

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

**MOBİL ROBOT PLATFORMU ÜZERİNDE DSP TABANLI
GERÇEK ZAMANLI ÇALIŞAN ŞERİT TESPİTİ
ALGORİTMASININ GELİŞTİRİLMESİ ve OPTİMAZSYONU**

YÜKSEK LİSANS

Elektronik Müh. Gürkan KÜÇÜKYILDIZ

Anabilim Dalı: Mekatronik Mühendisliği

Danışman: Doç. Dr. Hasan OCAK

KOCAELİ, 2012

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

**MOBİL ROBOT PLATFORMU ÜZERİNDE DSP TABANLI
GERÇEK ZAMANLI ÇALIŞAN ŞERİT TESPİTİ
ALGORİTMASININ GELİŞTİRİLMESİ ve OPTİMAZSYONU**

YÜKSEK LİSANS TEZİ

Elektronik Müh. Gürkan KÜÇÜKYILDIZ

Tezin Enstitüye Verildiği Tarih: 19 Aralık 2011

Tezin Savunulduğu Tarih: 12 Ocak 2012

Tez Danışmanı

Doç. Dr. Hasan OCAK


(.....)

Üye

Doç. Dr. H.Metin ERTUNÇ


(.....)

Üye

Yrd. Doç. Dr. Hakan ERDOĞAN


(.....)

KOCAELİ, 2012

ÖNSÖZ ve TEŞEKKÜRLER

Gelişen teknoloji ile beraber önceden beri insanlığın bir rüyasını süsleyen otonom araçlar günümüzde trafikte yer almaya başlamıştır. Bu otonom araçların, çevreyi algılayabilmesi için birçok çeşitli sensörler kullanılabileceği gibi görüntü işleme de kullanılabilir. Görüntü işleme tabanlı şerit bulma işleminde görüntüdeki gürültüler, ortamdaki ışık miktarının az ya da çok olması karşımıza sorun olarak çıkmaktadır. Ayrıca görüntü işleme algoritmalarının hesapsal yükü yüzünden görüntü işlemenin gerçek zamanlı olarak çalışmasında büyük sorunlar çıkmaktadır.

Bu çalışmada, şerit takibi için bir mobil robot platformu geliştirilmiş ve görüntü işleme tabanlı şerit bulma kodu geliştirilen mobil robot platformu üzerinde gerçekleştirilmiştir. Tez çalışması sırasında geliştirilen kodun gerçek zamanlı olarak çalıştırılması için birçok optimizasyon yapılmıştır.

Tez çalışması süresince her zaman ilgilerini yakından hissettiğim, çalışmalarına yapmış olduğu katkıları ve fikirleri sayesinde bugünlere gelmemde yol gösterici olan değerli danışmanım Sn. Doç. Dr. Hasan OCAK' a teşekkür ederim.

Ayrıca tez çalışmamın, üretilmesi konusunda benden asla yardımlarını esirgemeyen MAVI LAB çalışanlarına ve son olarak hayatım boyunca beni destekleyen aileme teşekkür ederim.

İÇİNDEKİLER

ÖNSÖZ ve TEŞEKKÜR.....	i
İÇİNDEKİLER	ii
ŞEKİLLER DİZİNİ.....	iv
TABLOLAR DİZİNİ	vii
SİMGELER ve KISALTMALAR	viii
ÖZET.....	viii
İNGİLİZCE ÖZET	ix
1. GİRİŞ	1
1.1. Otonom Araçlar.....	1
1.1.1. Otonom araçların kullanım alanları	2
1.1.1.1. Otonom hava araçları	2
1.1.1.2. Otonom kara araçları.....	3
1.2. Şerit Tespiti	4
1.2.1. Şerit tespiti için kullanılan yöntemler	5
1.2.1.1. İlgilenilen bölge için rastgele Hough dönüşümü	5
1.2.1.2. Farklı Çözünürlüklerdeki Hough Dönüşümü.....	6
1.2.1.3. Yönlendirilebilir filtre tabanlı şerit takibi	6
1.2.1.4. Dinamik programlama yöntemi	7
1.2.1.5. B-Snake yöntemi.....	7
1.2.1.6. SVM ile şerit değişiminin bulunması.....	8
1.2.1.7. Kalman filtresi kullanılarak şerit tespiti	8
1.2.1.8. İki kamera kullanarak engel tanıma ve şerit takibi	9
1.2.1.9. Catmull-Rom spline kullanarak şerit takibi	10
1.2.1.10. Renk modeli tabanlı şerit çizgilerinin bulunması	10
1.2.1.11. İki kamera kullanarak üç boyutlu şerit çizgilerinin bulunması	11
1.2.1.12. İmgedeki şekillerin benzerliği algoritması ile şerit tespiti	12
1.2.1.13. Oryantasyon ve uzunluk azaltımı ile şerit tespiti	12
1.2.1.14. Yapay Sinir Ağları Kullanarak Şerit Tespiti.....	13
1.2.2. Tezin Literatüre Katkısı	13
2. GÖRÜNTÜ İŞLEME TABANLI ŞERİT TESPİTİ ALGORİTMASI	15
2.1. Giriş.....	15
2.2. Algoritma	16
2.2.1. Filtreleme	18
2.2.2. Kenar bulma filtreleri.....	19
2.2.2.1 Sobel kenar bulma filtresi	20
2.2.3. Hough dönüşümü	21
2.2.4. Hough dönüşümü ile doğru tespiti	22
2.2.4.1. Hough dönüşümü ile şerit tespiti	26
2.2.4.2. Eliminasyon yöntemleri	26
2.2.5. Ters perspektif dönüşümü	33
2.2.6. Mobil robotun hareket yönünün kontrolü	37

2.3. Optimizasyon	42
2.3.1. Açık bilgisinin filtre çıkışından alınması	42
2.3.2. Trigonometrik Fonksiyonların Değerlerinin LUT'dan Elde Edilmesi.....	43
2.3.3. Hafıza Organizasyonu.....	44
3. MOBİL ROBOTUN TASARIMI	46
3.1. Giriş.....	46
3.2. Elektronik Devrelerin Tasarımı.....	47
3.2.1. Motor sürücü devresi ve PCB tasarımı	47
3.2.2. Regülatör devresi ve PCB tasarımı	51
3.2.3. BF561 işlemcisi.....	54
3.2.4. Görüntü işleme kartı (ADSP BF561 EZ-KIT LITE)	55
3.3. Mekanik Tasarım	57
3.3.1. Elektronik kutu.....	58
3.3.2. Ana gövde	58
3.3.3. Motor tutucuları	58
4. DENEYSEL SONUÇLAR	59
4.1. Giriş.....	59
4.2. Şerit Çizgilerinin Kesikli Olması Durumu.....	59
4.3. Ortamdaki Işık Şiddetinin Az Olması Durumu.....	61
4.4. Ortamda Fazla Işık Olması Durumu	62
4.5. Tek Şerit Olması Durumu	64
4.5.1. Sağda tek şerit olması durumu	64
4.5.2. Solda tek şerit olması durumu.....	65
SONUÇLAR ve ÖNERİLER.....	67
KAYNAKLAR	69
EKLER.....	74
ÖZGEÇMİŞ	103

ŞEKİLLER DİZİNİ

Şekil 1.1: Gnat-750 otonom hava aracı.....	3
Şekil 1.2: Google'un otonom arabası.....	4
Şekil 1.3: ASELSAN'ın İZCİ arabası.....	4
Şekil 2.1: Algoritmanın genel akış şeması.....	17
Şekil 2.2: Örnek imge ve filtreden geçirilmiş hali.....	18
Şekil 2.3: Örnek kenar bulma filtreleri sonuçları.....	19
Şekil 2.4: a)Orijinal imge b)Düşey Sobel kenar filtresi uygulanmış imge.....	21
Şekil 2.5: r ve θ parametre uzayı.....	23
Şekil 2.6: Hough dönüşümü akış şeması.....	24
Şekil 2.7: Hough matrisi için orijinal imge.....	25
Şekil 2.8: Elde edilen Hough dizisi.....	25
Şekil 2.9: Hough dönüşümü.....	25
Şekil 2.10: Örnek Hough dizisi.....	25
Şekil 2.11: Eşiklemeyen sonra elde edilen Hough dizisi.....	25
Şekil 2.12: Hough dizisindeki maksimumların bulunmasının akış şeması.....	28
Şekil 2.13: İki şerit çizgisinin bulunmasının akış şeması.....	30
Şekil 2.14: a)Orijinal imge b)Bütün Hough çizgileri c) Bulunan şerit çizgileri.....	31
Şekil 2.15: Tek şerit çizgisinin bulunmasının akış şeması.....	32
Şekil 2.16: a) Orijinal İmge b) Bütün Hough çizgileri c) Bulunan şerit çizgisi.....	33
Şekil 2.17: Perspektif hatası.....	34
Şekil 2.18: Şeritlere uygulanan perspektif dönüşümü.....	34
Şekil 2.19: Ters perspektif dönüşümü için orijinal imge.....	35
Şekil 2.20: Ters perspektif dönüşümü alınmış imge.....	35
Şekil 2.21: İmge uzayı ve perspektif etkisi.....	36
Şekil 2.22: Şerit çizgilerinin ters perspektif dönüşümü.....	38
Şekil 2.23: Mobil robotun varması istenen hedef nokta.....	38
Şekil 2.24: Mobil robotun yönünün kontrolü akış şeması.....	40
Şekil 3.1: Geliştirilen Mobil robot platformu.....	46
Şekil 3.2: Motor sürücü ve kontrol devresi.....	48
Şekil 3.3: Motor sürücü ve kontrol devresi PCB çizimi.....	48
Şekil 3.4: TLP521-4 entegresi.....	49
Şekil 3.5: 7805 entegresi.....	50
Şekil 3.6: LMD18200T entegresi.....	51
Şekil 3.7: Regülatör devresi.....	52
Şekil 3.8: Regülatör devresi PCB çizimi.....	52
Şekil 3.9: LM2576 entegresi.....	53
Şekil 3.10: LM7809 entegresi.....	54
Şekil 3.11: ADSP BF-561 EZ-KIT LITE.....	56
Şekil 3.12: ADSP BF-561 EZ-KIT LITE Şeması.....	56
Şekil 3.13: Mobil robotun mekanik çizimi.....	57
Şekil 3.14: Elektronik Kutu.....	58
Şekil 4.1: Şerit çizgilerinin kesikli olması durumunda alınan iki örnek görüntü.....	60

Şekil 4.2: Kesikli şerit çizgileri için alınan iki örnek görüntüden elde edilen bütün Hough çizgileri	60
Şekil 4.3: Kesikli şerit çizgileri için alınan iki örnek görüntüden eliminasyon sonrası tespit edilen şerit çizgileri	60
Şekil 4.4: Ortamda yetersiz ışık olması durumunda alınan iki örnek görüntü.....	61
Şekil 4.5: Ortamda yetersiz ışık olması durumunda alınan iki örnek görüntüden elde edilen Hough çizgileri	61
Şekil 4.6: Ortamda yetersiz ışık olması durumunda alınan iki örnek görüntüden elde edilen şerit çizgileri	61
Şekil 4.7: Ortamda fazla ışık olması durumunda alınan iki örnek görüntü	63
Şekil 4.8: Ortamda fazla ışık olması durumunda alınan iki örnek görüntüden elde edilen Hough çizgileri	63
Şekil 4.9: Ortamda fazla ışık olması durumunda alınan iki örnek görüntüden elde edilen şerit çizgileri	63
Şekil 4.10: Sağda tek şerit olması durumunda alınan iki örnek görüntü	65
Şekil 4.11: Sağda tek şerit olması durumunda alınan iki örnek görüntüden elde edilen Hough çizgisi.....	65
Şekil 4.12: Sağda tek şerit olması durumunda alınan iki örnek görüntüden elde edilen şerit çizgisi	65
Şekil 4.13: Solda tek şerit olması durumunda alınan iki örnek görüntü	66
Şekil 4.14: Solda tek şerit olması durumunda alınan iki örnek görüntüden elde edilen Hough çizgisi.....	66
Şekil 4.15: Solda tek şerit olması durumunda alınan iki örnek görüntüden elde edilen şerit çizgisi	66

