727,
default 15433727): +100M
Command (m for help): n
Command (m for help): p
Partition number (1-4, default 1): 2
First sector (15430560-15433727, default 15430560): (enter ile
geçilir)
Last sector, +sectors or +size(K,M,G) (15430560-15433727,
default 15433727): (enter ile geçilir)
```

İlk bölüm önyüklenebilir olarak ayarlanır. Ve yine ilk bölümün sistem kimliği "*W95 FAT32"* olarak seçilir. Ardından yapılan değişiklikler kaydedilir.

```
Command (m for help): a
Partition number (1-4):  1
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): b
```

MicroSD bellek üzerinde yukarıda anlatıldığı şekilde bölümler oluşturulduktan sonra, oluşturulan ilk bölüm "boot" ismiyle *FAT32* olarak, ikinci bölüm ise "root" ismiyle *ext3* olarak biçimlendirilir. Ardından "boot" bölümüne önyükleyici derlemesi sonucu oluşan "*MLO"* ve "*u-boot.img*" dosyaları ile çekirdek derlemesi sonucu oluşan "*uImage"* çekirdek görüntüsü kopyalanır.

Kullanılacak kök dosya sistemi "*tisdk-rootfs-am335x-evm"* adıyla sıkıştırılmış olarak gelmektedir. Bu dosya oluşturulan "root" bölümüne açılır ve aşağıda verilen betik ile çekirdek derlemesi sonucu oluşan çekirdek modülleri bu bölüme yüklenir. Böylece *BeagleBone* için derlenen *Linux* işletim sistemi *microSD* bellek üzerinde hazır hale gelmiştir. Bu *microSD* bellek *BeagleBone*'a yerleştirilerek sistem çalıştırılabilir.

```
$ make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
INSTALL_MOD_PATH=<kök dosya sistemi yeri> modules_install
```

Beaglebone açıldığında ilk olarak *microSD* belleğin ilk bölümüne bakar ve buradaki "*MLO*"yu belleğe yükleyip çalıştırır. *MLO* çok basit bir uygulama olup görevi "*u-boot. img" dosyasını* belleğe yükleyip çalıştırmaktır. *U-boot* programı üzerinde çalıştığı *AM3359* işlemcisinin çevresel birimlerini ilk çalışma için konfigüre eder ve *Linux* çekirdeği olan "*uImage"* görüntüsünü belleğe yükleyip çalıştırır. *Linux* çekirdeği ilgili konfigürasyonlara göre aygıt sürücülerini yükler, belleği organize eder, kök dosya

sisteminde gerekli arayüzleri oluşturur ve son olarak kök dosya sistemi içerisindeki "*init*" betiğini çalıştırır. Bu betik çalıştırılacak kullanıcı uzayı uygulamalarını sırasıyla çalıştırır.

BeagleBone platformu *USB* arayüzü üzerinden geliştirme ortamına bağlanıp, bir kabuk penceresi içerisinden *Minicom* seri haberleşme programı açıldığında önyükleyici ve çekirdek uyarı mesajlarının aktığı görülebilir. *Linux* çekirdeği işini bitirdiğinde sisteme kullanıcı girişi yapılabilir. Bu pencereden "*root*" kullanıcı adıyla sisteme yönetici haklarıyla giriş yapılır.

6.2 Destek Vektör Makinesi Kütüphanesinin Derlenmesi

Projede sınıflandırma işlemi için destek vektör makinesi kullanılmaktadır. Destek vektör makinesinin *SVM*^{light} *[53], mySVM, LIBSVM, MATLAB SVM Toolbox* gibi birçok yazılımsal gerçeklemesi bulunmaktadır. Proje hazırlık çalışmalarında yapılan testlerde *SVM*^{light} kütüphanesi için çapraz derleme işleminin daha zahmetsiz olması ve alınan performansın tatmin edici olması nedeniyle bu kütüphane tercih edilmiştir.

SVM^{light} kütüphanesi *C* diliyle yazılmış bir destek vektör makinesi gerçeklemesidir. Örüntü tanıma, regresyon, öğrenme algoritması problemleri için Vapnik'in destek vektör makinasını [39] temel alır. *SVM*^{light} kütüphanesinde kullanılan, Joachims tarafından önerilen [65], [66] optimizasyon algoritması ayarlanabilir hafıza kullanımına sahiptir ve binlerce destek vektörü kullanımı durumunda dahi hafızanın verimli kullanımı sağlar. Hata oranı ve duyarlılık (sensitivity) için XiAlpha [65], [67] yaklaşım metodu kullanılmaktadır.

SVM^{light} kütüphanesinin üç yeni türü bulunmaktadır. Bunlar;

- *SVM*^{struct}: Bu gerçeklemede kullanılan öğrenme algoritması çok değişkenli ve yapılı çıktıya izin verir.
- *SVM*^{perf} : Doğrusal destek vektör makinesi kullanımında büyük veri öbekleri için *SVM*^{light}'dan çok daha hızlı öğrenme algoritmasına sahiptir.
- *SVM*^{rank}: Sıralama düzeni (ranking) temelli destek vektör makinesi kullanımında *SVM*^{light}'dan çok daha hızlı öğrenme algoritmasına sahiptir

Projede *SVM*^{light} ve *SVM*^{perf} kütüphanelerinin ikisi de derlenmiştir. Derleme işlemi iki kütüphane için de benzer şekildedir. Öncelikle *SVM*^{light} kütüphanesinin son sürümünün

kaynak kodu indirilerek, sıkıştırılmış haldeki bu kaynak kodu geliştirme ortamı üzerinde uygun bir dizine açılır. Yeni bir kabuk penceresi açılarak kaynak kodunun bulunduğu dizine girilir.“Make” komutu ile derleme başlatılmadan “Makefile” dosyasının ihtiyaç duyacağı çevresel değişkenler aşağıdaki gibi ayarlanır. Bu değişkenler içerisinde çapraz derleyici öneki, çapraz derleme yapılacak hedef mimari için oluşturulmuş paylaşılan kütüphanelerin yolu, derlemede gerekli bazı bayraklar yer alır. Ardından “Makefile” dosyası açılarak *"CC=gcc”* kodunun bulunduğu satır yorum satırı haline getirilir. Böylece “Makefile” geliştirme ortamında bulunan *x86* derleyicisi yerine çapraz derleyiciyi kullanacaktır. “Make” komutu ile derleme işlemi başlatılır. Derleme işlemi sonucu “*svm_classify”* ve “*svm_learn”* adında iki program üretilir. Bunlardan “*svm_learn”* destek vektör makinesi ile öğrenmede, “*svm_classify”* programı ise sınıflandırmada kullanılacaktır.

```
export PATH=$PATH:/opt/ti-sdk-am335x-evm/linux-devkit/bin/
export ARM_TARGET_LIB=/opt/ti-sdk-am335x-evm/linux-devkit/arm-
arago-linux-gnueabi/usr/lib
export TOOL_PREFIX=/opt/ti-sdk-am335x-evm/linux-devkit/bin/arm-
arago-linux-gnueabi
export CXX=$TOOL_PREFIX-g++
export AR=$TOOL_PREFIX-ar
export RANLIB=$TOOL_PREFIX-ranlib
export CC=$TOOL_PREFIX-gcc
export LD=$TOOL_PREFIX-ld
export CCFLAGS="-march=armv7-a -mtune=cortex-a8 -mfpu=vfp -
I/opt/ti-sdk-am335x-evm/linux-devkit/arm-arago-linux-
gnueabi/usr/lib"
```

6.3 Destek Vektör Makinesi Kütüphanesi ile Sınıflandırma Uygulaması

SVM^{light} kütüphanesinin BeagleBone platformu için çapraz derlenmesi sonucu üretilen “*svm_learn”* ve “*svm_classify”* programları, *BeagleBone* üzerinde bulunan dosya sistemine kopyalanacak ve örnek bir sınıflandırma uygulaması yapılacaktır. *BeagleBone* platformu ile geliştirme ortamı arasında hızlı dosya paylaşımı yapmak üzere *ethernet* arayüzü kullanılabilir. Bunun için bir *ethernet* kablosu ile geliştirme ortamının bulunduğu bilgisayar ile *BeagleBone* birbirine bağlanır. Birbirine bağlı iki sistemde de otomatik *IP* adresi verecek bir *DHCP* sunucusu bulunmadığından, iki sisteme de el ile *IP* ataması yapılır. *Linux’da* ağ üzerinden hızlı dosya paylaşımı için *SCP* programı

kullanılabilmektedir. "*svm_learn"* ve "*svm_classify*" programlarının bulunduğu dizinde aşağıda betik çalıştırılarak bu programlar *BeagleBone* üzerinde bulunan dosya sistemine gönderilir.

```
$ scp svm_learn root@<BeagleBone IP'si>:/home/root/.
$ scp svm_classify root@<BeagleBone IP'si>:/home/root/.
```

SVM^{light} kütüphanesi ile sınıflandırma uygulamasında ilk olarak eğitim veri öbeği kullanılarak "*svm_learn"* programı ile sistem eğitilecek, ardından test veri öbeği kullanılarak "*svm_classify"* programı ile eğitilmiş sistem test edilecektir. "*svm_learn"* programı eğitim sonucu olarak bir sınıflandırma model dosyası oluşturur. Bu model dosyası test aşamasında "*svm_classify"* programına girdi olarak verilir. "*svm_learn"* programı aşağıdaki parametreleri alır.

```
$svm_learn [seçenekler] <eğitim_verisi_dosyası><model_dosyası>
```

Örnek sınıflandırma uygulaması olarak metin sınıflandırma yapılacaktır. Bu sınıflandırma ile Reuters [68] makaleleri iki kategoriye ayrılacaktır. Bunun için toplam 9947 öznitelik oluşturulmuştur. Öncelikle sınıflandırmada kullanılacak veri öbeği [69] indirilerek *BeagleBone* dosya sistemine kopyalanır. Ardından sıkıştırılmış haldeki bu dosya *SVM*^{light} programlarının bulunduğu dizine açılır. Açılan dosya içerisinde eğitim veri öbeğini içeren "*train.dat"* dosyası ile test veri öbeğini içeren "*test.dat"* dosyaları bulunur. Eğitim veri öbeği içerisinde 1000 pozitif ve 1000 negatif örnek bulunmaktadır. Test veri öbeği içerisinde ise 600 örnek bulunur. Önce eğitim veri öbeği dosyası kullanılarak "*svm_learn"* programı ile eğitim yapılacaktır. Bunun için *BeagleBone* platformunda aşağıda verilen betik çalıştırılır.

```
$svm_learn example1/train.dat example1/model
```

Bu betik çalıştırıldığında alınan ekran çıktısı Şekil 6.7'de verilmektedir.