TABLÖLAR DİZİNİ

Tablo 2.1 : Trigonometrik Fonksiyonların Değerlerinin LUT'dan elde edilmesi	44
---	----

SİMGELER ve KISALTMALAR

BF	: Blackfin işlemci ailesi
DSP	: Digital Signal Processor (Sayısal işaret işleyici)
G_x	: Dikey Sobel filtresi
G_y	: Yatay Sobel filtresi
H	: Perspektif dönüşümü için dönüşüm matrisi
LUT	: Look Up Table (Referans Tablosu)
PCB	: Printed Circuit Board (Baskı devre)
P	: Proportional (Oransal kontrolör)
PWM	: Pulse Width Modulation (Darbe Genişlik Modülasyonu)
r	: Hough uzayında doğrunun orijin noktasından olan uzaklığı (piksel)
SVM	: Support Vector Machines (Destek vektör makinaları)
θ	: Hough uzayında doğrunun açısı (derece)

MOBİL ROBOT PLATFORMU ÜZERİNDE DSP TABANLI GERÇEK ZAMANLI ŞERİT TESPİTİ ALGORİTMASININ GERÇEKLEŞTİRİLMESİ ve OPTİMİZASYONU

Gürkan KÜÇÜKYILDIZ

Anahtar Kelimeler: Şerit Takibi, Görüntü İşleme, Mobil Robot, P Kontrol, Gerçek Zamanlı Kontrol, DSP

Özet: Bu çalışmada, otonom araçların kontrolünde önemli problemlerin arasında olan görüntü işleme tabanlı gerçek zamanlı şerit takibi üzerinde çalışılmıştır. Mobil robotun hareketi birbirinden bağımsız olarak çalışan dört adet DC motorla sağlanmıştır. Görüntü işleme kodu Visual DSP 5.0 ortamında geliştirilmiş ve geliştirilen bu kod ADSP BF561 EZ KIT LITE geliştirme kartı ile BF-561 işlemcisi üzerinde çalıştırılmıştır. Başlangıçta saniyede sadece üç görüntü işleyebilen görüntü işleme kodu, yapılan optimizasyon sonrasında gerçek zamanlı olarak çalışabilecek hale getirilmiştir.

Çalışmanın görüntü işleme kısmında görüntünün Hough dönüşümü alındıktan sonra elde edilen Hough çizgileri aday şerit çizgileri olarak belirlenmiştir. Elde edilen bu Hough çizgileri arasından istenilen şerit çizgilerinin bulunabilmesi için çeşitli eliminasyonlar yapılmıştır. Şerit çizgileri bulunduktan sonra perspektif dönüşümü alınarak şerit çizgilerinin gerçek dünya koordinatlarındaki konumları bulunmuştur. Şerit çizgilerinin gerçek dünya koordinatları ve mobil robotun merkezi kullanılarak mobil robotun yönelim açısı belirlenmiştir.

Geliştirilen mobil robot platformu ve şerit bulma algoritması şerit çizgilerinin kesikli olması, ortamda az ışık olması, ortamda fazla ışık olması ve tek şerit çizgisi olması durumlarında denenmiştir. Bütün bu koşullar altında algoritmanın başarılı olarak çalıştığı ve ortamdaki bozucu etkilerden etkilenmeden şerit çizgisini tespit edebildiği gözlenmiştir.

DEVELOPMENT and OPTIMIZATION of DSP BASED REAL TIME LANE DETECTION ALGORITHM on a MOBILE ROBOT PLATFORM

Gürkan KÜÇÜKYILDIZ

Key Words: Lane Detection, Image Processing, Mobile Robot, P Control, Real Time Control, DSP

Abstract: In this study, image processing based real time lane detection, which is the one of significant problems in autonomous vehicles control, was explored. A mobile robot platform was developed for such purpose. The motion of the mobile robot was provided by four (direct current) DC motors which were independently controlled. Image processing code was developed on Visual DSP 5.0 environment and the code was run on BF-561 processor embedded in ADSP BF561 EZ KIT LITE evaluation board. After the optimization of the image processing code, the number of processed frames processed per second increased from 3 to 30, which should be satisfactory for real time applications.

In the image processing algorithm, Hough lines which were obtained from the Hough transform of image, were called candidate lane marks. Various elimination methods were implemented on these candidate lane marks to detect the actual lane marks. After the determination of the actual lane marks, the real world coordinates of the lane marks were computed by using inverse perspective mapping. The heading angle of mobile robot was determined based on the position of the lane marks and the center of the mobile robot.

The developed mobile robot platform and the lane detection algorithm was tested under various conditions including dashed lane marks, varying lighting conditions, and presence of one lane mark only. It was observed that the algorithm performed successfully and detected lane marks even in the presence of various disturbance effects under these conditions.

1. GİRİŞ

1.1. Otonom Araçlar

Otonom araçlar herhangi bir sürücü yardımı olmadan, çevreyi algılayarak kendi kendini yönlendirebilen araçlardır. Otonom araçlar algıladıkları çevreye göre önceden belirlenen bir hedefe gitmek için kendi yönlerini bulabilir ve yolculuk sırasında karşılarına çıkan engellerden zarar görmeden ilerleyebilirler. Otonom araçlarda çevreyi algılayabilmek için lazer, radar, GPS, görüntü işleme ve çeşitli sensörler kullanabilir.

Otonom araçlar bu algılayıcılardan aldıkları bilgiyi çeşitli kontrol algoritmalarıyla işleyerek kendisi için en uygun rotayı belirleyebilir ve yoldaki engel durumuna göre rotasını güncelleyebilirler. Her alanda hızla gelişen ve yenilenen teknoloji ile birlikte eskiden bir rüya olan otonom araçlar artık günümüzde yaygın bir halde kullanılmaya başlamıştır. Otonom araçların kullanılmasının sebebi sürücü hatasından kaynaklanan kazaların en aza indirilmesi ve olası bir kazada araç içerisinde aracı kullanan herhangi bir sürücü olmadığı için kazalarda can kaybının önlenmesi olarak söylenebilir.

1980'lerde Münih'teki Bundeswehr Üniversitesi'nde çalışan Ernst Dickmans ve ekibi tarafından geliştirilen Mercedes Benz'in robotik aracı trafiğin olmadığı yolda 100 km/s hıza ulaşabilmeyi başarmıştır. Yine 1980'lerde lazer ve görüntü işleme teknolojilerini kullanan ve 30 km/s hızla giden DARPA projesi geliştirilmiştir. Günümüzde ise General Motors, Volkswagen, Audi, BMW, Volvo, Google kendi otonom araçlarının testlerine başlamış bulunmaktadır. General Motors firması otonom araçlarının 2015 yılında teste başlayacağını ve 2018 yılında bu araçların yollarda olabileceğini belirtmiştir [1].

Otonom araçların trafikte kullanılması ile ilgili kanun Google'un da büyük desteğiyle, dünyada ilk olarak Amerika'nın Nevada eyaletinde Haziran 2011'de çıkarılmıştır [1].

Sensörler kullanılarak yapılan çalışmalarda ışık sensörleri tercih edilebileceği gibi kızılötesi sensörler de konum belirlemede kullanılabilir [2]. Görüntü işleme tabanlı algılama da mobil robotlarda çok sık kullanılan bir tekniktir [3]. Görüntü işleme ile birlikte radar ve çeşitli sensörler de kullanılabilir [4]. Ayrıca mobil robotların kontrollerinde robotların kinematik ve dinamik kontrolü de başka bir çalışma alanıdır [5,6]. Ayrıca mobil robotların gitmesi istenilen rotalarının hesaplanması üzerinde de birçok çalışma yapılmıştır [7]. Bu hesaplama işlemi her adımda gerçek zamanlı olarak değiştirilebilir [8]. Ayrıca bu belirlenen rotadaki gidilecek yolun optimizasyonu da farklı bir çalışma alanı olarak karşımıza çıkmaktadır [9]. Gidilecek yolların şekilleri üzerinde de yapılmış çalışmalar literatürde bulunmaktadır [10]. Yörünge takibi de mobil robotlarda sık çalışılan bir konudur [11,12]. Otonom araçların park etmesi de çalışılan alanlardan biridir [13].

1.1.1. Otonom araçların kullanım alanları

Otonom araçlar genel amaçlı birçok alanda kullanabildiği gibi özel amaçlı olarak askeri uygulamalarda savunma sanayisinde kullanılabilmektedir. Özellikle otonom hava araçları sınır güvenliğinin sağlanmasında kullanılmaktadır.

1.1.1.1. Otonom hava araçları

II. Dünya Savaşı sırasında kullanılan B-17'ler askeri amaçlı kullanılan ilk insansız hava araçları olarak tarihe geçmişlerdir. Daha sonra 1964-1967 yılları arasında Vietnam savaşı sırasında kullanılan insansız keşif dronları düşman hava sahasına tam 3400 sorti yapmışlardır. Türkiye 1993 yılında bir adet GNAT-750 sistemini deneme amaçlı olarak almış olup bu sistem hala kullanımdadır. Şekil 1.1'de bu otonom sistem gösterilmiştir.



Şekil 1.1: Gnat-750 otonom hava aracı

Bir otonom hava aracından tasarımında kritik hususlar şunlardır.

- GÜDÜM sistemlerinin hassasiyeti ve güvenilirliği
- Haberleşme sisteminin güvenilirliği
- Atmosferik ve Elektromanyetik çevre koşullarına uyum
- Kendini koruma yöntemleri

Otonom hava araçlarında da rota planlanması önemli bir çalışma alanıdır [14]. Ayrıca yine bu sistemler için hedef takibi de önemlidir [15]. Görüntü işleme tabanlı kontrol de otonom hava araçlarında kullanılmaya başlanmıştır [16, 17].

1.1.1.2. Otonom kara araçları

Otonom kara araçları genellikle askeri amaçlı kullanılan otonom hava araçlarına göre daha çok sivil amaçlı kullanılmaktadır. Bu konuda özellikle son yıllarda yapılan çalışmalar ile yakında otonom araçlar insanlara şehirlerarası yollarda hizmet vermeye başlayacaktır. Hava araçları kadar sık olmasa da otonom kara araçları da askeri amaçlı kullanılmaktadır.

Google özellikle yaptığı arabalarla otonom araçların en büyük destekçisi konumuna gelmiştir [18]. Google tarafından yapılan araba Şekil 1.2’de gösterilmiştir.



Şekil 1.2: Google'un otonom arabası [18]

Bu araba modifiye edilmiş bir Toyota Prius'dur. Aracın üstünde bir adet lazer, bir adet video kamera ve mesafe sensörleri bulunmaktadır. Araç Google'un haritasını kullanarak istenen hedefe otonom olarak gitmektedir.

Otonom kara araçları askeri amaçlı da kullanılabilirler. Şekil 1.3'de ASELSAN'ın ürettiği İZCİ arabası gösterilmektedir [19].



Şekil 1. 3: ASELSAN'ın İZCİ arabası [19]

1.2. Şerit Tespiti

Otonom araçlar fikri ilk ortaya atıldığından beri şerit tespiti özellikle bu konu için önemli hale geldiğinden, görüntü işleme tabanlı şerit tespiti için çalışmalar gittikçe artmaktadır. Görüntü işleme tabanlı şerit tespiti için birçok algoritma ve yöntem geliştirilmiş olup genellikle karşılaşılan sorunlardan bazıları şunlardır.

- Şerit çizgilerin kalınlığı
- Şerit çizgilerin birbirinden olan uzaklığı
- Ortamdaki ışık miktarının yetersiz ya da çok oluşu
- Şerit çizgilerinin kesikli çizgilerden oluşması
- Ortamda var olan gürültüler, gölgeler
- Algoritma için gereken işlem yükü ve hafıza ihtiyacı

Geliştirilen algoritmalarda en sık yaşanan sorun işlem yükü fazla olduğu için şerit takibinin gerçek zamanlı olarak çalışması olarak gösterilebilir. Algoritmaları hızlandırmak ve işlem yükünü hafifletmek için birçok çalışma yapılmasına rağmen gerçek zamanlı şerit tespiti hala güncel bir sorun olarak karşımıza çıkmaktadır.

1.2.1. Şerit tespiti için kullanılan yöntemler

Bu bölümde şerit tespiti için bir literatür taraması yapılmış olup, literatürde bulunan algoritmalarından bazıları temel olarak açıklanmış ve bu algoritmalarla ilgili makaleler referans verilmiştir.

1.2.1.1. İlgilenilen bölge için rastgele Hough dönüşümü

Wang ve arkadaşları yayımladıkları makalelerinde standart Hough dönüşümünden farklı bir yöntem kullanmışlardır [20]. Öncelikli olarak alınan orijinal imge üzerinde ön işlemleri yapmışlardır. Daha sonra imge üzerinde histgoram eşitleme kullanılarak imgedeki gri seviye dağılımı daha düzgün hale getirilmiştir. Histogram eşitlemesi yapılan imge üzerinde yatay ve düşey Sobel kenar filtreleri uygulanmış olup imge Hough dönüşümü için uygun hale getirilmiştir. Son olarak imge üzerinde Hough dönüşümü alınarak yol üzerindeki şerit çizgilerinin olduğu bölge bulunmuştur. Şerit çizgileri ile bulunan bölge bir sonraki imge için taranılacak bölge olarak seçilmiş ve sonraki imgelerde sadece bu bölge taranılarak Hough dönüşümünün hesapsal yükü azaltılmıştır.

1.2.1.2. Farklı Çözünürlüklerdeki Hough Dönüşümü

Yu ve Jain yayımladıkları makalelerinde standart Hough dönüşümünden farklı bir yöntem kullanmışlardır [21]. Yol modelini $y=mx+b$ denklemi yerine bir parabolik bir denklem kullanmışlardır. Normalde birbirine paralel olan şerit çizgilerinin perspektif hatası yüzünden imge üzerindeki görünüşünün paraboliksel bir görünümde olduğunu varsayarak bu parabolik denklemi yol modeli olarak kabul etmişlerdir. Fakat eğer denklem bu şekilde değiştirilirse önceden iki boyutlu olan Hough dönüşümünün akümülatör matrisi artık üç boyutlu hale gelir. Bu üç boyutun getirdiği hesapsal yükü azaltmak için k , v parametrelerini iki boyutlu farklı bir uzay β parametresini de farklı bir uzay olarak değerlendirmiştir.

İlk önce imge canny kenar filtresinden geçirilip imgedeki kenar bilgileri elde edilmiştir. Daha sonra birçok farklı çözünürlükte Hough dönüşümü alınarak imge üzerindeki şerit çizgileri bulunmuştur. Yazarlar yaptıkları deneysel çalışmalarda algoritmanın özellikle gürültülü imgelerde (gölge, ışık miktarının az oluşu) başarılı bir biçimde çalıştığı gözlenmiştir.

1.2.1.3. Yönlendirilebilir filtre tabanlı şerit takibi

McCall ve Trivedi yayımladıkları makalelerinde yönlendirilebilir filtre kullanmışlardır [22]. Geliştirdikleri sisteme makalenin İngilizcesinin baş harflerinden oluşan VioLET ismini vermişlerdir. Yönlendirilebilir filtrelerin imgelerdeki daire, şerit ve kesikli şeritleri bulabileceği gibi yoldaki ışık şiddetinin değişimi, gölgeler gibi bozucu etkilerden de az etkilenmesi nedeniyle yazarlar bu yöntemin şerit takibi için uygun bir yöntem olduğunu düşünmüşlerdir. Üç farklı filtre ile imgenin konvolüsyon toplamı kullanılarak imgedeki şerit bilgileri elde edilmiştir. Yazarlar ilk önce parabolik bir yol modeli oluşturmuşlardır. Daha sonra imgenin ikinci türevine bağlı olarak iki boyutlu Gauss filtresini imge üzerine uygulayıp imge üzerindeki şeritlerin özelliklerinin çıkartıp, şerit çizgileriyle oluşan parabolün sınırlarını belirlemişlerdir. Ayrıca aracın konumu da yazarlar için önemli bir veri olup şerit takibi sırasında aracın pozisyonunu da korumasını sağlamışlardır. Pozisyon takibi işleminde ise Kalman filtresinden faydalanmışlardır.

Yazarlar yaptıkları sistemi San Diego’da 5 km’lik bir alanda denemişlerdir ve çalışma sonucunun başarılı olduğunu görmüşlerdir.

1.2.1.4. Dinamik programlama yöntemi

Kang ve Jung yayımladıkları makalelerinde şerit takibi için dinamik programlama ve yerel çizgi çıkartımı yöntemlerini kullanmışlardır [23]. Yerel çizgi çıkartımı yöntemi, imge üzerindeki kenar bilgilerine dayanarak şerit bilgisinin başlangıç pozisyonunu elde eder ardından dinamik programlama bu bulunan başlangıç pozisyonunu geliştirerek şerit çizgilerinin daha doğru olarak bulunmasını sağlar.

İlk önce alınan imge dikey doğrultuya göre alt bölgelere bölündükten sonra çizgi çıkartımı ile alt bölgelerdeki şerit çizgisi olmaya aday çizgiler elde edilir. Daha sonra aday çizgiler arasından en baskın olanlar, doğruların türevlerinin değerlerinin bulan dinamik programlama yöntemi ile bulunur. Yazarlar yaptıkları çalışma sırasında özellikle dinamik programlamanın hesapsal yükü önemli bir miktarda azalttığını görmüşlerdir.

1.2.1.5. B-Snake yöntemi

Wang ve arkadaşları yayımladıkları makalelerinde şerit takibi için B-Snake yöntemini kullanmışlardır [24]. B-Snake yöntemi temelde basit bir B-Spline yöntemini kullanarak kontrol noktalarını kapsayan kapalı bir şekil oluşturur. Ayrıca bu yöntemde yoldaki şerit çizgilerinin bulunması problemi, perspektif etkisinden dolayı şerit çizgilerinin imgenin üst kısmında bir noktada(birleşme noktası) kesişeceği için, bu noktanın bulunması problemi ile birleştirilir. Yolun orta noktası bulunduktan sonra diğer çizgiler birbirine paralel olarak bulunabilir. B-Snake’in başlangıç pozisyonunu belirleyebilmek için yazarlar CHEVP adını verdikleri bir algoritma geliştirmişlerdir. Geliştirdikleri bu algorithmada yazarlar öncelikli olarak alınan yol görüntüsünü canny kenar bulma filtresinden geçirerek görüntüdeki kenar noktalarını elde edip daha sonra bu görüntü üzerindeki doğruları bulabilmek için görüntü üzerinde Hough dönüşümü uygulamışlardır. Daha sonra birleşme noktalarını belirlemek için imge yatay olarak farklı alt bölgelere bölünmüştür. Bu alt bölgelerdeki doğruları bulabilmek için alt bölgeler için yeni bir Hough uzayı

oluşturur. Bu uzaydaki doğru çiftleri eşleştirilerek en yüksek puanlı noktalar elde edilir. Doğru çiftlerinin puanı hesaplanırken bir önceki adımda elde edilen Hough dizisinin değerleri kullanılır. Bu işlem imgedeki bütün alt bölgeler için tekrarlanır ve bulunan maksimum değeri bölgelerin birleşme noktası olarak kabul edilir. Daha sonra birleşme noktalarına gradiyent vektör takibi yöntemi uygulanır ve en küçük karesel hata metodu kullanılarak şerit çizgilerinin parametreleri elde edilir.

Yazarlar bu yöntemi laboratuvar ortamında önceden çekilen 50 farklı yol imgesi üzerinde uygulamış olup algoritmanın gürültülü, farklı ışık şiddeti olan, gölgelerin olduğu yolda başarılı bir biçimde çalıştığını gözlemişlerdir.

1.2.1.6. SVM ile şerit değişiminin bulunması

Mandalia ve Salvucci yaptıkları çalışmada şerit değişimini algılamak için genellikle sınıflandırma problemi için kullanılan SVM (Destek vektör makinaları) yöntemini kullanmışlardır [25]. Yazarlar SVM'in eğitim verisi olarak aracın ivme bilgisi ve dönme açısı başta olmak üzere birçok veri kullanmışlardır. SVM'in eğitiminde kullanılan özelliğin seçimi SVM eğitimi içinde en önemli işlem olduğu için şerit değişimi sırasında en çok değişen özelliğin kullanılması çalışmada önemli bir parça olmuştur. Yazarlar öndeki yol gösteren aracın hızı ve far hareketlerini SVM'in daha doğru sonuçlar elde etmesi için kullanılabilecek özellikler olarak belirlemişlerdir.

Yazarların geliştirdikleri sistem başlangıçtaki manevradan itibaren 0.3 saniye içerisinde %87 oranında başarılı sonuç elde etmiştir. Aynı şekilde manevradan 1.5 saniye sonra %97 oranında başarılı sonuç elde etmiştir.

1.2.1.7. Kalman filtresi kullanılarak şerit tespiti

Borkar ve arkadaşları yaptıkları çalışmada yoldaki şerit çizgilerini algılamak için RANSAC genetik algoritması ile beraber Kalman filtresini kullanmışlardır [26]. Yazarlar yaptıkları çalışmada öncelikli görüntüdeki perspektif etkisini yok etmek için ters perspektif dönüşümü uygulayıp görüntüdeki perspektif etkisinin yok etmişlerdir. Daha sonra görüntüyü binary (ikili) hale getirmek için, görüntü üzerinde Canny kenar bulma filtresini uygulamışlardır. Daha sonra ikili hale getirilen bu görüntü

tahminen şerit çizgilerini kapsayan iki bölüme ayrılır. Elde edilen bu bölümler üzerinde düşük çözünürlükte Hough dönüşümü uygulanarak Hough matrisindeki değeri en büyük olan 10 şerit çizgileri arasındaki aday çizgiler olarak seçilir. Elde edilen bu aday çizgilerin orta noktasını bulabilmek için tek boyutlu bir Gauss filtresi bütün aday çizgilere uygulanır ve bu aday çizgilerinin arasında korelasyonu belirli bir eşiğin üstünde olan değerler seçilir. Kalanların aday çizgilere RANSAC (rastgele örnek uyuşması) algoritması uygulanarak bir yol modeli oluşturulmuştur. Son olarak da doğrusal en küçük kareler yöntemi kullanılarak şerit çizgileri bulunur. Son olarak konum vektörü p , Θ parametreleriyle $x(n)=[x(n) \ x'(n) \ \Theta(n) \ \Theta'(n)]$ olarak ifade edilerek şerit takibi için Kalman filtresi uygulanır.