```
root@hogdev_linux:~/svm_test# ./svm_learn example1/train.dat example1/model
Scanning examples...done
Reading examples into memory...100..200..300..400..500..600..700..800..900..100)
Setting default regularization parameter C=1.0000
Optimizing.....................................................................)
Optimization finished (5 misclassified, maxdiff=0.00085).
Runtime in cpu-seconds: 3.04
Number of SV: 878 (including 117 at upper bound)
L1 loss: loss=35.67674
Norm of weight vector: |w|=19.55576
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=383.42791
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.01
XiAlpha-estimate of the error: error<=5.85% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>95.40% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>93.07% (rho=1.00,depth=0)
Number of kernel evaluations: 45954
Writing model file...done
root@hogdev_linux:~/svm_test#
```

Şekil 6.7 *SVM*^{light} kütüphanesi ile eğitim işlemi sonucu alınan ekran çıktısı

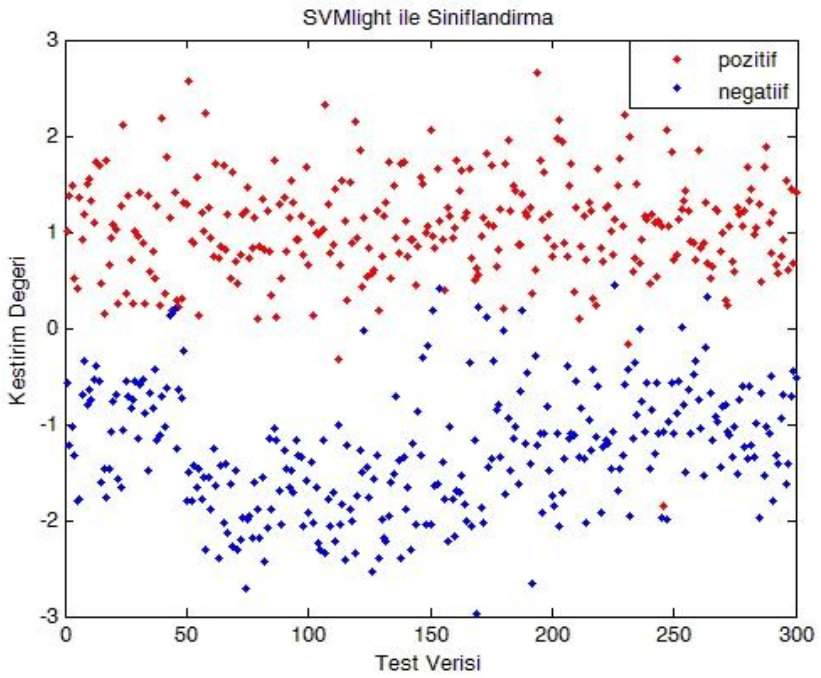
Eğitim sonucu oluşan model dosyası ile test veri öbeği için sınıflandırma işlemi "*svm_classify"* programı ile yapılacaktır. Bunun için *BeagleBone* platformunda aşağıda verilen betik çalıştırılır. Bu betik çalıştırıldığında alınan ekran çıktısı Şekil 6.8'de verilmektedir. *"svm_classify"* programı çıktı olarak "*prediction*" adında bir dosya oluşturur. Bu dosya içerisinde örneklerin kestirim değerleri bulunur.

```
$ svm_classify example1/test.dat example1/model
example1/prediction
```

```
root@hogdev_linux:~/svm_test# ./svm_classify example1/test.dat example1/model ex
ample1/prediction
Reading model...OK. (878 support vectors read)
Classifying test examples..100..200..300..400..500..600..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 97.67% (586 correct, 14 incorrect, 600 total)
Precision/recall on test set: 96.43%/99.00%
root@hogdev_linux:~/svm_test#
```

Şekil 6.8 SVM^{light} kütüphanesi ile test işlemi sonucu alınan ekran çıktısı

Sınıflandırma için eşik seviye 0 olarak kabul edildiğinde alınan başarım Şekil 6.8'de görülmektedir. 600 örnekten 586'sı doğru tahmin edilmiştir. Başarı yüzdesi %97,67 olarak ölçülmüştür. Test sonucu oluşturulan prediction dosyasını *MATLAB* programındaki "plot" fonksiyonu ile çizdirdiğimizde Şekil 6.9'daki çıktıyı alırız.



Şekil 6.9 SVM[light] kütüphanesi ile yapılan sınıflandırma sonucu kestirim değerleri

BÖLÜM 7

YÖNLÜ GRADYANLARIN HİSTOGRAMI ALGORİTMASI VE DESTEK VEKTÖR MAKİNESİ İLEYAYA TANIMA UYGULAMASI

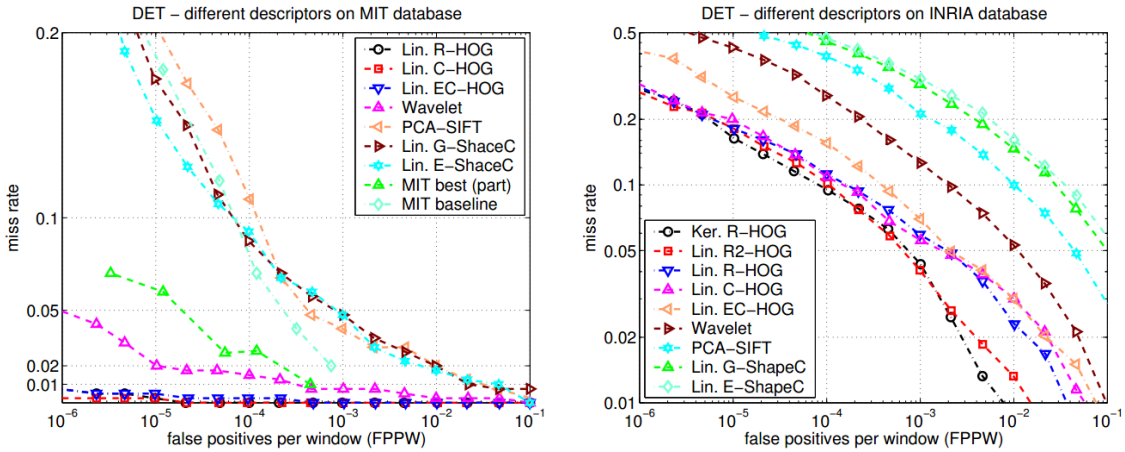
Bu bölümde FPGA üzerinden çalışan Yönlü Gradyanları Histogramı (HOG) algoritması ve mikroişlemci üzerinde çalışan Destek Vektör Makinesi (SVM) ile bir yaya tanıma uygulaması yapılacaktır. Bunun için FPGA ve mikroişlemcili melez bir sistem oluşturulmuştur. İlk olarak yaya tanıma uygulamaları hakkında bilgi verilecektir. Ardından tasarlanan sistemin bileşenleri anlatılacaktır. Son olarak sistem çalıştırılacak ve başarımı ölçülecektir.

7.1 Yaya Tanıma Uygulaması

Yaya tanıma günümüzde en ilgi çekici bilgisayarla görü (computer vision) alanlarından biridir. Araç kamerası kullanımıyla trafik kazalarını önleme ya da güvenlik kamerası kullanarak suç önleme konularına yönelik olarak birçok tanıma algoritması bulunmaktadır. Yaya tanıma, yaya tespiti ve takibi olmak üzere iki önemli işlemi içermektedir. Bu iki işlemde de tanıma amacıyla öznitelik tanımlayıcıları kullanılmaktadır.

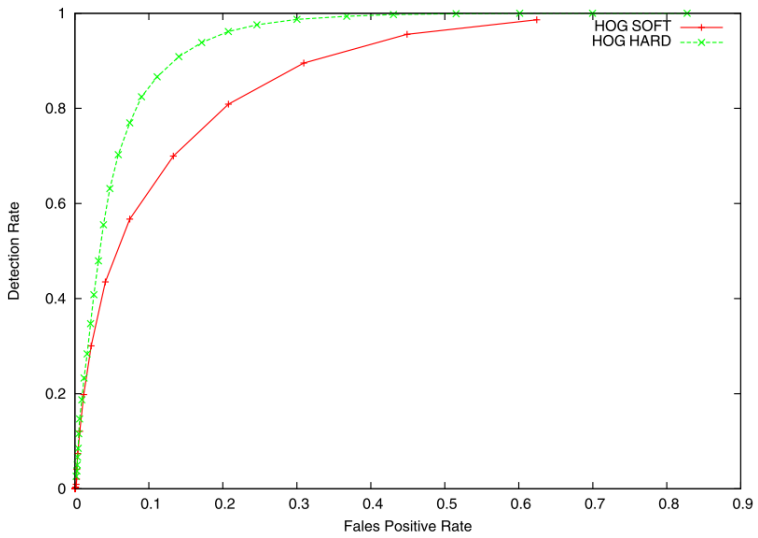
Yaya tanıma uygulamalarında kullanılmak üzere çeşitli öznitelik çıkarıcı metodlar önerilmektedir. Bunlara Gabor Filtreleri [68], Haar Dalgacıkları [70], Haar-benzeri öznitelikler [71] ve daha çok yaya tanımada kullanılan HOG [6] metodu örnek olarak gösterilebilir. Bu çalışmada öznitelik çıkarmada HOG metodu kullanıldığından, burada daha önce HOG metodu kullanılarak yapılmış yaya tanıma uygulamalarından bahsedilecektir.

Yaya tanıma amacıyla HOG metodunun kullanılmasını ilk olarak Dalal ve Triggs [6] tarafından önerilmiştir. Burada HOG metodu ile çıkarılan öznitelikler ile doğrusal SVM kullanılarak sınıflandırma işlemi yapılmıştır. Sistem MIT ve INRIA yaya veritabanı ile test edilmiş ve performansı diğer metotlar ile karşılaştırılmıştır. (Şekil 7.1)



Şekil 7.1 MIT ve INRIA yaya veritabanı üzerinde çeşitli metotların performansı [6]

Kadota vd. [13] HOG algoritmasının donanımsal gerçekleyen bir mimari önermiştir. Negi vd. [14] önerilen mimariyi geliştirerek FPGA üzerinde gerçeklemiştir. Çıkarılan öznitelikleri sınıflandırmada Adaboost temelli bir zayıf sınıflandırıcı (weak classifier) kullanılmaktadır. NICTA yaya veritabanı ile test edilen sistemin başarımını gösteren ROC eğrisi verilmektedir.



Şekil 7.2 NICTA yaya veritabanı için Negi ve diğerleri tarafından oluşturulan sistemin ROC eğrisi [14]

7.2 Sistemin Bileşenleri

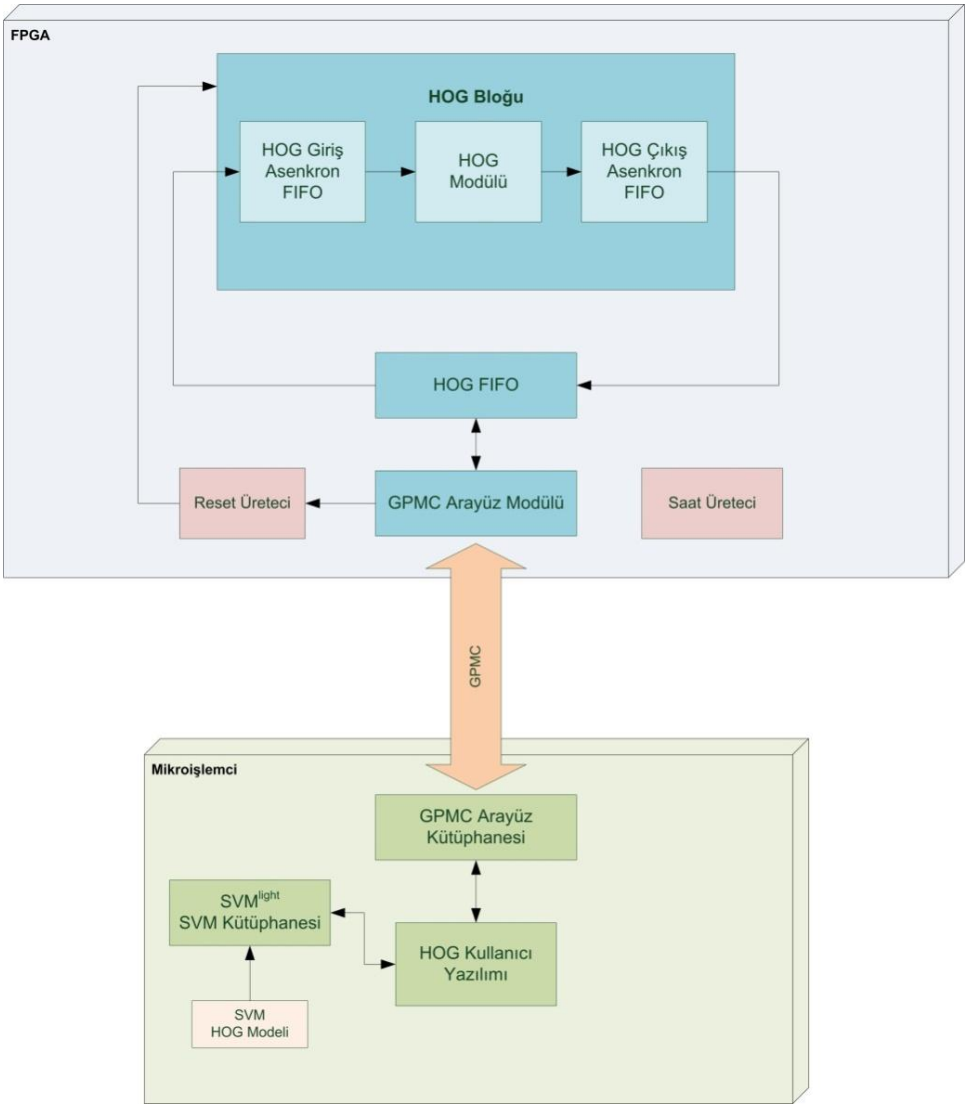
Tasarlanan yaya tanıma uygulaması öznitelik çıkarma ve sınıflandırma bölümlerinden oluşmaktadır. Öznitelik çıkarma metodu olarak HOG algoritması, sınıflandırıcı olarak ise SVM kullanılmaktadır. HOG algoritması donanımsal olarak gerçeklenmiştir ve FPGA üzerinde çalışmaktadır. HOG algoritmasının FPGA üzerinde donanımsal olarak gerçeklenmesi Bölüm 5'te anlatılmıştır. Sınıflandırıcı olarak ise yazılımsal bir SVM gerçeklemesi olan *SVM*^{light} kütüphanesi kullanılmaktadır ve bu yazılım bir gömülü mikroişlemci üzerinde koşmaktadır. Bölüm 6'da *SVM*^{light} ile bir gömülü mikroişlemci üzerinde sınıflandırma uygulaması gerçekleştirilmiştir. Oluşturulan bu sistemde, önceki bölümlerde anlatılan bu iki gerçekleme beraber kullanılarak bir yaya tanıma uygulaması yapılmaktadır.

Sistemin donanımsal altyapısını *Altera DE0 Nano* [52] uygulama geliştirme kartı ve *BeagleBone* [56] uygulama geliştirme kartı oluşturmaktadır. Bu iki kartın haberleştirmek üzere basit bir arayüz kartı tasarlanmıştır. Tasarlanan sistemin blok yapısı Şekil 7.3'de verilmektedir.

7.2.1 FPGA Üzerinde Bulunan Bileşenler

Sistemde öznitelik çıkaran HOG algoritması FPGA üzerinde gerçeklenmiştir. Bu gerçekleme Bölüm 5'te anlatılmaktadır. FPGA üzerinde;

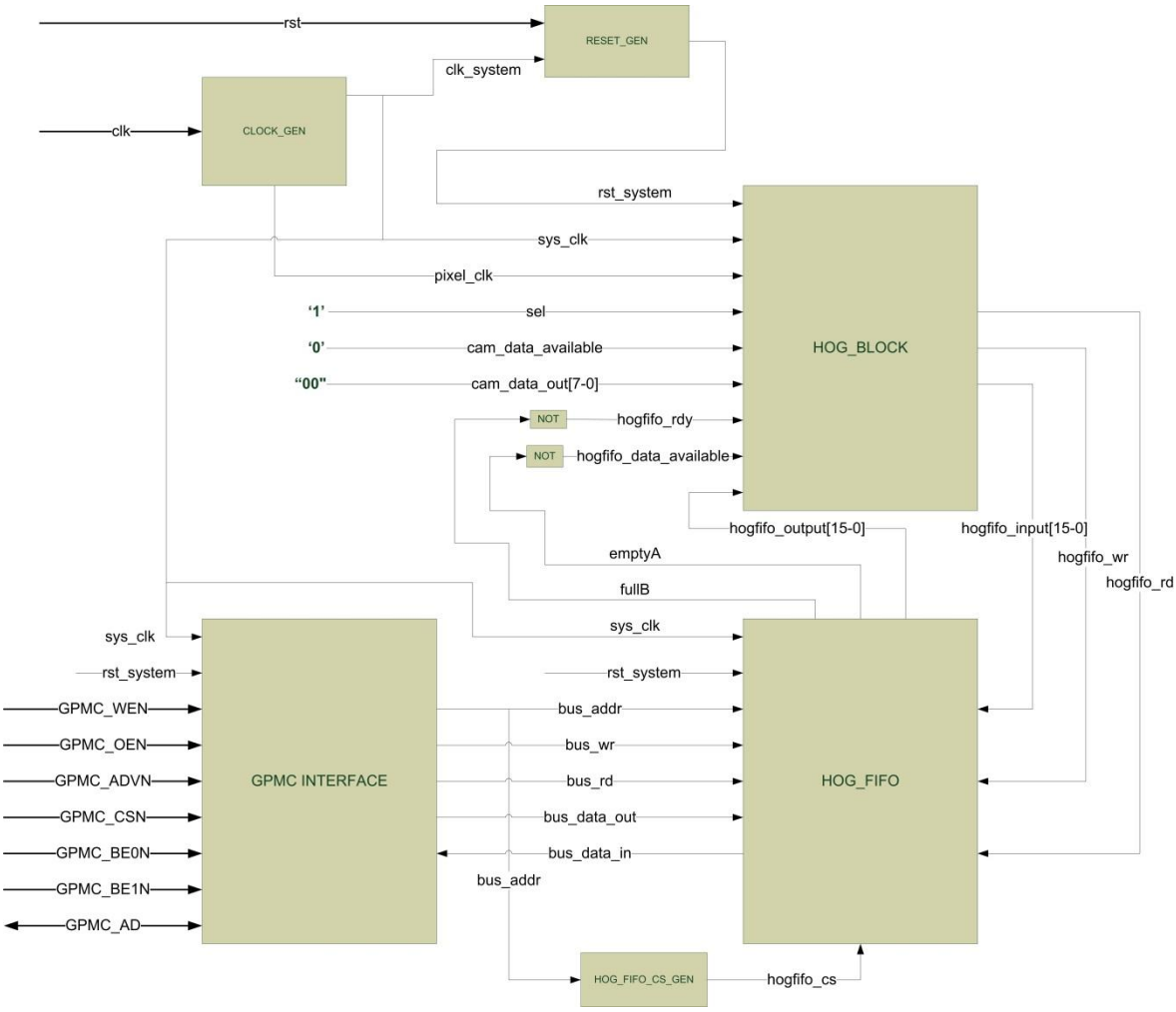
- HOG Modülü, Asenkron HOG Giriş FIFO'su ve Asenkron HOG Çıkış FIFO'sundan oluşan "HOG Bloğu",
- FPGA ve mikroişlemci arasında haberleşmeyi sağlayan "GMPC Arayüz Modülü",
- GMPC Arayüz Modülü ve HOG Bloğu arasında yer alan "HOG FIFO Modülü",
- Saat üreteci ve reset üreteci bulunmaktadır.



Şekil 7.3 FGPA – Mikroişlemci temelli melez yaya tanıma sistemi blok yapısı

Mikroişlemci, GPMC Arayüz Modülü'nde HOG FIFO Modülü için belirlenen adrese özniteliği çıkarılacak görüntü verisini yazar ve çıkarılan öznitelikleri bu adresten okur. Mikroişlemci reset üretici ile de bu GPMC Arayüzü Modülü üzerinden haberleşir ve sistemi resetleyebilir. Saat üretici sistemin ihtiyaç duyduğu sistem saati ve piksel saatini üretir. Tasarımın üst seviye blok yapısı Şekil 7.4'de verilmektedir. Tasarım gerçek zamanlı yaya tanıma uygulamasına yönelik olarak yapılmıştır. HOG Bloğu bir kamera üzerinden gönderilen görüntüyü gerçek zamanlı olarak işleyebilir. Hazırlanan sistemde gerçek zamanlı çalışma modu için gerekli altyapı bulunmakla beraber, bu mod şu anda pasif olarak bırakılmıştır. Mevcut sistem mikroişlemci tarafından gönderilen görüntüyü