Yazarlar çalışmayı yaparken bir kamera ile görüntüyü bilgisayar ortamına aktarıp görüntüyü MATLAB ortamında işleyip sonuçları elde etmişlerdir. Yazarlar yaptıkları çalışmada şehir yollarındaki değişen trafikte %76 oranında başarı yakalamışlardır.

1.2.1.8. İki kamera kullanarak engel tanıma ve şerit takibi

Nedevschi ve arkadaşları yaptıkları çalışmada iki kamera kullanarak engel ve şerit tespiti üzerinde çalışmışlardır [27]. Yazarlar aracın sağında ve solunda birbirine paralel olarak bulunan iki kameradan aldıkları iki görüntüyü işleyerek yoldaki engelleri ve şerit çizgilerini tespit etmişlerdir. Sol tarafta yer alan kamerayla elde ettikleri görüntüyü işleyerek şerit çizgisini, her iki kameradan elde ettikleri görüntüleri birleştirilmesi ile elde edilen görüntüyü işleyerek ise aracın önündeki boş alanı tespit etmişlerdir. Geliştirilen sistemde alınan görüntünün işlenmesiyle elde edilen görüntü ise bir ekran ile sürücüye gösterilmektedir.

Yazarlar öncelikli olarak şerit takibi için yoldaki şerit çizgilerinin varlığına dayanan bir görüntü eşleşmesi yöntemi kullanmışlardır. İlk önce alınan görüntüdeki perspektif etkileri perspektif dönüşümü kullanılarak yok edilmiştir. Daha sonra alınan görüntüdeki parlaklık etkisine bakılmıştır. Şerit çizgilerinin bulunduğu piksellerin parlaklık değerlerinin diğer piksel parlaklık değerlerinden daha büyük olması gerektiği yaklaşımına dayanarak yoldaki şerit çizgilerinin konumları tahmini olarak belirlenmiştir. Ayrıca bu parlaklık yaklaşımıyla görüntüyü ikili hale getirmek için kullanılan eşik değeri her defasında dinamik olarak yeniden hesaplanmıştır. Daha

sonra şerit çizgilerinin birbirine paralel olması gerektiği bilgisi kullanılarak şerit çizgileri belirlenmiştir.

Yazarlar geliştirdikleri sistemi MOBLAB'ın otonom arabasında denemiştirler. Yazarlar bu araçla 80 km/s hızla 3000 km'den fazla yolda denemiş olup başarılı sonuçlar elde etmişlerdir.

1.2.1.9. Catmull-Rom spline kullanarak şerit takibi

Wang ve arkadaşları yaptıkları çalışmada Spline kullanarak şerit tespiti üzerinde çalışmışlardır [28]. B-Spline yöntemini kontrol noktalarını kapsayan rastgele kapalı bir şekil oluşturduğundan dolayı diğer yöntemlere göre şekillerin bulunmasında daha etkili bir yöntemdir. Bu yöntemde şerit çizgilerinin bulunması problemi modeldeki kontrol noktalarının bulunması problemi olarak değişmektedir. Şerit çizgilerinin bulunması için kullanılan algoritma ise maksimum benzerlik algoritmasıdır.

Bu yöntemde amaç yol modelinin oluşturmak için kontrol noktalarının elde edilmesidir. Geliştirilen yol modeli görüntü üzerinde aranmadan önce görüntü Canny kenar bulma filtresinden geçirilerek görüntü üzerinde kenar noktası olan pikseller elde edilir. Bu kenar pikselleri görüntü üzerinde kontrol noktalarını aranacağı alan olarak belirlenir. Kontrol noktaları aranacağı alan belirlendikten sonra yol modeli oluşturulur. Oluşturulan bu model elde edilen görüntü üzerine uygulanır ve maksimum benzerlik yaklaşımı kullanılarak görüntü üzerindeki şerit çizgileri aranır.

Yazarlar yaptıkları çalışmada resim için farklı çözünürlükler kullanarak işlem yükünü azaltıp algoritmanın çalışmasını hızlandırmışlardır. Ayrıca gölge olan yollarda başarılı sonuçlar elde etmişlerdir.

1.2.1.10. Renk modeli tabanlı şerit çizgilerinin bulunması

Sun ve arkadaşları yaptıkları çalışmada HSI renk modelini kullanarak şerit tespiti üzerinde çalışmışlardır [29]. İmgelerin gösteriminde birçok renk modeli kullanılmasına rağmen, özellikle HSI modelinde yer alan yoğunluk ve doyum değerleri, HSI modelinin şerit tespitinde daha doğru sonuçlar vermesini sağlamaktadır. Şerit çizgilerinin olduğu piksel değerleri RGB renk modelinde de

diğer piksel değerlerinden farklı olmasına rağmen HSI renk modelinde bu piksel değerleri diğer piksel değerlerinden belirgin şekilde farklıdır ve basit eşik değerleri kullanılarak belirlenebilirler.

Yazarlar ilk önce kameradan RGB formatında gelen görüntüyü ön işlem olarak HSI formatına çevirmişlerdir. Daha sonra işlemsel yükü azaltmak için imgedeki sadece belirli bir alanı (ROI) belirleyip ki burada görüntünün alt kısmıdır, bu da belirlenen kısımda şerit çizgilerini aramaktadırlar. Şerit çizgilerinin bulunduğu piksellerin I değeri diğer piksellerden daha farklı olduğu için I değeri ayırtaç olarak kullanmışlardır. I değerine bağlı olarak bir eşik oluşturup imgedeki ilgilenilen bölgeyi (ROI) bulanık c ortalama (Fuzzy C-Means) yöntemini kullanarak alt bölgelere ayırmışlardır. Daha sonra imge ikili hale getirilip bağlantılı bileşen etiketleme algoritması kullanılarak imge üzerindeki şekiller belirlenmiştir. Son olarak bu belirlenen şekiller arasından şerit çizgisi için gerekli genişlik uzunluk oranını sağlayanlar şerit çizgisi olarak kabul edilmiştir.

Yazarlar yaptıkları çalışmada RGB renk modeli yerine HSI renk modelinin kullanılmasının işlemsel yükü azalttığını ve farklı ışık şiddetlerinde de başarılı olarak çalıştığını belirlemişlerdir.

1.2.1.11. İki kamera kullanarak üç boyutlu şerit çizgilerinin bulunması

Nedevschi ve arkadaşları yaptıkları çalışmada iki kamera kullanarak üç boyutlu şerit tespiti üzerinde çalışmışlardır [30]. Görüntünün üç boyutlu bilgisi sayesinde yoldaki şerit çizgilerinin ve engellerin ayrılmasında kullanılabilir. Yazarlar, üç boyutlu görüntüde yatay ve düşey bilgiler kullanılarak, şerit çizgilerinin, çizgilerinin genişliği ve açılarına bağımlı bir eğri olarak ifade edilebilmesi özelliğini kullanmışlardır.

Şerit tespiti takip işlemi ile birleştirilmiştir. İlk anki şerit parametreleri, eski parametreler ve araç dinamikleri kullanılarak tahmin edilmiş olup, bu tahmin sayesinde bir sonraki görüntüde şerit çizgilerinin aranacağı alanlar belirlenmiştir. Yazarlar ilk önce düşeydeki üç boyutlu bilgileri kullanarak bir model oluşturup daha sonra da yataydaki üç boyutlu görüntüde bu model eşleme tekniğini kullanarak şerit

izgilerini tespit etmeye alıřmıřlardır. Saę ve sol řerit izgileri bulunduktan sonra yolun aısı hesaplanarak ve bu hesaplanan deęer Kalman filtresi kullanılarak gncellenir.

Yazarlar yaptıkları alıřmada bu teknięin dz yollarda doęru bir biimde alıřtığını gzlemlemiřlerdir.

1.2.1.12. İmgedeki řekillerin benzerlięi algoritması ile řerit tespiti

Kreucher ve arkadaşları yaptıkları alıřmada řerit izgilerini bulabilmek iin imgedeki řekillerin benzerlięi algoritmasını (LOIS) kullanmıřlardır [31]. Aracın bařlangı konumu LOIS algoritması ile bulunmuř ve bu konum bilgisi zamanın bir fonksiyonu řeklinde yazılmıřtır. Daha sonra bir Kalman filtresi uygulanarak aracın bir sonraki konumunun tahmini yapılmıřtır. LOIS kullanılarak aracın konumu bir metre hassasiyetle tespit edilmiř daha sonra Kalman filtresi kullanılarak bir saniyeden daha az bir srede bu hassasiyet 0.8 metreye ykseltilmiřtir.

Yazarlar LOIS algoritmasına gre ncelikli olarak imgedeki btn yolları kapsayan bir parabolik yol modeli oluřturmuřlardır. Daha sonra yol grnts zerinde benzerlik fonksiyonunu kullanarak yol zerinde řerit izgisini tespit etmeye alıřmıřlardır. Bu tespit ettikleri řerit bilgilerini daha sonraki grntlerde yol modeli olarak kullanmıřlardır. Bunu yapabilmek iin bu nceki bilgileri bir Bayes atısı ile birleřtirip yol řeklinin kestirimi iin MAP algoritmasını kullanmıřlardır.

Yazarlar geliřtirdikleri sistemi 1400 farklı grnt zerinde denemiř ve bařarılı sonular elde etmiřlerdir.

1.2.1.13. Oryantasyon ve uzunluk azaltımı ile řerit tespiti

Lai ve Yung yaptıkları alıřmada oryantasyon ve uzunluk azaltımını kullanarak řerit tespiti zerinde alıřmıřlardır [32]. Yazarlar geliřtirdikleri algoritmada btn yol bilgisinin oryantasyon ve řerit izgilerinin uzunluk bilgisini kullanarak elde etmiřlerdir. Yazarlar ilk nce kenar bulma yntemlerinin kullanıp kenar bilgisini elde edip bu kenar bilgilerine gre imgedeki dz izgileri elde etmiřlerdir. Daha sonra bu oryantasyon ve uzunluk bilgileri kullanılarak iki boyutlu imgeden  boyutlu imgeye

geçip, burada k-ortalama sınıflandırmasını kullanarak görüntü üzerinde şerit çizgilerinin bulmuşlardır. Kullanılan k-ortalama algoritması sayesinde yüksek oryantasyonu ve yeterli uzunluğu olmayan çizgiler şerit çizgisi olamayacak çizgiler olarak kabul edilmiş ve elenmiştir.

Yazarlar geliştirdikleri sistemin her pratikte başarılı çalışma, iki şeridi de tespit etme, sadece bir görüntüye ihtiyacı olması gibi birçok avantajı olduğunu belirtmişlerdir.