işleyip çıkarılan öznitelikleri yine mikroişlemciye gönderen mod olan, eğitim ve test modunda çalışmaktadır.



Şekil 7.4 FPGA tasarımı üst seviye blok yapısı

7.2.1.1 Saat Üreteci

Saat Üreteci Modülü sistemin ihtiyaç duyduğu sistem saati ve piksel saatini üretir. Sistemin giriş saati FPGA modülü üzerinde bulunan osilatör tarafından üretilir. Giriş saatinin frekansı 50 MHz'dir. Saat Üreteci Modülü FPGA üzerinde yer alan faz kilitlemeli döngü frekans sentezleyicisini (Phase-Locked Loop Frequency Synthesizer– PLL) kullanarak 25 MHz'lik "piksel saati" ve 120 MHz'lik "sistem saat"ini üretir.

HOG Bloğu dışındaki tüm bloklar yalnızca sistem saatine ihtiyaç duyar. HOG Bloğu içerisinde bulunan HOG Modülü'nün bazı bileşenleri piksel saati ile çalışır. Bu yüzden HOG Modülü giriş ve çıkışına çift saat ile çalışan asenkron FIFO'lar bağlanmıştır

7.2.1.2 Reset Üreteci

Reset Üreteci Modülü sistemdeki blokların ihtiyaç duyduğu reset sinyalini üretmektedir. Sistem üç durumda resetlenebilir. İlk olarak sistem açıldığında reset üreteci tüm sistemi belirli bir süre resette tutar. Reset Üreteci Modülü'ne dışarıdan bir reset düğmesi girişi bulunur. Reset düğmesine basıldığında Reset Üreteci yine tüm sistemi belirli bir süre resette tutar. Son olarak Reset Üreteci'ne mikroişlemci GPMC arayüzü üzerinden reset sinyali gönderebilir. Bu durumda Reset Üreteci sadece HOG Bloğu için reset sinyali üretir. Üretilen bu resetin uzunluğu mikroişlemci tarafından belirlenir.

7.2.1.3 GPMC Arayüz Modülü

Genel Amaçlı Hafıza Kontrol (Global Purpose Memory Controller – GPMC) arayüzü *Texas Instruments* firmasının mikroişlemcilerinde bulunan, tüm standart hafızalar ile haberleşmede kullanılabilen esnek bir hafıza kontrol arayüzüdür [72]. GPMC arayüzü;

- Asenkron okuma/yazma erişimi,
- Asenkron sayfa (page) okuma erişimi (4, 8, 16 word),
- Senkron okuma/yazma erişimi,
- Senkron ivedi (burst) okuma/yazma erişimi (4, 8, 16 word),
- Adres/veri çoğullamalı (multiplexed) erişim,
- Düşük ve yüksek öncelikli erişim türlerini desteklemektedir.

GMPC arayüzü ile birçok harici aygıt ile haberleşilebilir. Bunlar;

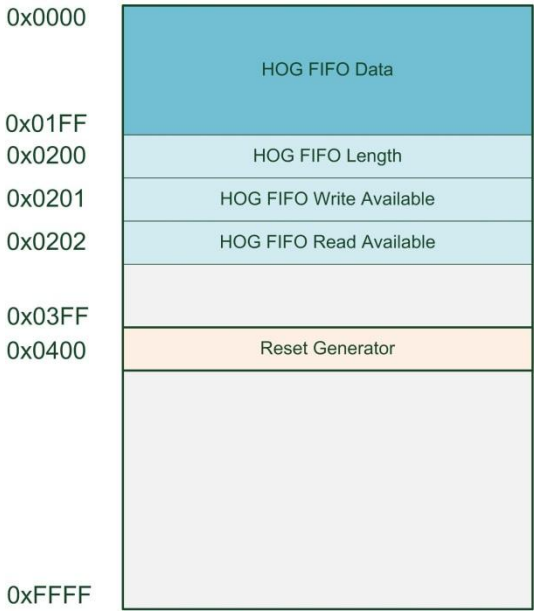
- Harici senkron veya asenkron 8-bit/16-bit genişlikli hafıza veya aygıt,
- Harici 16-bit sınırlı adres genişliğinde çoğullamasız aygıtlar (2KB),
- Harici 16-bit adres/veri çoğullamalı NOR flash hafızalar,
- Harici 8-bit/16-bit NAND flash hafızalar,
- Harici 16-bit psödo SRAM hafızalardır.

Tasarımda FPGA mikroişlemci haberleşmesinde kullanılan GPMC arayüzü, 16-bit adres/veri çoğullamalı NOR flash hafıza modunda çalışmaktadır. GPMC arayüzü maksimum 100 MHz hızında çalışmaktadır. GPMC arayüzü ile pratikte maksimum 51 MB yazma, 40 MB okuma hızına ulaşılmıştır [72].

FPGA tasarımında kullanılan GPMC Arayüzü Modülü, *Logi-Bone* [73] projesinde kullanılan GPMC arayüzü temel alınarak oluşturulmuştur. Sistemde kullanılan GPMC

Arayüz Modülü 16-bit adres uzayına sahiptir. Dolayısıyla 65536 word genişliğinde bir alanı adresleyebilmektedir. Tasarlanan sistemde GPMC adres uzayında (0x0000 – 0x03FF) adresleri arasındaki alan HOG FIFO Modülü'ne, 0x0400 adresi ise Reset Üreteci'ne tahsis edilmiştir. Sistemin GPMC arayüzü adres uzayı Şekil 7.5'de verilmektedir.

Reset Üreteci adresinde varsayılan olarak sıfır değeri bulunur. Mikroişlemci buraya resette kalınacak çevrim sayısı değerini yazdığında, HOG Modülü bu değer kadar resette tutulur. HOG FIFO Modülü adres alanında (0x0000 – 0x01FF) adresleri arası 512 word boyunda veri okuma ve yazma alanıdır. HOG FIFO Modülü'ne ivedi olarak "*memcpy", "memset"* gibi fonksiyonlarla bir seferde maksimum 1024 bayt boyutunda veri yazılabilir ve okunabilir. HOG FIFO Modülü adres alanında (0x0200 – 0x03FF) adresleri arası kontrol yazmaçlarına tahsis edilmiştir.



Şekil 7.5 Sistemin GPMC arayüzü adres uzayı

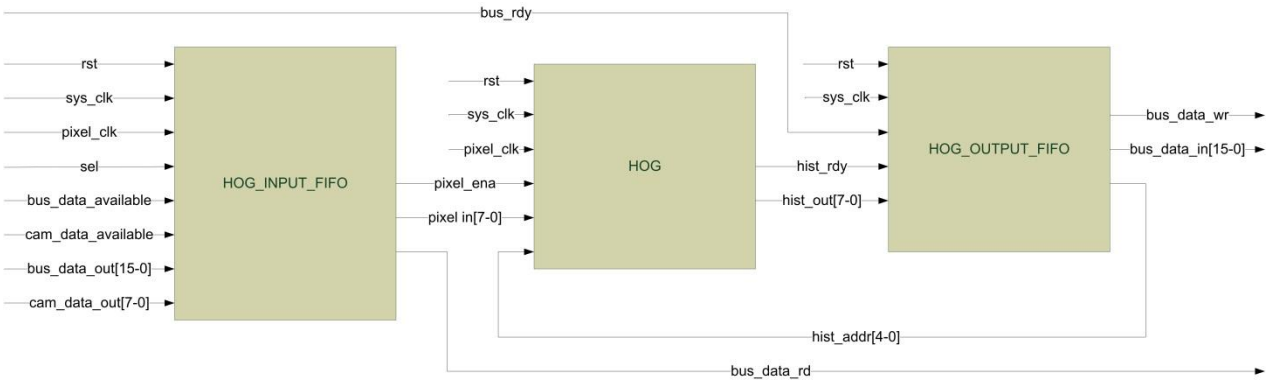
7.2.1.4 HOG FIFO Modülü

HOG FIFO Modülü içerisinde HOG Bloğu'na yazılacak veriler için bir yazma FIFO'su, HOG Bloğu'ndan okunacak veriler için de bir okuma FIFO'su bulunur. Sistemin GPMC adres uzayında HOG FIFO Modülü veri alanına yazılan veri yazma FIFO'suna aktarılır. Bu adreslerden yapılan okuma işleminde ise veri okuma FIFO'sundan alınmaktadır.

HOG FIFO Modülü adres alanında 0x0200 adresinden FIFO boyutu olan 1024 değeri okunur. 0x0201 adresinden HOG FIFO Modülü yazma FIFO'sundaki mevcut veri boyutu değeri, 0x0202 adresinden HOG FIFO Modülü okuma FIFO'sundaki mevcut veri boyutu değeri okunabilir. 0x0201 ve 0x0202 adreslerine sıfır değeri yazılarak yazma ve okuma FIFO'ları boşaltılabilir. Mikroişlemci HOG FIFO Modülü'nden ne kadar veri okuyacağına ya da taşma olmadan ne kadar veri yazabileceğine bu adreslerdeki değerleri okuyarak karar verir.

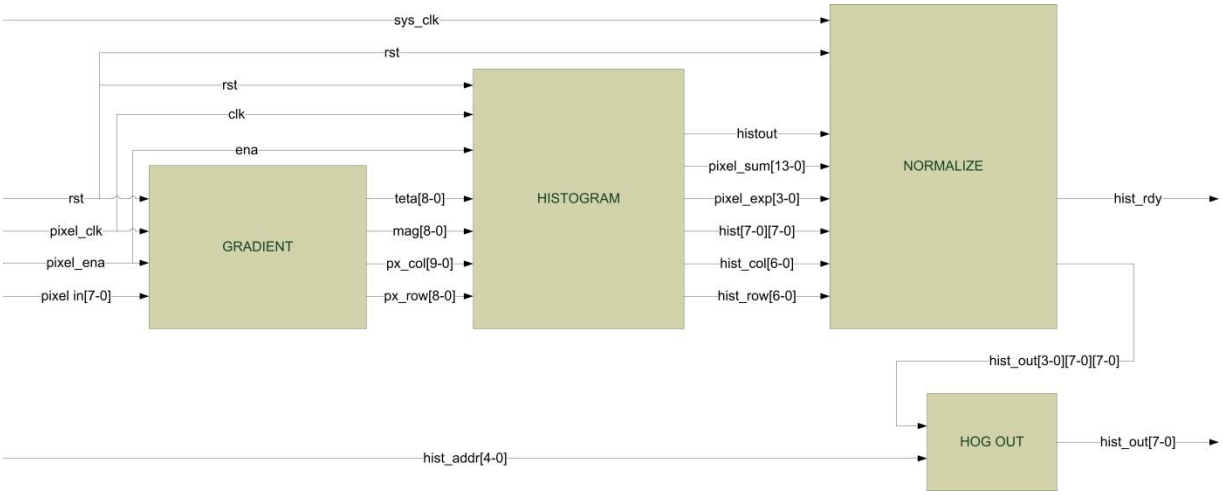
7.2.1.5 HOG Bloğu

Tasarlanan HOG Bloğu içerisinde, Asenkron HOG Giriş FIFO'su, Asenkron HOG Çıkış FIFO'su ve HOG Modülü bulunmaktadır. Tasarlanan HOG Bloğu blok yapısı Şekil 7.6'te verilmektedir. Blok içerisinde veri akış yönü Asenkron HOG Giriş FIFO'su, HOG Modülü ve Asenkron HOG Çıkış FIFO'su şeklindedir. Asenkron HOG Giriş FIFO'su HOG FIFO Modülü yazma FIFO'sunda bir veri olduğunda buradan okuma yapar ve HOG Modülü'ne gönderir. Asenkron HOG Çıkış FIFO'su ise HOG Modülü'nde veri hazır olduğunda bu veriyi okuyarak HOG FIFO Modülü okuma FIFO'suna göndermektedir.



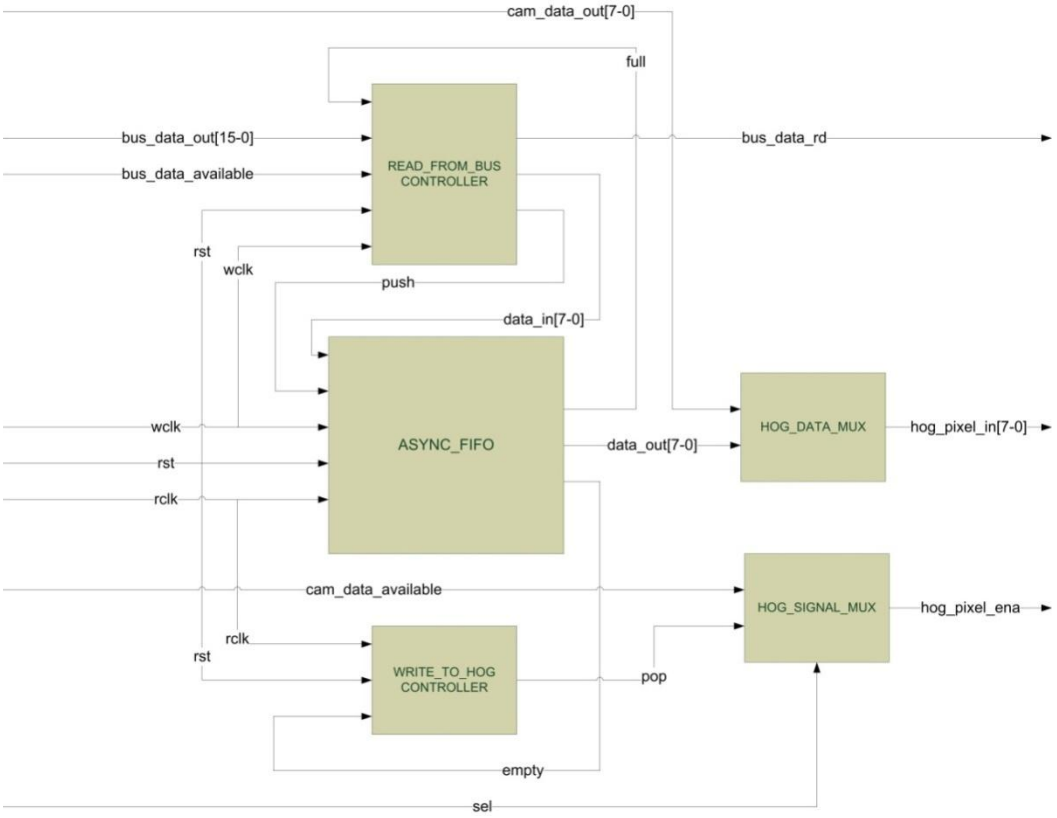
Şekil 7.6 Tasarlanan HOG Bloğu'nun blok yapısı

HOG Modülü'nün tasarımı Bölüm5'te anlatılmaktadır. Burada HOG Modülü Şekil 5.4'te verilen tasarıma küçük bir ekleme yapılarak kullanılmaktadır. Kullanılan HOG Modülü blok yapısı Şekil 7.7'de verilmektedir.

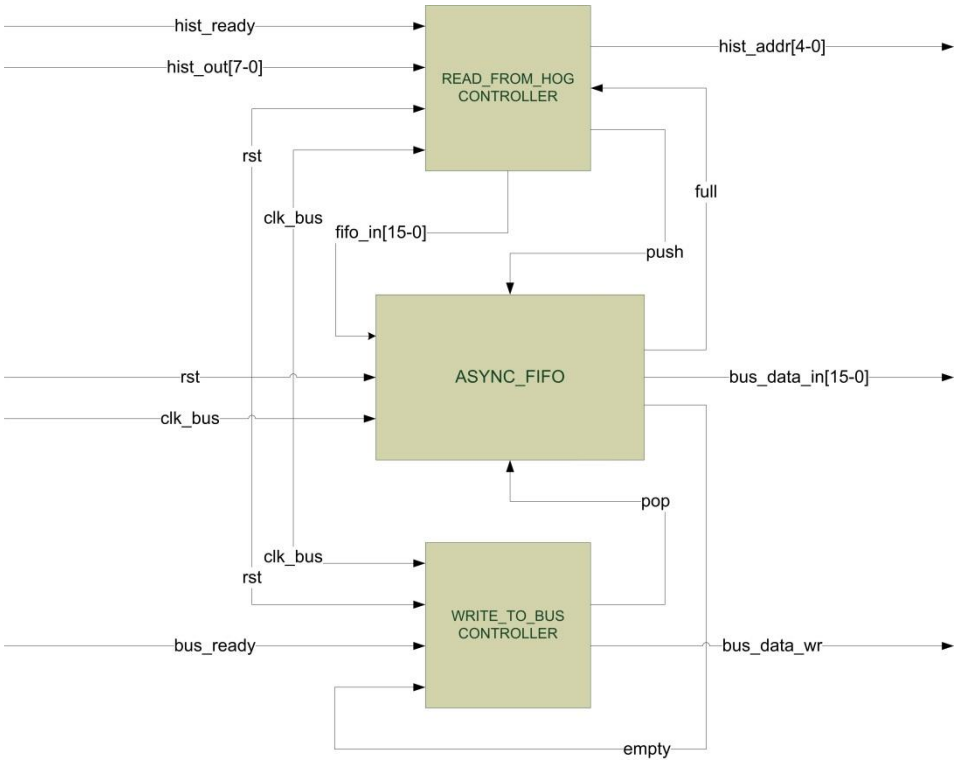


Şekil 7.7 Sistemde kullanılan HOG Modülü blok yapısı

Asenkron HOG Giriş FIFO'su çift saat işareti ile çalışır. Bunlar sistem saati ve piksel saatidir. HOG FIFO Modülü'nden sistem saati ile 16 bit genişliğinde aldığı veriyi, HOG Modülü'ne piksel saati ile 8 bit genişliğinde iki parça olarak gönderir. Asenkron HOG Çıkış FIFO'su çift saat girişine sahiptir. Fakat burada senkron olarak çalışmaktadır. Bunun nedeni HOG Modülü'nde çıkış işaretlerinin de sistem saati ile olmasıdır. Asenkron HOG Çıkış FIFO'su HOG Modülü'nden 8 bit genişliğinde aldığı verileri ikişer birleştirerek, HOG FIFO Modülü'ne 16 bit genişliğinde gönderir. Asenkron HOG Giriş ve Asenkron HOG Çıkış FIFO'ları blok yapıları Şekil 7.8 ve Şekil 7.9'de verilmektedir.



Şekil 7.8 HOG bloğu asenkron HOG giriş FIFO'sunun blok yapısı



Şekil 7.9 HOG bloğu asenkron HOG çıkış FIFO'sunun blok yapısı

7.2.2 Mikroişlemci Üzerinde Bulunan Bileşenler

Sistemde sınıflandırmada kullanılan Destek Vektör Makinesi (SVM) mikroişlemci üzerinde gerçeklenmiştir. Bu gerçekleme Bölüm 6'da anlatılmaktadır. Mikroişlemci üzerinde;

- GPMC arayüzü kütüphanesi ve konfigürasyon yazılımı,
- *SVM*^{light} kütüphanesi,
- HOG kullanıcı yazılımları bulunur.

Mikroişlemci, özniteliği çıkarılacak görüntüyü GPMC arayüzü üzerinden FPGA üzerinde çalışan HOG Bloğu'na gönderir ve yine buradan çıkarılan öznitelikleri okur. Bu öznitelikler *SVM*^{light} kütüphanesi ile öğrenme ve sınıflandırmada kullanılır.

7.2.2.1 GPMC Arayüzü Kütüphanesi ve Konfigürasyon Yazılımı

Mikroişlemci ile FPGA arasında haberleşmede GPMC arayüzü kullanılmaktadır. Bu arayüzün mikroişlemci içerisinde koşan kullanıcı uygulamaları tarafından kullanılabilmesi için kullanıcı uzayında çalışan bir kütüphane oluşturulmuştur. Bunun dışında mikroişlemci üzerinde bulunan GPMC modülünü konfigüre eden bir konfigürasyon yazılımı hazırlanmış, mikroişlemcinin giriş – çıkış arayüzlerini GPMC haberleşmesine göre konfigüre eden bir kabuk betiği yazılmıştır.

Sistemde kullanılan mikroişlemci olan *Texas Instruments AM3359* mikroişlemcisinin giriş – çıkış arayüzlerleri birçok arayüzü destekleyecek şekilde tasarlanmıştır. Bunun için mikroişlemci üzerinde bir çoğullayıcı (multiplexer) yapısı bulunur. Mikroişlemci üzerindeki çoğullayıcıyı "*addr_mux.sh"* kabuk betiği, GPMC arayüzü için konfigüre etmektedir.

Mikroişlemci üzerinde GPMC arayüzü kullanılmadan önce *Texas Instruments AM3359* mikroişlemcisinin GPMC modülü konfigüre edilmelidir. Bunun için "*gpmc_kur"* yazılımı hazırlanmıştır. Bu yazılım ile GPMC modülü 16-bit adres/veri çoğullamalı NOR flash hafıza modunda çalışacak şekilde ayarlanır. GPMC arayüzünün adres uzayındaki yeri ve genişliği konfigüre edilir. GPMC modülünün asenkron haberleşme zamanlamaları düzenlenir ve GPMC haberleşme saati kurulur. Bu yazılım oluşturulurken *Texas Instruments* tarafından sağlanan *AM3359*'un teknik referans dokümanından [74] yararlanılmıştır.

GPMC arayüzü kütüphanesi kullanıcı uzayında çalışacak şekilde yazılmıştır. GPMC modülü fiziksel adresine kullanıcı uzayından erişim *Linux*'un "*mmap"* fonksiyonu kullanılarak sağlanır. GPMC arayüzü kütüphanesinde "*mmap"* fonksiyonu ile GPMC modülünün fiziksel adresi kullanıcı uzayındaki sanal adres uzayına haritalanır (memory mapping). GPMC arayüz kütüphanesinde "*fifo_open, fifo_close, fifo_read, fifo_write, direct_read, direct_write, fifo_reset, system_reset, fifo_getSize, fifo_getNbFree, fifo_getNbAvailable"* fonksiyonları bulunmaktadır.

GPMC arayüzü kütüphanesinde *"fifo_open"* ve "*fifo_close*" fonksiyonları ile arayüz açılır ve kapatılır. "*fifo_read"* ve *"fifo_write"* fonksiyonları ile kontrollü okuma ve yazma, "*direct_read"* ve *"direct_write"* fonksiyonları ile kontrolsüz okuma ve yazma yapılır. Burada kontrolden kasıt okuma ve yazma yapılırken "*fifo_getNbFree"* ve *"fifo_getNbAvailable"* fonksiyonları ile okuma ve yazma FIFO'larının dolu ya da boş olduğunun kontrol edilmesidir. "*fifo_getSize"* fonksiyonu ile yazma ve okuma FIFO'larının boyutu öğrenilirken, "*fifo_reset"* fonksiyonu ile bu FIFO'lar boşaltır. "*system_reset"* fonksiyonu ise FPGA üzerindeki HOG Bloğu'nu resetler.

7.2.2.2 *SVM*^{light} Destek Vektör Makinesi Kütüphanesi

SVM^{light} destek vektör makinesi kütüphanesi hakkında Bölüm 6.2'de bilgiler verilmiş ve derleme aşamaları gösterilmiştir. Bu kütüphanenin kullanımı Bölüm 6.3 ile benzer olduğundan burada tekrar anlatılmayacaktır.

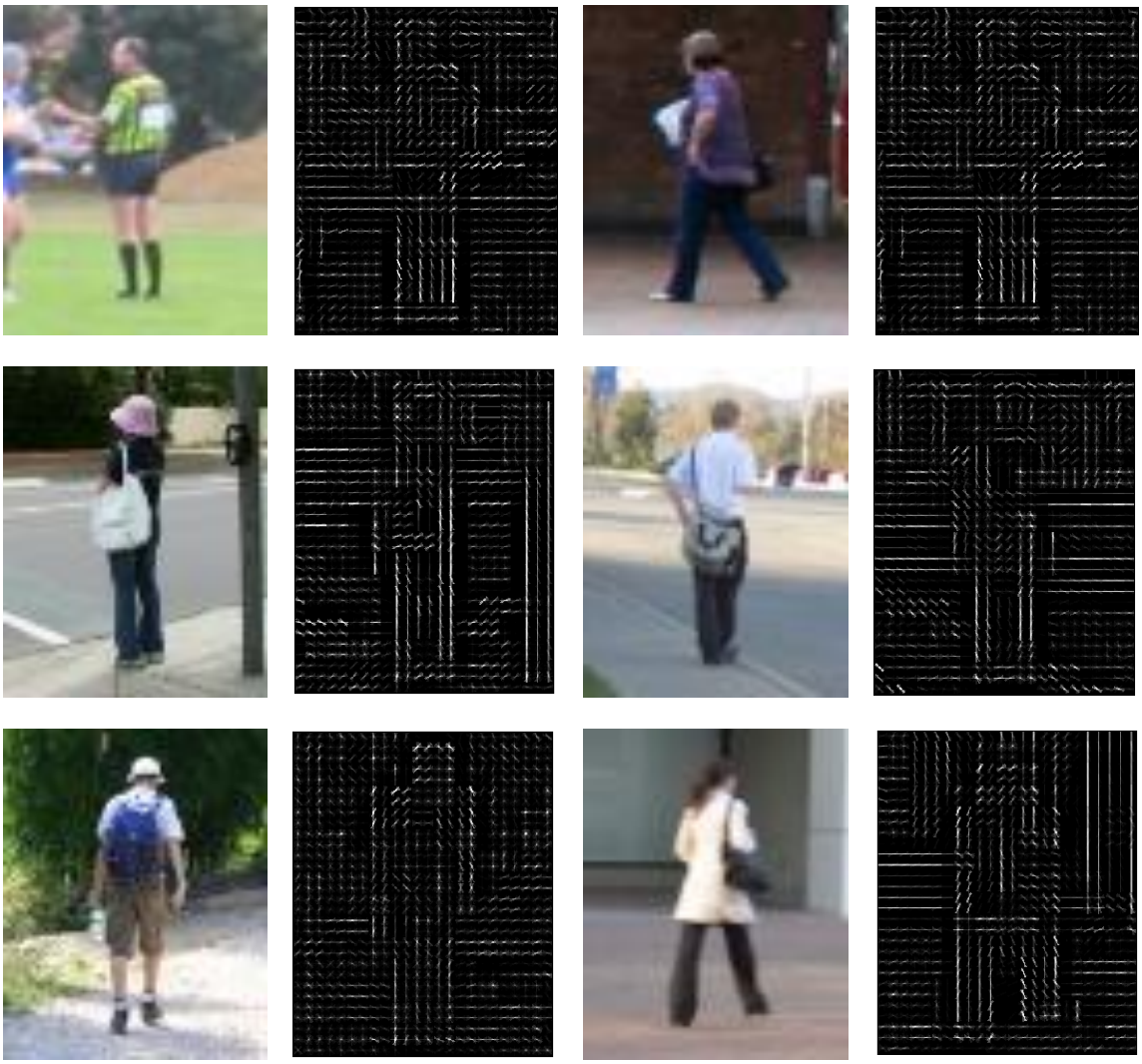
7.2.2.3 HOG kullanıcı yazılımları

Tasarlanan HOG yazılımları ile GPMC arayüzü kütüphanesi kullanılarak giriş görüntülerinin öznitelikleri çıkarılır. Sistemde ilk olarak kendisine gelen tek bir giriş görüntüsünün özniteliğini çıkaran "*hogdev"* yazılımı oluşturulmuştur. Bu yazılım INRIA yaya veri tabanında [8] bulunan 64x80 çözünürlüğündeki örnekler için optimize edilmiştir. Aşağıda verildiği gibi parametre olarak sadece giriş görüntüsünün adını almaktadır.