1.2.1.14. Yapay Sinir Ağları Kullanarak Şerit Tespiti

Jochem ve arkadaşları yaptıkları çalışmada yapay sinir ağları kullanarak şerit tespiti üzerinde çalışmışlardır [33]. Yazarlar basit ön görüntü işleme yöntemlerini kullanarak elde ettikleri görüntüyü giriş olarak oluşturdukları üç katmanlı bir yapay sinir ağına vermişlerdir. Yazarların oluşturdukları yapay sinir ağı (ALVINN) 960 giriş ünitesi, dört gizli katman ve 50 çıkış ünitesinden oluşmaktadır ve öğrenmek için geri yayılım algoritmasını kullanmaktadır. Giriş olarak kameradan alınan görüntü varken çıkışta araca hangi yöne kaç derece açı ile yöneleceğinin bilgisi verilmektedir.

Yazarlar geliştirdikleri bu yapay sinir ağlarını birleştirerek MANIAC adını verdikleri çoklu bir ağ sistemi oluşturmuşlardır. MANIAC sistemi 30x32 adet girişten ve 8 adet gizli katmandan oluşmaktadır.

Yazarlar MANIAC sistemini oluşturup eğitimini tamamladıktan sonra elde ettikleri sonuçların hataları ile ALVINN sistemi ile karşılaştırmışlardır. Özellikle dört şerit bulunan yollarda MANIAC sisteminin ALVINN sistemine göre daha az hata yaptığını gözlemlemişlerdir.

1.2.2. Tezin Literatüre Katkısı

Literatürde yapılan çalışmaların birçoğu görüntüleri bir kamera ile kaydedip görüntüleri bilgisayar ortamına aktarıp, görüntü işleme kısmını ise MATLAB ortamında yapmışlardır. Görüntünün işlenmesi sırasında MATLAB ortamı kullanılınca PC veya dizüstü bilgisayar kullanıldığından dolayı görüntünün işlendiği işlemcinin ve hafızanın hızı artmaktadır. Bu tez çalışması sırasında Visual DSP

ortamında geliştirilen kod, bir ADSPBF561 işlemcisi üzerinde çalıştırıldığı için, işlemcinin ve hafızanın hızı ortalama bir PC'ye göre bile çok daha az olmaktadır. Bu yüzden bu çalışmada kodun gerçek zamanlı olarak çalışabilmesi için birçok optimizasyon uygulanmıştır. Ayrıca tez çalışması sırasında bir mobil robot platformu geliştirilmiş olup, geliştirilen metotlar gerçek zamanlı olarak bu mobil robot üzerinde gerçekleştirilmiştir.

Geliştiren mobil robot platformu, mobil robotun hareketini sağlayan motorların limit değerlerinden dolayı maksimum 4 km/h hızla hareket etmektedir. Buna göre mobil robotun saniyede aldığı yol ise 1.11 m'dir. Mobil robotun görüntü işlemeyle gerçek zamanlı olarak yönlendirilip 80 cm genişliğindeki şerit çizgilerinin dışına çıkmaması için, mobil robotun manevra kabiliyeti dikkate alındığında saniyede yaklaşık olarak 5 görüntü işlemesi gerekmektedir. Geliştirilen görüntü işleme kodu yapılan optimizasyonlar sayesinde saniyede 30 görüntü işleyebilecek hale getirilerek, kodun gerçek zamanlı olarak çalışması ve çok daha yüksek hızda hareket eden mobil robotlar veya araçlar için de kullanılabilmesi sağlanmıştır.

2. GÖRÜNTÜ İŞLEME TABANLI ŞERİT TESPİTİ ALGORİTMASI

2.1. Giriş

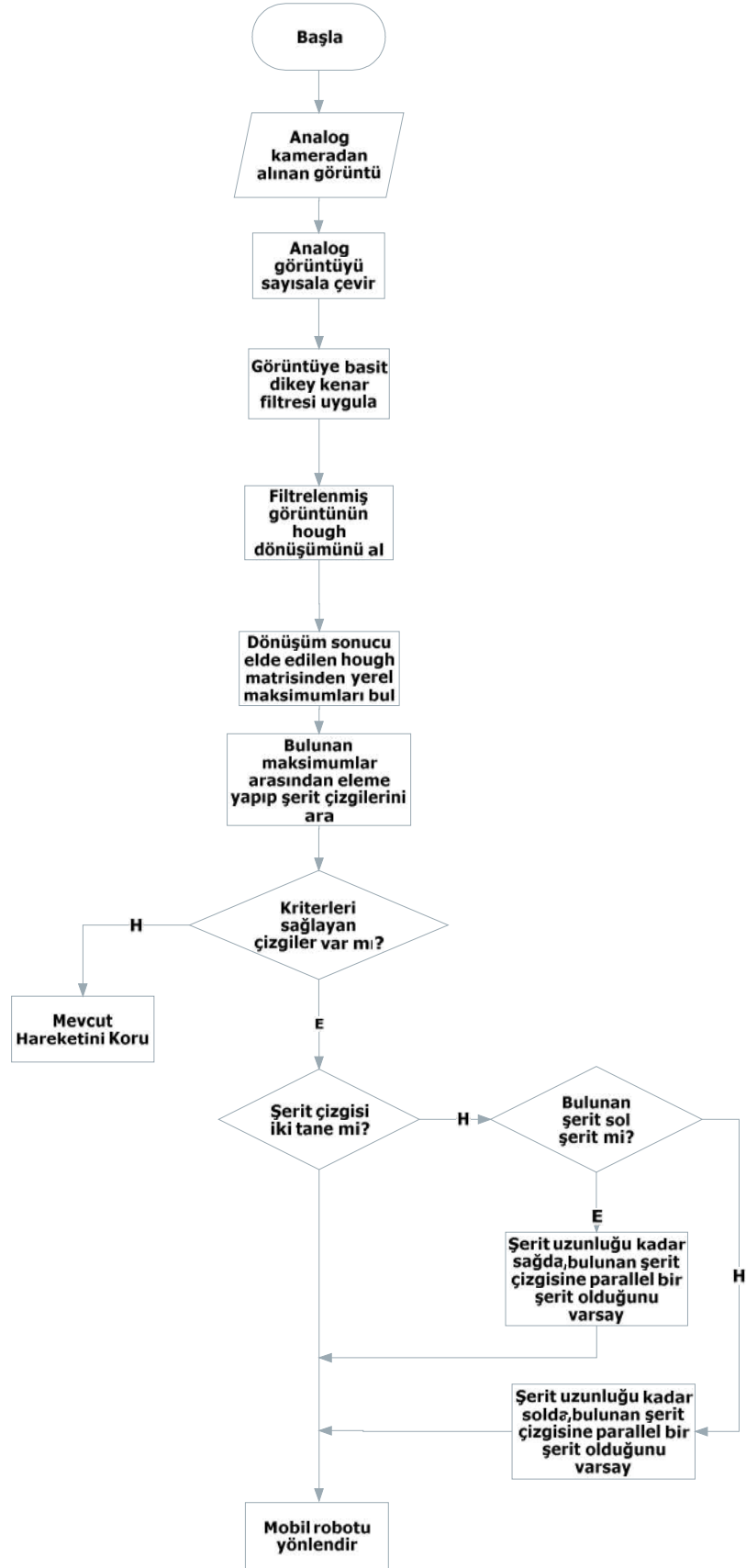
Bu bölümde, tez çalışmasının konusu olan otonom hareket eden mobil robotun şerit dışına çıkmamasını amaçlayan görüntü işleme algoritması ve bu algoritmanın görüntü işleme kartı üzerinde gerçekleştirilmesi anlatılacaktır. Algoritmanın mantığı, gerçek zamanlı olarak gerçekleştirilmesi için yapılan optimizasyonlar ve algoritmanın karşılan sorunlara karşı nasıl davrandığı teker teker ele alınmış olup, bu durumlar tezin bu kısmında ayrıntılı olarak açıklanmıştır. İlk olarak görüntü işleme kartı, kartın üzerinde yer alan BF-561 işlemcisi ve bu karta görüntü verisinin alınması ele alınmıştır. Kod geliştirme ortamı olan Visual DSP 5.0 ortamından da kısaca bahsedilmiş olup bu ortamda proje geliştirmek için temel aşamalar açıklanmıştır.

Tez çalışması sırasında, şerit tespiti kodu ilk olarak MATLAB ve C# ortamlarında geliştirilmiş olup daha sonra aynı kod C ortamında geliştirilmiştir. İlk olarak kodun çalışma mantığı ve algoritmada kullanılan yöntemler hakkında temel bilgiler verilecektir. Ayrıca C ortamında bu kodun optimizasyonu üzerinde çalışılmış olup başlangıçta saniyede bir görüntü işleyebilen kod saniyede yirmi beş görüntü işleyebilecek hale getirilmiştir. Yapılan bu optimizasyon aşamaları tezin bu kısmında detaylı olarak ele alınacaktır.

2.2. Algoritma

Tez çalışması şerit tespiti için kullanılan algoritma ve geliştirilen C kodu tezin bu bölümünde ayrıntılı olarak anlatılmıştır. İlk olarak görüntü ADSP BF-561 görüntü işleme kartına bir analog kamera ile alınmıştır. Alınan bu görüntü bilgisi kart üzerindeki DMA yardımıyla ilgili değişkenlere aktarılmıştır. Sonrasında alınan bu görüntü kenar bulma filtresinden geçirilerek görüntüdeki kenarlar bulunmuş, daha sonra da görüntüdeki şeritleri bulabilmek için görüntüye Hough dönüşümü uygulanmıştır. Bu dönüşüm sonucunda elde edilen Hough matrisinin maksimumlarına şerit olabilmesi için gerekli eliminasyonlar uygulanmıştır ve kriterlere uygun olmayanlar aralarından elenmiştir. Bu elemeler sonucunda kalanlar şerit çizgisi olarak kabul edilmiş ve yolun açısı hesaplanarak mobil robotun navigasyonu sağlanmıştır. Algoritmanın değişik koşullar altında elde ettiği sonuçlar 4. Bölümde verilecektir.

Algoritmanın genel akış şeması Şekil 2.1’de gösterilmiş olup her bir adım ayrıntılı olarak anlatılacaktır.



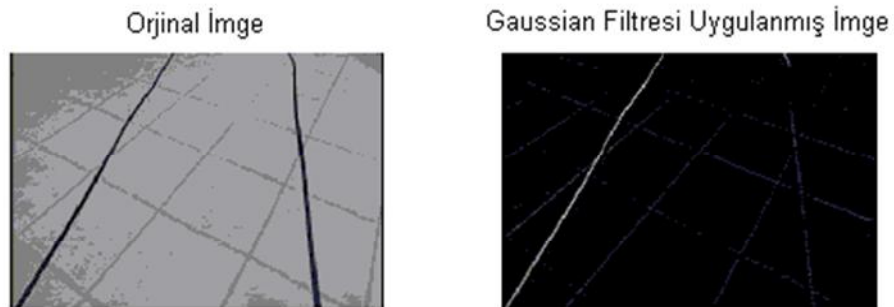
Şekil 2.1: Algoritmanın genel akış şeması

2.2.1. Filtreleme

Filtreleme işlemi görüntü işlemede gürültü azaltımı, kenar tespiti gibi farklı konularda çok sık kullanılan tekniklerden biridir. Filtreleme işleminde, bir kernel (çekirdek) matrisi, görüntü matrisi üzerinde dolaştırılarak, üst üste gelen değerler çarpılarak toplanır ve bulunan sonuç filtre toplamına bölünerek yeni piksel değerleri hesaplanır. Her seferinde merkez piksel değeri değiştirilerek filtreleme işlemi yapılır. Genellikle 3x3'lük olan bu kernel (çekirdek) matrisi filtrenin tipini belirler.