```
$hogdev <giris goruntusu>
```

Yazılımda görüntü gönderilmeden önce lüminans değeri hesaplanır. FPGA'ya gri ölçekli lüminans görüntüsü gönderilir.Şekil 7.10'de *"hogdev"* yazılımına gönderilen test görüntüsü ve çıkış öznitelikleri yön haritası verilmektedir. Burada çıkış özniteliklerinden yön haritası *MATLAB* üzerinde elde edilmektedir.

Şekil 7.10 Sisteme verilen test görüntüsü ve çıkış öznitelikleri yön haritası



Şekil 7.11 Sistem verilen INRIA yaya veritabanı test görüntüleri ve çıkış öznitelikleri yön haritası

INRIA yaya veritabanındaki yaya örneklerinin *"hogdev"* yazılımı ile öznitelikleri FPGA üzerinde çıkarılmıştır. (Şekil 7.11) Elde edilen öznitelik dosyaları bilgisayar ortamında *MATLAB* yazılımı ile işlenerek yön haritaları çıkarılmıştır. Yön haritası çıkarmada Piotr'un "MATLAB Image&Video" kütüphanesinden faydalanılmıştır [75] .

Sistemde çıkarılan öznitelikler sınıflandırmada *SVM*^{light} kullanılmaktadır. *SVM*^{light} kütüphanesinde bulunan *"svm_learn"* eğitim ve *"svm_classify"* sınıflandırma yazılımlarının giriş formatı Çizelge 7.1'de verilmektedir. Eğitim ya da test için kullanılacak öznitelik veri kümesinde her satıra bir örneğin öznitelikleri gelmelidir. Burada *"target"* etiketi bu örneğin sınıfını bildirmekte olup *"+"* veya *"–"* karakterinden birini alır. Ardından örneğin öznitelik numarası ve değerleri gelmektedir. *"feature"* etiketi öznitelik numarasını, *"value"* etiketi ise bu özniteliğin değerini alır

Çizelge 7.1 SVM^{light} kütüphanesi öznitelik giriş formatı

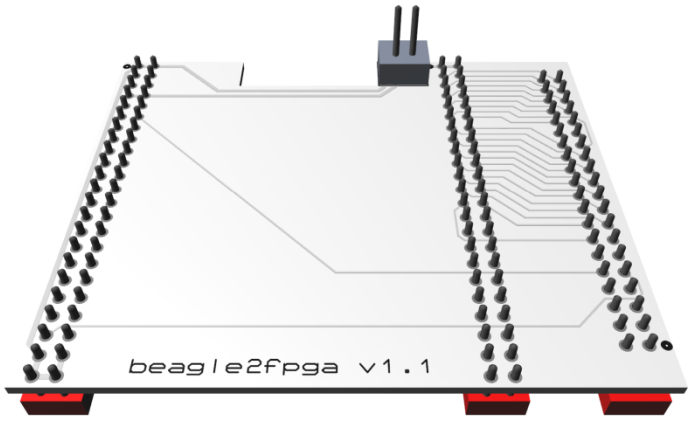
<target><feature>:<value><feature>:<value> ... <feature>:<value>
<target><feature>:<value><feature>:<value> ... <feature>:<value>
<target><feature>:<value><feature>:<value> ... <feature>:<value>
...
...
<target><feature>:<value><feature>:<value> ... <feature>:<value>

Tasarlanan sistemde bir görüntü veritabanı için toplu olarak öznitelik çıkarıp bunları *SVM*^{light} kütüphanesinin kabul edeceği formatta dosyaya yazan *"exractFeature"* programı oluşturulmuştur. Bu program aşağıda verildiği gibi "pozitif örnek dizini", "negatif örnek dizini" ve "çıkış dosyası" parametrelerini almaktadır.