Filtreleme işlemi sırasında temel olarak karşılaşılabilecek iki tane temel sorun vardır. Bunlardan biri çarpma, toplama, bölme işlemleri sırasında elde edilen değerlerin maksimum sınırları geçmesi sorunudur ki bu sorun sonuçların maksimum veya minimum değerlerine geri çekilmesi ile çözülebilir. Diğer bir sorun ise filtre matrisinin görüntü dışına taşması sorunudur ki bu sorun kenar piksellerden uzak durularak çözülebilir.

Tez çalışması sırasında şerit tespiti yapılacağı için kenar bulma filtrelerinden basit düşey kenar bulma filtresi kullanılmıştır. İlerideki bölümde kenar bulma filtreleri hakkında temel bilgiler verilecek ve tezde kullanılan sobel kenar bulma filtresi detaylı olarak açıklanmıştır. Şekil 2.2'de filtrelenmiş bir resim görülmektedir.



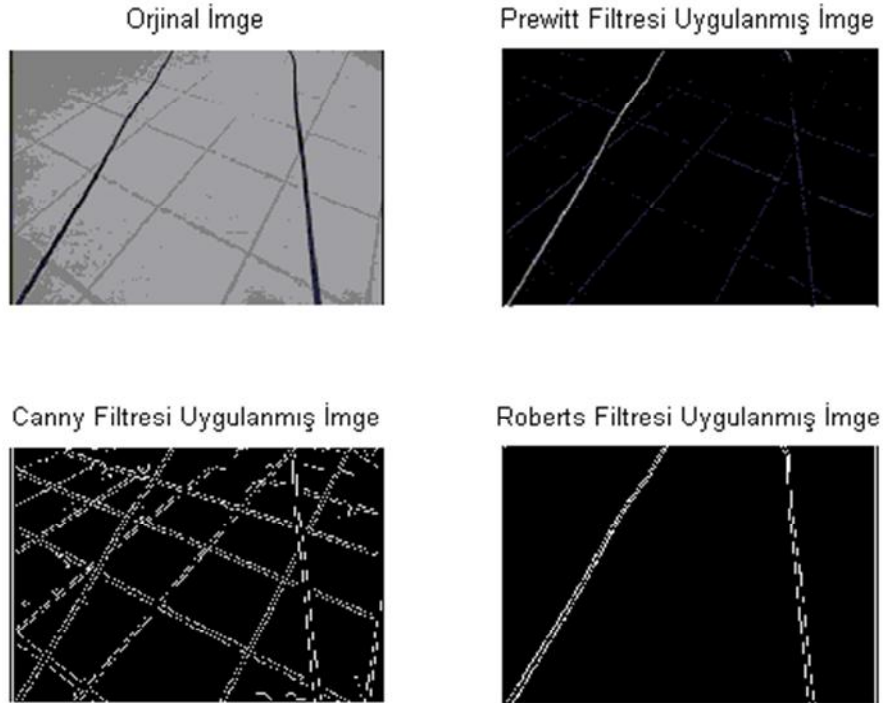
Şekil 2.2: Örnek imge ve filtreden geçirilmiş hali

Filtreleme işlemi görüntü işlemede ön işlem olarak çok sık kullanılan bir tekniktir. Görüntü işlemede kullanılabilecek birçok filtre arasından istenilen kriterlere en uygununun seçilmesi gerekmektedir [34].

2.2.2. Kenar bulma filtreleri

Kenar bulmak için birçok filtre olmasına rağmen, bu filtreleri temel olarak kullandıkları yöntemler açısından gradiyent ve Laplace yöntemlerini kullanan filtreler olmak üzere ikiye ayırmak mümkündür. Gradyent yöntemini kullanan filtreler, kenar bilgisini, imgenin birinci türevinin maksimum ve minimum noktalarını bularak elde ettikleri için, birinci derece filtreler olarak isimlendirilirler. Laplace yöntemini kullanan filtreler ise, kenar bilgisini, imgenin ikinci türevinin sıfır geçişlerini bularak elde ettikleri için, ikinci derece filtreler olarak isimlendirilirler. Her iki filtre türü imgedeki yatay veya düşey kenarları bulmak için kullanılabilirler fakat yatay ve düşey kenarlar için oluşan kernel (çekirdek) matris elemanları birbirinden farklıdır.

İmge işlemede en sık kullanılan birinci derece kenar bulma filtreleri Prewitt ve Sobel kenar bulma filtreleridir. Şekil 2.3’de çeşitli uygulama sırasında elde edilen görüntülerden kenar bulma filtrelerinden geçirilmiş imgeler gösterilmiştir.



Şekil 2.3: Örnek kenar bulma filtreleri sonuçları

2.2.2.1. Sobel kenar bulma filtresi

Tez çalışması sırasında kenar bulma filtrelerinden basit düşey kenar bulma filtresi kullanılmış olup bu bölümde sobel kenar filtresi hakkında temel bilgi verilmiştir. Sobel filtresinin kenar bulma algoritmalarında çok sık şekilde kullanılabilir [35,36].

Bir önceki bölümde de bahsedildiği gibi Sobel kenar filtresi gradiyent yöntemini kullanan birinci derece bir filtredir. Temelde yatay ve dikey olmak üzere 3x3'lük iki farklı Sobel kernel tipi vardır.

Dikey Sobel filtresi kernel matrisi aşağıdaki gibidir;

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

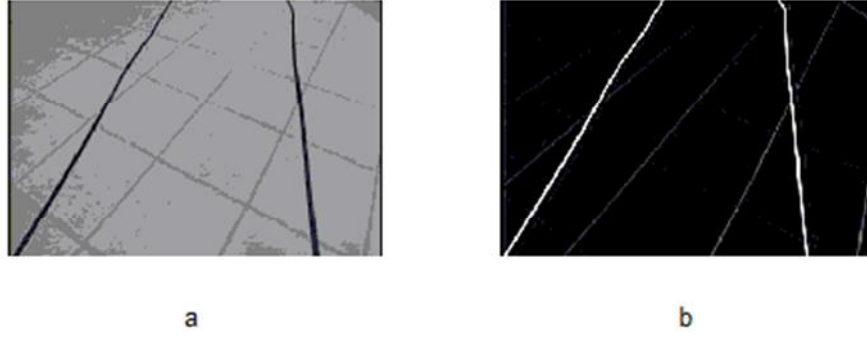
Yatay Sobel filtresi kernel matrisi aşağıdaki gibidir;

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel filtresinin kullanılmasının bazı sebepleri şunlardır.

- Birinci derece filtre olduğu için gürültüden az etkilenmesi
- Kolaylıkla uygulanabilir olması
- Kenar çizgilerini Canny, Roberts gibi diğer filtrelere göre daha kalın bulması

Şekil 2.4'de dikey Sobel filtresinin kullanılmasına örnek verilmiştir.



Şekil 2.4: a) Orijinal imge b)Dikey Sobel kenar filtresi uygulanmış imge

2.2.3. Hough dönüşümü

Hough dönüşümü 1972 yılında Richard Duda ve Peter Hart tarafından bulunmuş olmasına rağmen bu teknik 1981 yılında Dana H.Ballard tarafından yayımlanan makale ile görüntü işlemede sıkça kullanılmaya başlanmıştır [37]. Başlangıçta sadece imgedeki doğruları bulmak için kullanılan Hough dönüşümünün, bu makale ile özellikle elips ve daireler başta olmak üzere birçok şeklin algılanması için kullanılabileceği anlaşılmıştır [38,39]. Ayrıca Hough dönüşümü ile imgedeki şekillerin birleşim noktaları bulunabilir [40]. Hough dönüşümünün uygulanabilirliği ve performansı da üzerinde araştırma yapılan alanlardan biridir [41,42].

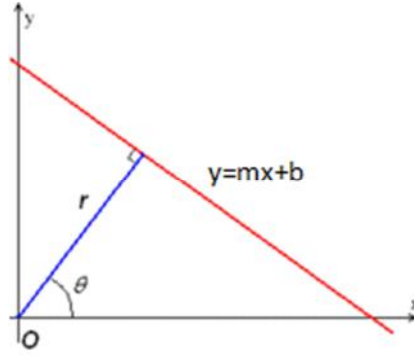
Hough dönüşümünü imgedeki gürültülerden ve boşluklardan çok az etkilenmesine rağmen ki bu durum Hough dönüşümünün en büyük avantajlarından biri olarak gösterilebilir, Hough dönüşümünde işlem yükü çok fazladır. Hough dönüşümünün en büyük dezavantajlarından biri olan bu işlem yükü çeşitli optimizasyon yöntemleri kullanılarak azaltılabilir. Bu hesapsal yük Hough yönteminin gerçek zamanlı şerit tespitinde kullanılmasındaki en büyük zorluk olarak karşımıza çıkmaktadır. Tez çalışması sırasında mobil robotun hareketini sağlamak için şerit tespiti işleminin gerçek zamanlı olarak yapılması gerekmektedir. Tez çalışması sırasında yapılan optimizasyon çalışmaları optimizasyon başlığı altında ele alınacaktır.

Hough dönüşümü temel olarak imge uzayında aranılan şeklin matematiksel ifadesinden yola çıkarak bir akümülatör matrisi oluşturulup, bu oluşturulan matrisin maksimumlarının bulunmasına dayanmaktadır. Akümülatör matrisinin boyutlarını

aranılan şeklin denkleminde kullanılan parametrelerin boyutları belirler. Bu parametreler bir doğru için orijin (başlangıç) noktasından olan uzaklık ve açı değeri iken bir daire veya elips için yarıçap değeri ve merkez noktasının koordinat değerleridir. Hough dönüşümünün hesapsal yükü bu akümülatör matrisinin boyutları ile ilgilidir. Akümülatör matrisinin elemanları ne kadar çok olursa o elemanlar içerisinde maksimumları bulmak ve bulunan değerleri işlemek o derece zaman alacaktır. Bu da algoritmanın hesapsal yükü olarak karşımıza çıkmaktadır. Hesapsal yükü azaltmak için akümülatör matrisinin eleman sayısını azaltmamız gerekir fakat bunu yapmak için çözünürlüğü düşürmemiz gerekir ki bu da algoritmayı olumsuz yönde etkileyip algoritmanın bulacağı sonuçların yanılmasına neden olabilir. Bu yüzden kullanılacak dönüşümün parametrelerinin algoritmanın başarımını sağlayacak minimum çözünürlükte olması gerekir.