```
$extractFeature –p <pozitif örnek dizini> -n <negatif örnek
dizini> -o <çıkış dosyası>
```

7.2.3 FPGA – Mikroişlemci Arayüz Kartı

Sistemde kullanılan *Altera DE0 Nano* uygulama geliştirme kartı ile *BeagleBone* uygulama geliştirme kartını haberleştirmek üzere bir arayüz kartı tasarlanmıştır. Bu arayüz kartında mikroişlemcinin GPMC sinyalleri FPGA geliştirme kartına taşınmıştır. Ayrıca FPGA için gerekli olan güç de *BeagleBone* tarafından karşılanmakta ve arayüz kartı üzerinden taşınmaktadır. Tasarlanan arayüz kartı görseli verilmektedir.



Şekil 7.12 Tasarlanan FPGA-Mikroişlemci arayüz kartı

7.3 Sistemin Test Edilmesi

Tasarlanan sistemin testi için öncelikle eğitim örnekleri için öznitelikler çıkarılacak ve *SVM*^{light} kütüphanesinde bulunan "*svm_learn"* yazılımı ile sistem eğitilecektir. Ardından eğitilen sistem test edilmek üzere test örnekleri için öznitelikler çıkarılacak ve *SVM*^{light} kütüphanesinde bulunan "*svm_classify"* yazılımı ile sınıflandırma işlemi yapılacaktır.

Öznitelik çıkarmada kullanılan HOG kullanıcı yazılımı olan "*exractFeature"* programına giriş dizini olarak NICTA yaya veritabanının [51] 64x80 çözünürlüğündeki 1000 pozitif ve 1000 negatif eğitim örneği içeren dizinleri aşağıdaki gibi verilmiştir. Böylece 2000 örnek için FPGA'da öznitelikler çıkarılarak "*train.dat"* adındaki eğitim veri seti oluşturulur.

```
$ extractFeature
-p
nicta/positive/NICTA_Pedestrian_Positive_Train_Set_A/00000000/
-n
nicta/negative/NICTA_Pedestrian_Negative_Train_Set_A/00000000/
-o train.dat
```

Çıkarılan öznitelik veri seti "*svm_learn"* yazılımına aşağıdaki gibi verilerek bu eğitim veri seti için bir sınıflandırma model dosyası oluşturulur. Bu model dosyası sınıflandırmada kullanılmak üzere oluşturulan destek vektörlerini içermektedir.

```
$svm_learn train.dat hog_model
```

```
root@hogdev_linux:~/svm_test# ./svm_learn train.dat model
Scanning examples...done
Reading examples into memory...100..200..300..400..500..600..700..800..900..100.
1500..1600..1700..1800..1900..2000..OK. (2000 examples read)
Setting default regularization parameter C=0.0852
Optimizing.....................................................................)
Optimization finished (17 misclassified, maxdiff=0.00092).
Runtime in cpu-seconds: 74.92
Number of SV: 501 (including 224 at upper bound)
L1 loss: loss=86.70720
Norm of weight vector: |w|=4.71775
Norm of longest example vector: |x|=5.67097
Estimated VCdim of classifier: VCdim<=674.95801
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.66
XiAlpha-estimate of the error: error<=19.25% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>81.00% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>80.60% (rho=1.00,depth=0)
Number of kernel evaluations: 36298
Writing model file...done
root@hogdev_linux:~/svm_test#
```

Şekil 7.13 SVM^{light} kütüphanesi ile eğitim işlemi sonucu alınan ekran çıktısı

Öznitelik çıkarmada kullanılan *"exractFeature"* programına bu sefer sınıflandırmada kullanılmak üzere, giriş dizini olarak NICTA yaya veritabanının 64x80 çözünürlüğündeki 1000 pozitif ve 1000 negatif test örneği içeren dizinleri aşağıdaki gibi verilmiştir. Böylece 2000 örnek için FPGA'da öznitelikler çıkarılarak *"test.dat"* adındaki test veri seti oluşturulur.

```
$ extractFeature -p
nicta/positive/NICTA_Pedestrian_Positive_Valid_Set_A/00000000/
-n
nicta/negative/NICTA_Pedestrian_Negative_Valid_Set_A/00000000/
-o test.dat
```

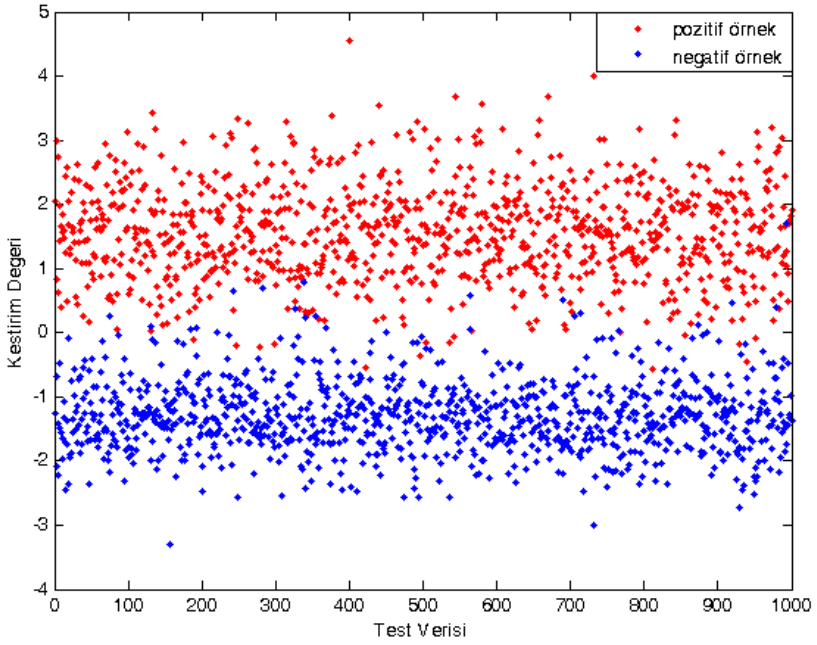
Çıkarılan öznitelik veri seti sınıflandırılmak üzere *"svm_classify"* yazılımına sınıflandırma model dosyası ile birlikte aşağıdaki gibi verilir. Bu yazılım veri setindeki örnekleri sınıflandırarak her satırında pozitif ve negatif sayılar bulunan bir çıktı dosyası verir. Bu sayıların değeri o satıra karşılık düşen giriş örneğinin hangi sınıfta olabileceğini bir olasılık değeri ile vermektedir.

```
$ svm_classify example1/test.dat example1/model
example1/prediction
```

```
root@hogdev_linux:~/svm_test# ./svm_classify test.dat model prediction
Reading model...OK. (501 support vectors read)
Classifying test examples..100..200..300..400..500..600..700..800..900..1000..1e
Runtime (without IO) in cpu-seconds: 0.86
Accuracy on test set: 98.15% (1963 correct, 37 incorrect, 2000 total)
Precision/recall on test set: 97.82%/98.50%
root@hogdev_linux:~/svm_test# _
```

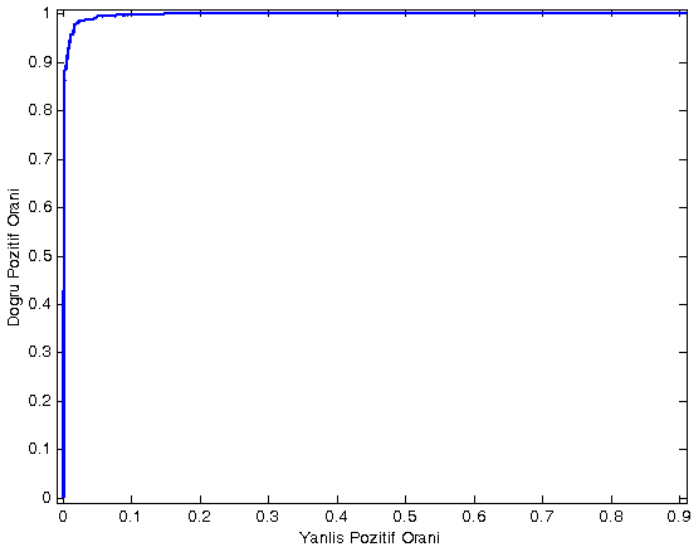
Şekil 7.14 SVM^{light} kütüphanesi ile sınıflandırma işlemi sonucu alınan ekran çıktısı

SVM^{light} kütüphanesinin "*svm_classify*" yazılımı ile sınıflandırma sonucu alınan ekran görüntüsü Şekil 7.14'te verilmektedir. Sınıflandırma sonucu 2000 örnekten 1963 tanesi doğru, 37 tanesi yanlış olarak sınıflandırılmıştır. Sistemin başarımı %98,15 olarak ölçülmüştür. Sınıflandırma sonucu her örnek için hesaplanan kestirim değerleri *MATLAB*'in "plot" fonksiyonu ile çizdirilmiştir. (Şekil 7.15)

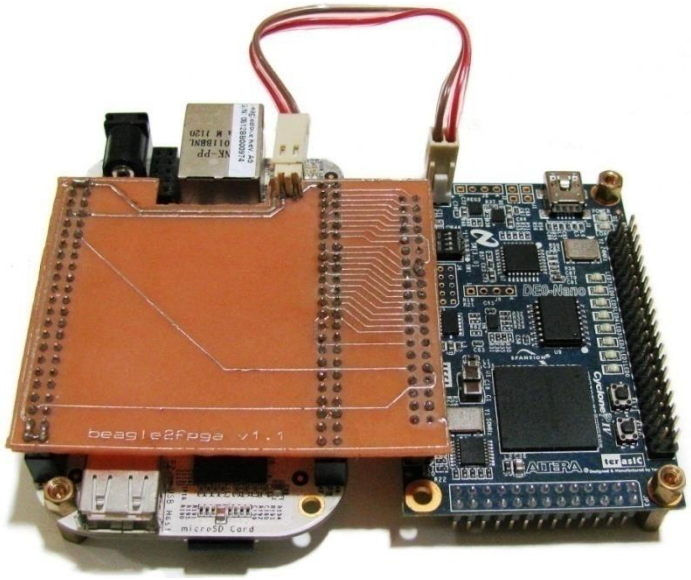


Şekil 7.15 NICTA yaya veri tabanının sistem tarafından sınıflandırılması sonucu kestirim değerleri

Elde edilen sonuçlar Şekil 7.16'da alıcı işlem karakteristiği (Receiver Operating Characteristic – ROC) eğrisi ile gösterilmektedir. Bu grafik yanlış pozitif oranı ile doğru pozitif oranı (saptama oranı) arasındaki ilişkiyi göstermektedir. ROC eğrisi, grafik üzerinde sol üst köşeye ne kadar yakınsa sistemin sınıflandırma başarımı o kadar yüksektir. Bu eğri iki sınıfı birbirinden ayıran eşik değerinin kademeli olarak değiştirilmesiyle elde edilmektedir.



Şekil 7.16 Sistemin NICTA yaya veri tabanı üzerindeki sınıflandırma başarımının ROC eğrisi ile gösterimi



Şekil 7.17 Yaya tanıma uygulaması için oluşturulan sistemin donanımsal altyapısı

BÖLÜM 8

SONUÇ VE ÖNERİLER

Bu tez çalışmasında Yönlü Gradyanların Histogramı (HOG) algoritması ve Destek Vektör Makinesi (SVM) kullanılarak bir yaya tanıma uygulaması gerçekleştirilmiştir. Uygulamada, nesne tanımaya yönelik olarak öznitelik çıkarma işi FPGA'da gerçeklenen HOG algoritması kullanılarak, özniteliklerle sınıflandırma işi ise mikroişlemcide SVM ile yapılmıştır.

Tez çalışması kapsamında, tasarlanan sistemi gerçeklemek üzere FPGA ve mikroişlemcili bir düzenek kurulmuş, düzeneğin birimleri olan FPGA ve mikroişlemciyi haberleştirmek üzere basit bir arayüz kartı tasarlanmıştır. Kurulan düzenekte yer alan FPGA ve mikroişlemci birimlerinin seçiminde düzeneğin taşınabilirliği ve güç tüketimi ön planda tutulmuştur. Bu bağlamda FPGA tasarımı boyunca kaynakların da mümkün olabildiğince az kullanılması amaçlanmıştır. Tasarımda kullanılan *Altera DE0 Nano* uygulama geliştirme kartı üzerinde düşük maliyetli *Altera Cyclone 4* FPGA bulunmaktadır. Tez çalışmasında yapılan VHDL tasarımın FPGA kaynaklarını kullanım miktarı Çizelge 8.1'de verilmiştir.

Çizelge 8.1 Tasarımın FPGA kaynaklarını kullanım raporu

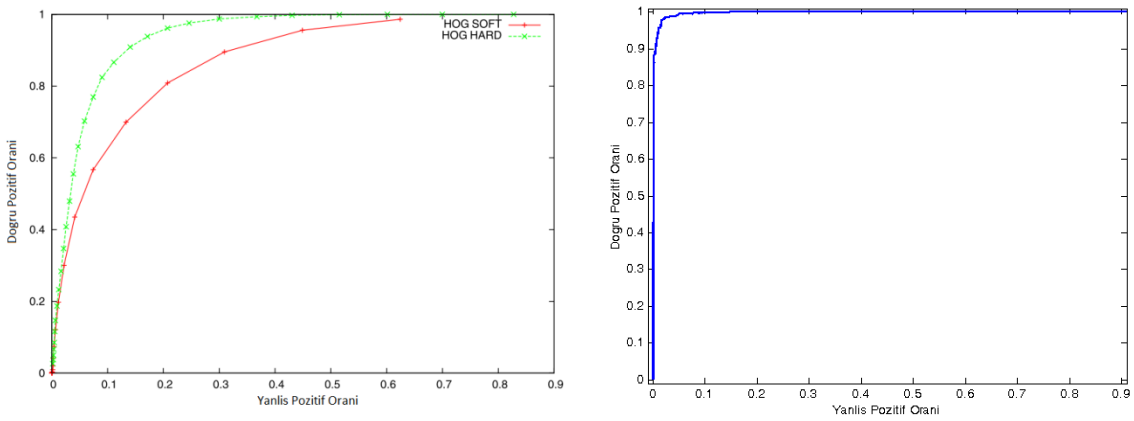
	Kullanılan	Kullanılabilir	%
Lojik Eleman	6,899	22,320	31
Atanmış Lojik Kaydedici	3,596	22,320	16
Blok RAM (bit)	112,620	608,256	19
9-bit Çarpıcı	1	132	1
PLL	1	4	25
Maksimum Frekans (MHz)	129.22		

Sistemde düşük güç tüketimi, maliyeti ve boyutları nedeniyle *ARM* mimarisini kullanan *BeagleBone* uygulama geliştirme kartı mikroişlemci olarak seçilmiştir. Sistemin toplam güç tüketimi Çizelge 8.2'de verilmiştir.

Çizelge 8.2 Tasarlanan sistemin güç tüketim raporu

	mA@5V	mA@3.3V	mA@2.5V	mA@1.2V	watt
Terasic Altera DE0 (Maks.)	225	139	35	654	2.456
BeagleBone Rev A6 (Maks.)	502	-	-	-	2.51
Toplam	727	139	35	654	4.966

Tez kapsamında gerçeklenen yaya tanıma sisteminin başarımı NICTA yaya veritabanı ile ölçülmüştür. Veritabanından alınan 1000 pozitif, 1000 negatif eğitim örneğinin FPGA'da bulunan HOG algoritması ile öznitelikleri çıkarılmış, bu öznitelikler kullanılarak SVM ile sınıflandırma modeli oluşturulmuştur. Aynı veritabanından alınan 1000 pozitif ve 1000 negatif test örneğinin de FPGA'da bulunan HOG algoritması ile öznitelikleri çıkarılmış, çıkarılan öznitelikler sınıflandırma modeli kullanılarak SVM ile sınıflandırılmıştır. 2000 örnekten 1963 tanesinin doğru, 37'sinin yanlış olarak sınıflandırıldığı NICTA yaya veritabanı için sistemin sınıflandırma başarımı %98.15 olarak ölçülmüştür.



Şekil 8.1 NICTA yaya veritabanı için Negi vd. tarafından oluşturulan sistemin (sol) ve bu çalışmada tasarlanan sistemin ROC eğrisi (sağ)

Tez kapsamında gerçeklenen HOG algoritması gerçek zamanlı çalışacak şekilde tasarlanmıştır. Kendisine gelen 60 kare/saniye hızında VGA çözünürlüğündeki görüntüden gerçek zamanlı olarak öznitelik çıkarabilmektedir.

Yapılan çalışmada mikroişlemci üzerinde gerçeklenen SVM tabanlı sınıflandırıcının yaklaşık 4 kare/saniye'lik sınıflandırma hızı, öznitelik çıkarmaya göre oldukça yavaş kalmaktadır. Bu çalışmada kullanılan SVM kütüphanesini hızlandıracak yöntemler çok fazla irdelenmemiştir. İleride yapılacak çalışmalarda kullanılan SVM Kütüphanesi optimize edilerek daha hızlı çalışması sağlanabilir. Görüntünün tüm bölgeleri sınıflandırıcıya verilmeyip, tasarlanacak akıllı arama algoritmaları ile görüntünde sadece yaya bulunması yüksek olan bölgeler için sınıflandırma işlemi yapılarak kabul edilebilir bir sınıflandırma hızına çıkılabilir.

FPGA'da görüntü giriş kaynağı olarak bir araç kamerası kullanımı ile gerçek zamanlı yaya tanıyan bir uygulama oluşturulabilir. Sistemin çıkışından alınan kestirim sonuçları araç bilgisayarına verilerek, yaya tespiti durumunda olası kazaları önlemek üzere, aracın bilgisayar tarafından durdurulduğu, bir kaza önleyici sistem geliştirilebilir.

KAYNAKLAR

[1] Tsukiyama, T. ve Shirai, Y., (1985). "Detection of the movements of persons from a sparse sequence of TV images", Pattern Recognition, 18: 207-213.

[2] Oren, M., Sinha P., Osuna E. ve Poggio T., (1997). "Pedestrian detection using wavelet templates", Computer Vision and Pattern Recognition, 17-19 Jun 1997, San Juan

[3] Gavrila, D.M. ve Philomin, V., (1999). "Real-time object detection for “smart” vehicles ", The Proceedings of the Seventh IEEE International Conference on Computer Vision, 20-27 Sep 1999, Kerkyra

[4] Papageorgiou, C. ve Poggio, P., (2000). "A Trainable System for Object Detection", International Journal of Computer Vision, 38: 15-33.

[5] Viola, P., Jones, M.J. ve Snow, D., (2003). "Detecting pedestrians using patterns of motion and appearance", The Proceedings of the Ninth IEEE International Conference on Computer Vision, 13-16 Oct. 2003 Nice, France

[6] Dalal, N. ve Triggs, B., (2005). "Histograms of Oriented Gradients for Human Detection", IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 25-25 June 2005 San Diego

[7] Center for Biological and Computational Learning, CBCL Pedestrian Database, http://cbcl.mit.edu/software-datasets/PedestrianData.html, 25 Mart 2013.

[8] Dalal N., (2005). INRIA Person Dataset, http://pascal.inrialpes.fr/data/human/, 21 Şubat 2013.

[9] Dalal, N., Triggs, B. ve Schmid, C., (2006). "Human Detection Using Oriented Histograms of Flow and Appearance", 9th European Conference on Computer Vision, 7-13 May 2006, Graz, Austria

[10] Sabzmeydani, P. ve Mori, G., (2007). "Detecting Pedestrians by Learning Shapelet Features", IEEE Conference on Computer Vision and Pattern Recognition, 17-22 June 2007, Minneapolis

[11] Wu, B. ve Nevatia, R., (2007). "Detection and Tracking of Multiple, Partially Occluded Humans by Bayesian Combination of Edgelet based Part Detectors", International Journal of Computer Vision, 75: 247-266.

[12] Bauer, S., Brunsmann, U. ve Schlotterbeck-macht, S., (2009). "FPGA Implementation of a HOG-based Pedestrian Recognition System", MPC-Workshop, July 2009, Aschaffenburg, Germany

[13] Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Miyamoto, R. ve Nakamura, Y., (2009). "Hardware Architecture for HOG Feature Extraction", Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 12-14 Sept. 2009, Kyoto

[14] Negi, K., Dohi, K. , Shibata, Y. ve Oguri, K., (2011). "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm", International Conference on Field-Programmable Technology, 12-14 Dec. 2011 New Delhi

[15] World Health Organization (WHO), (2004). "World report on road traffic injury prevention", Geneva

[16] Wikimedia Foundation, Inc., Outline of object recognition, http://en.wikipedia.org/wiki/Outline_of_object_recognition, 12 Nisan 2013.

[17] De Araújo, S.A. ve Kim, H.Y., (2007). "Grayscale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast ", Advances in Image and Video Technology.

[18] Alpaslan, N., (2013). Gradyan Tabanlı Heterojen Öznitelik Çıkarma Yöntemlerine Yeni Yaklaşımlar, Yüksek Lisans Tezi, İnönü Üniversitesi Fen Bilimleri Enstitüsü, Malatya.

[19] Canny, J., (1986). "A Computational Approach to Edge Detection", Pattern Analysis and Machine Intelligence, PAMI-8: 679 - 698.

[20] Morevec, H.P., (1977). "Towards Automatic Visual Obstacle Avoidance (short version)", Proceedings of the 5th international joint conference on Artificial intelligence, San Francisco

[21] Förstner, W., (1986). "A Feature Based Correspondence Algorithm for Image Matching", International Archives of Photogrammetry and Remote Sensing, 26.

[22] Harris, C. ve Stephens, M., (1988). "A Combined Corner and Edge Detector", Fourth Alvey Vision Conference, Manchester

[23] Tomasi, C. ve Kanade T. , (1991). "Shape and Motion from Image Streams: a Factorization Method Part 3: Detection and Tracking of Point Features", Carnegie Mellon University Technical Report.

[24] Kitchen, L. ve Rosenfeld, A., (1980). "Gray Level Corner Detection", Technical Report of Computer Center 887.

[25] Bastanlar, Y. ve Yardimci, Y., (2007). "Selecting Image Corner Points Using Their Corner Properties", Signal Processing and Communications Applications.

[26] Lowe, D.G., (2004). "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, 60: 91-110.

[27] Bay, H., Tuytelaars, T. ve Van Gool, L. (2006). "SURF: Speeded Up Robust Features", The Proceedings of the 9th European Conference on Computer Vision, 7-13 May 2006, Graz

[28] Matas, J., Chum, O., Urban, M. ve Pajdla, T., (2004). "Robust wide-baseline stereo from maximally stable extremal regions", Image and Vision Computing, 22: 761–767.

[29] Güney, M. ve Arıca, N., (2009). "Desen Tabanlı İlgi Bölgesi Tespiti", Naval Science and Engineering, 5: 94-106.

[30] Safran, M.I. ve Oktem, R., (2007). "A Fast Hough Transform Approximation and Its Application for Barcode Localization", IEEE 15th Signal Processing and Communications Applications, 11-13 June 2007 Eskisehir

[31] Shashua, A., Gdalyahu, Y. ve Hayun, G. (2004). "Pedestrian detection for driving assistance systems: single-frame classification and system level performance", The Proceedings of IEEE Intelligent Vehicles Symposium, 14-17 June 2004 Parma

[32] Watanabe, T., Ito, S. ve Yokoi, K. (2009). "Co-occurrence histograms of oriented gradients for pedestrian detection", Third Pacific Rim Symposium on Image and Video Technology, 13-16 Jan. 2009, Tokyo

[33] Viola, P. ve Jones, M., (2001). "Rapid object detection using a boosted cascade of simple features", Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 8-14 December 2001, Kauai, Hawaii

[34] Cao, T.P. ve Deng, G., (2008). "Real-Time Vision-Based Stop Sign Detection System on FPGA", Digital Image Computing: Techniques and Applications, 1-3 Dec. 2008 Canberra, ACT

[35] Karakaya, F., Altun, H. ve Çavuşlu, A., (2009). "Gerçek Zamanlı Nesne Tanıma Uygulamaları için HOG Algoritmasının FPGA Tabanlı Gömülü Sistem Uyarlaması", IEEE 17th Signal Processing and Communications Applications Conference, 9-11 Nisan 2009, Kocaeli

[36] Prisacariu, V. ve Reid, I., (2009). fastHOG - a real-time GPU implementation of HOG, http://www.robots.ox.ac.uk/~victor/papers/pdf/prisacariu_reid_tr2310_09.pdf , 21 Aralık 2012.

[37] ITU-R BT.601, (1995). Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios, International Telecommunication Union,

[38] Dalal, N., (2006). "Finding People in Images and Videos", Master Thesis, Grenoble Institute of Technology Computer Sciences, Applied and Pure Mathematics, Grenoble, France.

[39] Vapnik, V.N., (2000). The Nature of Statistical Learning Theory, New Jersey.

[40] Kavzoğlu, T. ve Çölkesen, İ., (2010). "Destek Vektör Makineleri ile Uydu Görüntülerinin Sınıflandırılmasında Kernel Fonksiyonlarının Etkilerinin İncelenmesi ", Harita Dergisi, 144: 73-82.

[41] Bottou, L., Cortes, C., Denker, J., Drucker, H., Guyon, I., Jackel, L., LeCun, Y., Muller, U., Sackinger, E., Simard, P., ve Vapnik, (1994). "Comparison of classifier methods:A case study in handwriting digit recognition", The Proceedings of the 12th IAPR International Conference on Pattern Recognition, 9-13 Oct 1994, Jerusalem

[42] Cao, L.J. ve Tay, F., (2003). "Support Vector Machine With Adaptive Parameters in Financial Time Series Forecasting", IEEE Transactions On Neural Networks, 14: 1506 - 1518.

[43] Georgoulas, G., Georgopoulos, V.C. ve Stylios, C.D., (2006). "Speech Sound Classification and Detection of Articulation Disorders with Support Vector Machines and Wavelets", The Proceedings of the 28th Annual International Conference of the IEEE, 30 Aug - 3 Sep 2006 New York

[44] Hyungkeun, J., Kyunghee, L. ve Sungbum, P., (2004). "Eye and face detection using SVM", International Conference on Intelligent Sensors,Sensor Networks and Information Processing, 14-17 Dec 2004

[45] İbrikçi, T., Çakmak, A., Ersöz, D. ve Açıkkar, M., (2004). "Destek Vektörlerinin Proteinlerin İkincil Yapılarını Tahmin Etmek İçin Uygulanması", National Meeting on Biomedical Engineering,

[46] Altera, Quartus II Web Edition Software, http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html, 17 Nisan 2013.

[47] Xilinx, ISE WebPACK Design Software, http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm, 17 Nisan 2013.

[48] Mentor Graphics, ModelSim PE Student Edition - HDL Simulation, http://model.com/content/modelsim-pe-student-edition-hdl-simulation, 17 Nisan 2013.

[49] Sigasi, Sigasi Starter Edition, http://www.sigasi.com/, 17 Nisan 2013.

[50] Wilson, T., Glatz, M. ve Hödlmoser, M., (2009). "Pedestrian Detection Implemented on a Fixed-Point Parallel Architecture", 13th International Symposium on Consumer Electronics, 25-28 May 2009 Kyoto

[51] NICTA, Computer Vision Datasets, http://www.nicta.com.au/research/projects/AutoMap/computer_vision_datasets, 25 Mart 2013.

[52] Terasic Technologies, Altera DE0 Board, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364, 11 Mart 2013.

[53] Joachims, T., (2008). Support Vector Machine, http://svmlight.joachims.org/, 2013 2 Şubat.

[54] Canonical Ltd., Ubuntu: The world's most popular free OS, http://www.ubuntu.com/, 21 Nisan 2013.

[55] Coley, G., (2012). BeagleBone System Reference Manual Rev A6.0.0, http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf, 15 Aralık 2012.

[56] BeagleBoard.org, BeagleBone, http://beagleboard.org/Products/BeagleBone, 12 Aralık 2012.

[57] Texas Instruments, Sitara ARM Cortex-A8 Microprocessor http://www.ti.com/product/am3359, 22 Nisan 2013.

[58] Texas Instruments, Single-Chip PMIC for Battery-Powered Systems for AM335x ZCZ, DDR2, http://www.ti.com/product/tps65217b, 22 Nisan 2013.

[59] Texas Instruments, Sitara™ AM335x ARM® Cortex™-A8 Microprocessors, http://www.ti.com/lit/gpn/am3359, 23 Şubat 2013.

[60] Computer Architecture, Levy, M., (2003). The History of The ARM Architecture: From Inception to IPO, http://www.reds.ch/share/cours/ReCo/documents/TheHistoryOfTheArmArchitecture.pdf, 17 Nisan 2013.

[61] DENX Software Engineering, Das U-Boot -- the Universal Boot Loader, http://www.denx.de/wiki/U-Boot, 18 Nisan 2013.

[62] Koç, N., (2012). Gömülü Linux Sistemleri, Pusula Yayıncılık, İstanbul.

[63] D'Amore, G., (1995). The Linux Filesystem Standard, http://garrett.damore.org/fsstnd/fsstnd.html, 27 Nisan 2013.

[64] Linux Kernel Organization, The Linux Kernel Archives, http://www.kernel.org, 17 Nisan 2013.

[65] Joachims, T., (2002). Learning to Classify Text Using Support Vector Machines, Kluwer Academic Publishers, New York.

[66] Morik, K., Brockhausen, P. ve Joachims, T. (1999). "Combining statistical learning with a knowledge-based approach - A case study in intensive care monitoring", Machine Learning.

[67] Joachims, T., (2000). "Estimating the Generalization Performance of a SVM Efficiently", Proceedings of the International Conference on Machine Learning,

[68] Cheng, H., Zheng, N. ve Qin, J., (2005). "Pedestrian detection using sparse gabor filter and support vector machine", IEEE Intelligent Vehicles Symposium, 6-8 June 2005 Las Vegas

[69] Carnegie Group, Reuters-21578 text categorization test collection, http://download.joachims.org/svm_light/examples/example1.tar.gz, 3 Mart 2013.

[70] Oren, M., Osuna, E. ve Poggio, T., (1997). "Pedestrian Detection Using Wavelet Templates", Conference on Computer Vision and Pattern Recognition, June 17-19, 1997, San Juan, Puerto Rico

[71] Viola, P., Jones, M.J. ve Snow, D., (2005). "Detecting pedestrians using patterns of motion and appearance", Ninth IEEE International Conference on Computer Vision, 13-16 Oct. 2003 Nice, France

[72] Texas Instruments, AM3715/03 GPMC Subsystem, http://processors.wiki.ti.com/index.php/AM3715/03_GPMC_Subsystem, 4 Şubat 2013.

[73] Valentfx, LOGI-BONE - FPGA Cape for the Beaglebone, http://valentfx.com/prj/fpga-dev/19-logi-bone-cape, 30 Aralık 2012.

[74] Texas Instruments, AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. H), http://www.ti.com/lit/ug/spruh73h/spruh73h.pdf, 8 Nisan 2013.

[75] Dollár, P., (2013). Piotr's Image and Video Matlab Toolbox (PMT), http://vision.ucsd.edu/~pdollar/toolbox/doc/, 19 Şubat 2013.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Ahmet Remzi ÖZCAN

Doğum Tarihi ve Yeri : 25.11.1986, TRABZON

Yabancı Dili : İngilizce

E-posta : ozcanar@gmail.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Lisans	Elektronik ve Haberleşme Mühendisliği	Yıldız Teknik Üniversitesi	2009
Lise	Fen Bilimleri	Bursa A.O.S. Fen Lisesi	2004

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2010	TÜBİTAK BİLGEM UEKAE	Araştırmacı/Uzay Kripto Uygulamaları
2010	PAVO Tasarım Üretim Elektronik Tic. A.Ş.	Tasarım Mühendisi