2.2.4. Hough dönüşümü ile doğru tespiti

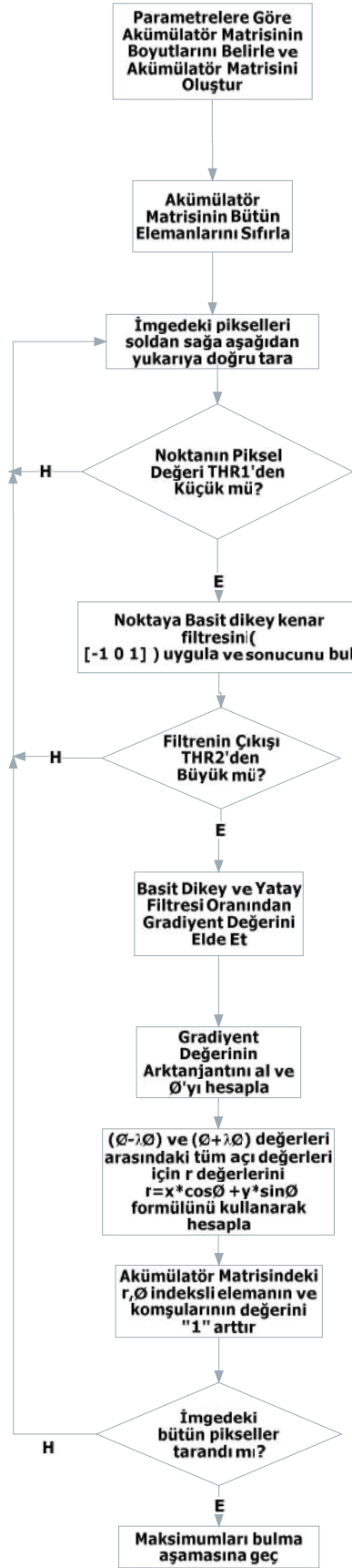
x, y uzayında bir doğru denklemi herkesin bildiği gibi $y=mx+b$ şeklinde ifade edilir. Bu denklemdaki m parametresi doğrunun eğimini verirken b parametresi ise doğrunun y eksenini kestiği noktayı bize vermektedir. Hough dönüşümün amacı bu doğruyu x, y uzayından çıkartıp m, b parametre uzayında göstermektir. Fakat doğrunun eğiminin alabileceği sıfır veya sonsuz değerleri hesaplamalarda büyük karmaşıklıklar çıkartabileceği için doğru denklemi $r = x\cos(\Theta) + y\sin(\Theta)$ şeklinde değiştirilerek yazılabilir. Buradaki r parametresi doğrunun orijin (başlangıç) noktasından olan dik uzaklığı belirtirken Θ parametresi ise orijin (başlangıç) noktasının doğruya olan dik uzaklığının açısını vermektedir. Doğrunun bu iki parametre ile gösterimi kutupsal gösterim olarak da bilinmektedir. Bu sayede hesaplamalarda karşılaşılan zorlukların üzerinden gelinmiştir. Şekil 2.5’de bu parametre dönüşümü kolaylıkla anlaşılabilir.



Şekil 2.5: r ve θ parametre uzayı

Hough dönüşümü bu denkleme göre yapılırsa, dönüşümün çözünürlüğü θ değerinin 0 ile 2π arasındaki adım aralığı olarak düşünülebilir. Bu çözünürlük ne kadar artarsa r ve θ değerlerinden oluşan akümülatör matrisinin boyutları da o oranda artacağı için işlem yükü de artacaktır. Tez çalışması sırasında yapılan Hough dönüşümünde θ aralığı -85 ile $+85$ değerleri arasında değişmektedir ve 1 derece çözünürlükle Hough dönüşümü alınmaktadır.

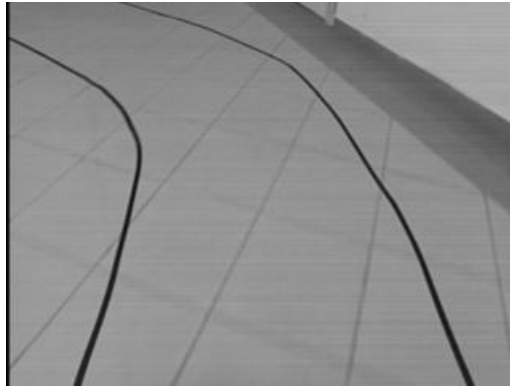
Hough dönüşümü imgedeki doğruları bulmak için kullanıldığında sırasıyla şu işlemler yapılır. İmgedeki gri tonu seviyesi belirli bir seviyenin üstünde olan pikseller için dikey kenar filtresi uygulanır. Filtre çıkışı yine belirli bir eşğin üstünde olan pikseller için, gradiyent alınarak θ aralığı $-\Delta\theta$ ve $\theta + \Delta\theta$ olarak belirlenir. Gradiyent işlemi tam doğru sonuç vermeyebileceği için böyle bir aralık belirlenmiştir. Aralık belirlendikten sonra θ aralığı boyunca, θ çözünürlüğünde r değerleri hesaplanır. Bu r değeri hesaplandıktan sonra akümülatör matrisinin (r, θ) indeksli elemanın değeri 1 artırılır. Bu işlem imgedeki bütün pikseller için tekrarlanır. Son olarak bu akümülatör matrisinden maksimumlar bulunarak doğruların açısı ve orijinden uzaklıkları bulunabilir. Şekil 2.6'da tez çalışması sırasında imgedeki çizgileri bulmak için kullanılan Hough dönüşümünün akış şeması görülmektedir.



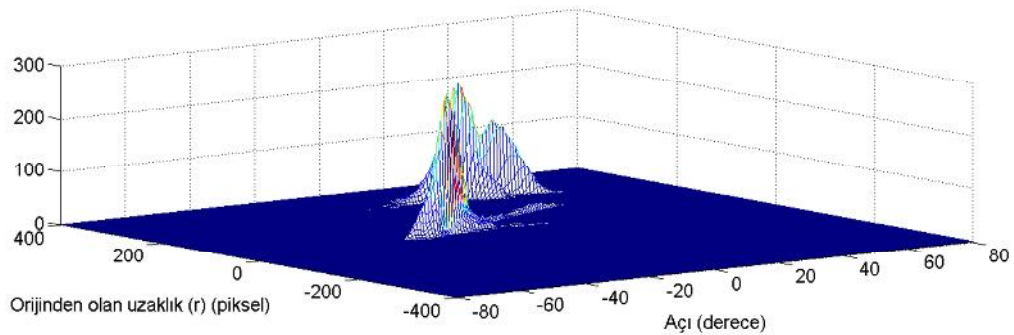
Şekil 2.6:Hough dönüşümü akış şeması

İmgenin Hough dönüşümü uygulanırken dönüşümün daha doğru olması ve iyi sonuçlar vermesi için yapılması önerilen işlemlerden biri de akümülatör matrisindeki ilgili (r, θ) elemanının değerini “1” arttırdıktan sonra komşu elemanlarının değerlerini de arttırmaktır. Tez çalışması sırasında THR1 ve THR2 değerleri deneme-yanılma yoluyla sırasıyla 75 ve 50 olarak bulunmuştur. $\Delta\theta$ değeri ise yine deneme yanılma yoluyla 6 olarak belirlenmiştir.

Tez çalışma sırasında geliştirilen kodda Hough matrisi, üzerinde gerçekleştirilecek işlemlerin hızlı olabilmesi için tek boyutlu dizi olarak tanımlanmış olup, indeks değerleri de tek boyutlu diziye uygun olacak şekilde yeniden ayarlanmıştır. Şekil 2.8’de, Şekil 2.7’deki orijinal imge için Hough dönüşümü alındıktan sonra elde edilen Hough dizisi verilmiştir.



Şekil 2.7: Hough matrisi için orijinal imge



Şekil 2.8: Elde edilen Hough dizisi

Hough dizisinden, hough matrisi elde edilirken indeks deęerleri yeniden d zenlenmelidir. Őekil 2.9’da, Őekil 2.8’deki Hough dizisinden elde edilen Hough matrisi verilmiřtir.



Őekil 2.9: Hough d n ř m 

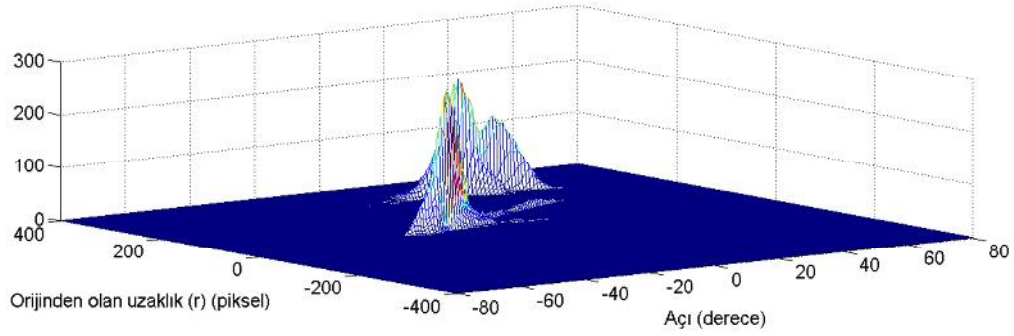
2.2.4.1. Hough d n ř m  ile řerit tespiti

Hough dizisini elde edip maksimumlarını bulduktan sonra bu maksimum noktadaki r ve θ deęerleri bize doęrunun denkleminde doęrunun uzaklıęını ve yerini (ačısını) verir. Őekil 2.8’de de g r ld ę  gibi Hough dizisinde bir ok yerel maksimum noktası elde edilmiřtir. Bunların bir oę  g r lt  tabanlı oluřan maksimumlardır. Bulunan bu yerel maksimum deęerlerinden  eřitli eliminasyonlar yapılarak i lerinden řerit  izgisi olabilecek olanlar  ıkarılır. Yapılan bu eliminasyonlar y ntemleri bařlıęı altında incelenecektir.

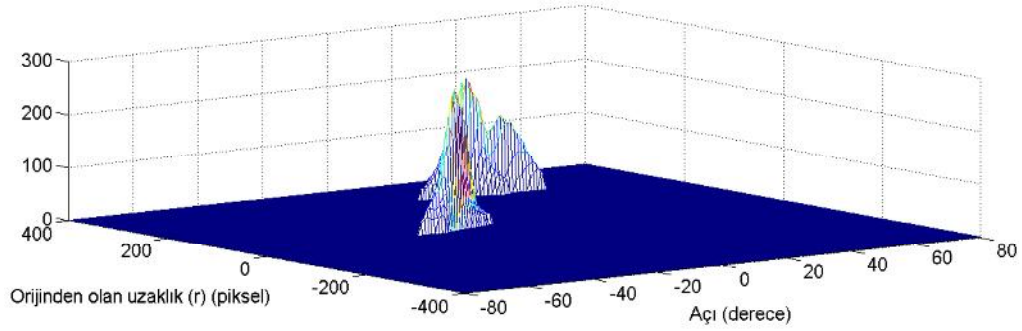
2.2.4.2. Eliminasyon y ntemleri

Bir  nceki kısımda da belirtildięi gibi Hough dizisini elde edilen maksimumlarının bir oę  g r lt lerden ve yoldaki dięer řekillerden elde edilen maksimumlardır. Elde edilen bu maksimumların arasından řerit  izgilerini tespit edebilmek i in bir takım eliminasyonlar uygulanmıřtır.

İlk olarak elde edilen Hough dizisindeki yerel maksimumlar için bir eşik değeri belirlenmiştir. Bu eşik değerinin altında kalan değerler şerit çizgisi olarak kabul edilmemiştir. Şekil 2.10'da örnek imge için elde edilen Hough dizisi gösterilirken, Şekil 2.11'de ise eşikten geçirildikten sonra elde edilen Hough dizisi gösterilmiştir. Tez çalışması sırasında eşikleme işlemi için kullanılan değer deneme yanılma yoluyla 20 olarak belirlenmiştir.

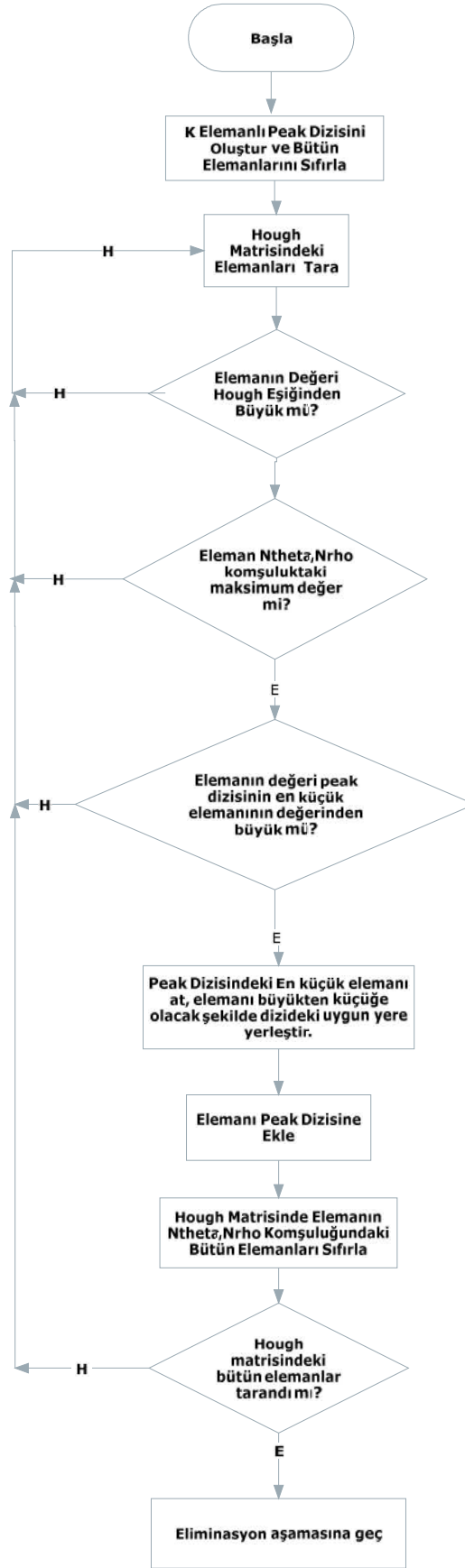


Şekil 2.10: Örnek Hough dizisi



Şekil 2.11: Eşiklemeyen sonra elde edilen Hough dizisi

Eşikleme işlemi yapıldıktan sonra bulunan Hough dizisi içerisinde en büyük skorlara sahip yerel maksimumlar belirlenmiş ve bulunan bu maksimumlar başka bir diziye sıralı olacak şekilde atılmıştır. Bu işlemler için kullanılan akış şeması Şekil 2.12'de gösterilmiştir.



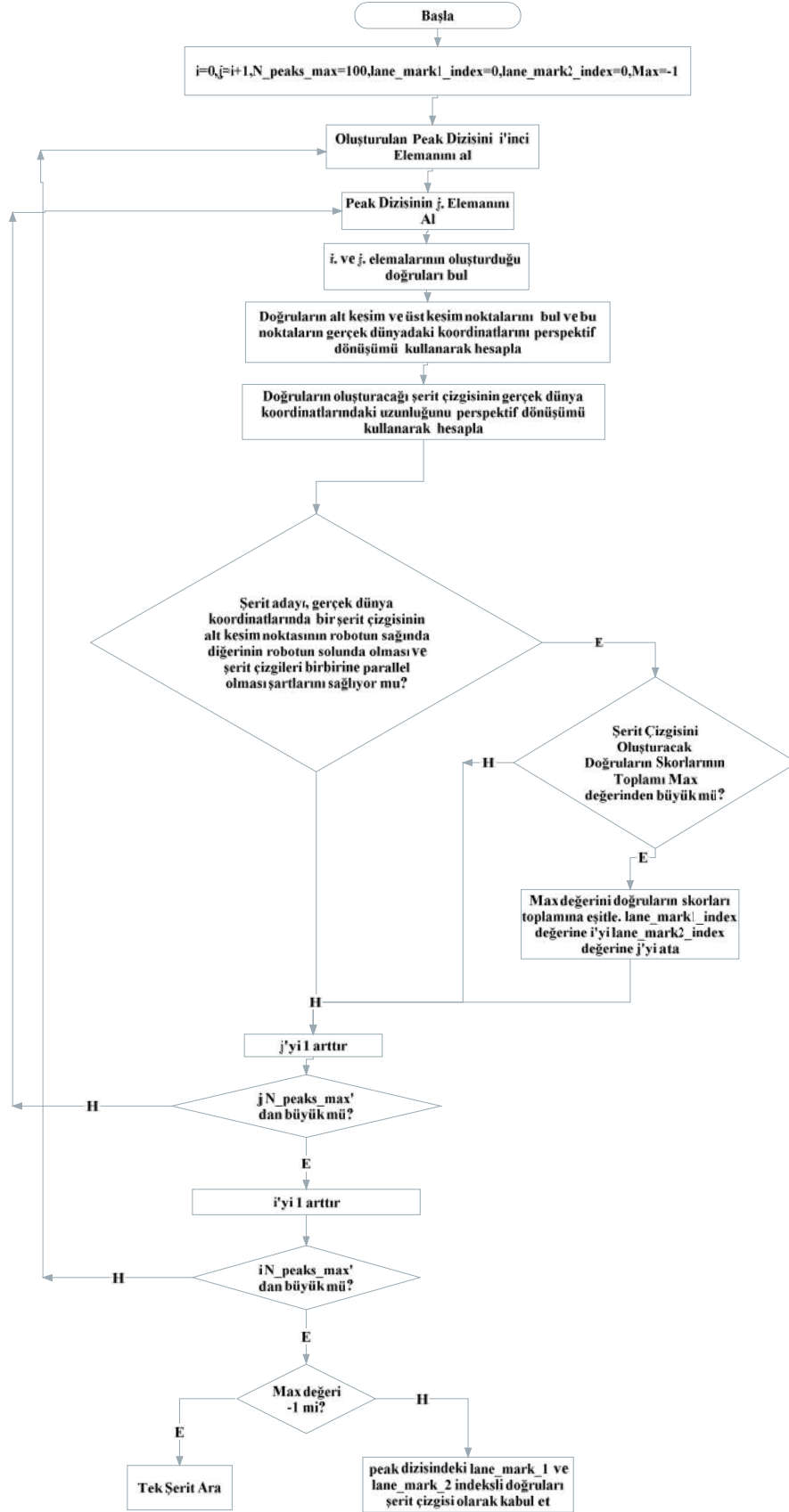
Şekil 2.12: Hough dizisindeki maksimumların bulunmasının akış şeması

Deney sırasında yapılan çalışmada peak dizisinin boyutunu belirleyen K değeri deneme yanılma yoluyla 100 olarak bulunmuştur. Ntheta ve Nrho değerleri de sırasıyla 5 ve 10 olarak belirlenmiştir.

Daha sonra bulunan maksimumlar içerisinde en iyi ikililer seçilmiştir. Bu ikililer arasından şerit çizgisi olabilecek olanlar seçilip, diğerleri elenmiştir. Bir doğru çiftinin şerit çizgisi olarak kabul edilebilmesi için şu şartları sağlaması gerekmektedir.

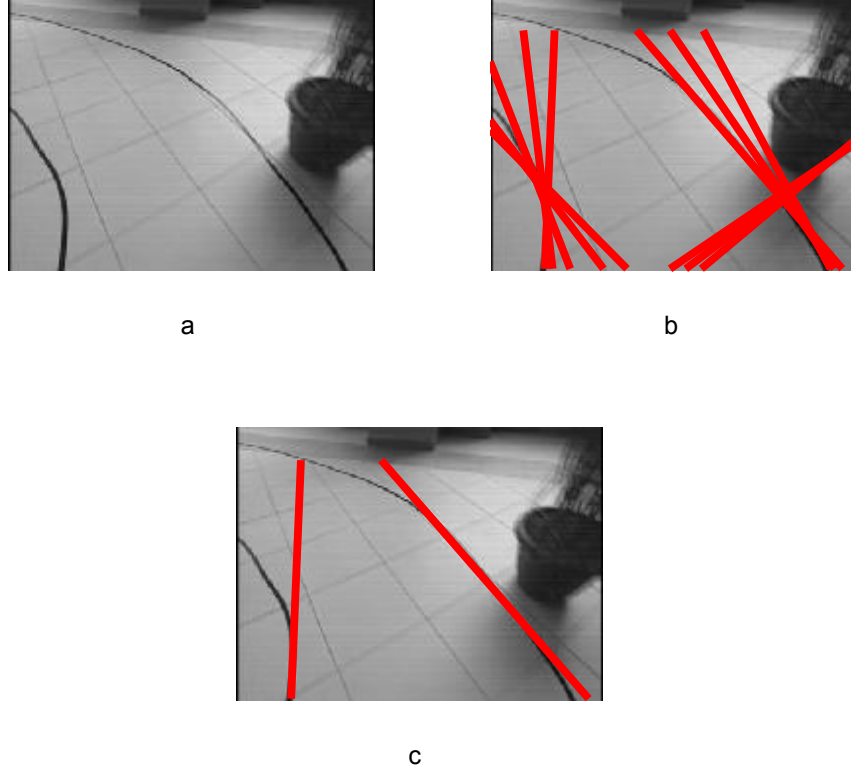
- Doğruların imge uzayındaki alt kesim noktaları arasındaki uzaklık, üst kesim noktaları arasındaki uzaklıktan fazla olması
- Doğruların imgedeki alt kesim noktalarının perspektif dönüşümü alınarak elde edilen gerçek dünya koordinatlarındaki şerit genişliğinin, gerçek şerit genişliğine oranı belirli toleransla eşit olmalı
- Doğruların imgedeki üst kesim noktalarının perspektif dönüşümü alınarak elde edilen gerçek dünya koordinatlarındaki şerit genişliğinin, doğrunun imgedeki alt kesim noktalarının perspektif dönüşümü alınarak elde edilen gerçek dünya koordinatlarındaki şerit genişliğine oranının belirli sınırlar içerisinde olması

Şekil 2.13’de iki şerit bulma akış şeması verilmiştir. Akış şemadaki şartlardan biri olan iki çizginin birbirine paralel olması durumu da şu şekilde hesaplanır. İki çizgi birbirine paralel olursa, bu doğrular arasındaki uzaklık paralellikten dolayı her noktada sabittir. Kod aşamasında paralellik kontrol edilirken perspektif dönüşümü kullanılarak gerçek dünya koordinatlarındaki uzaklık kontrol edilmiştir. Eğer iki çizginin birbirine olan uzaklığı görüntünün alt noktasında ve üst noktasında %10 toleransla aynı ise bu iki çizgi birbirine paralel olarak kabul edilebilir.



Şekil 2.13: İki şerit çizgisinin bulunmasının akış şeması

Şekil 2.14’de örnek bir imgede oluşan Hough çizgileri ve eliminasyon yapıldıktan sonra elde edilen Hough çizgileri verilmiştir.

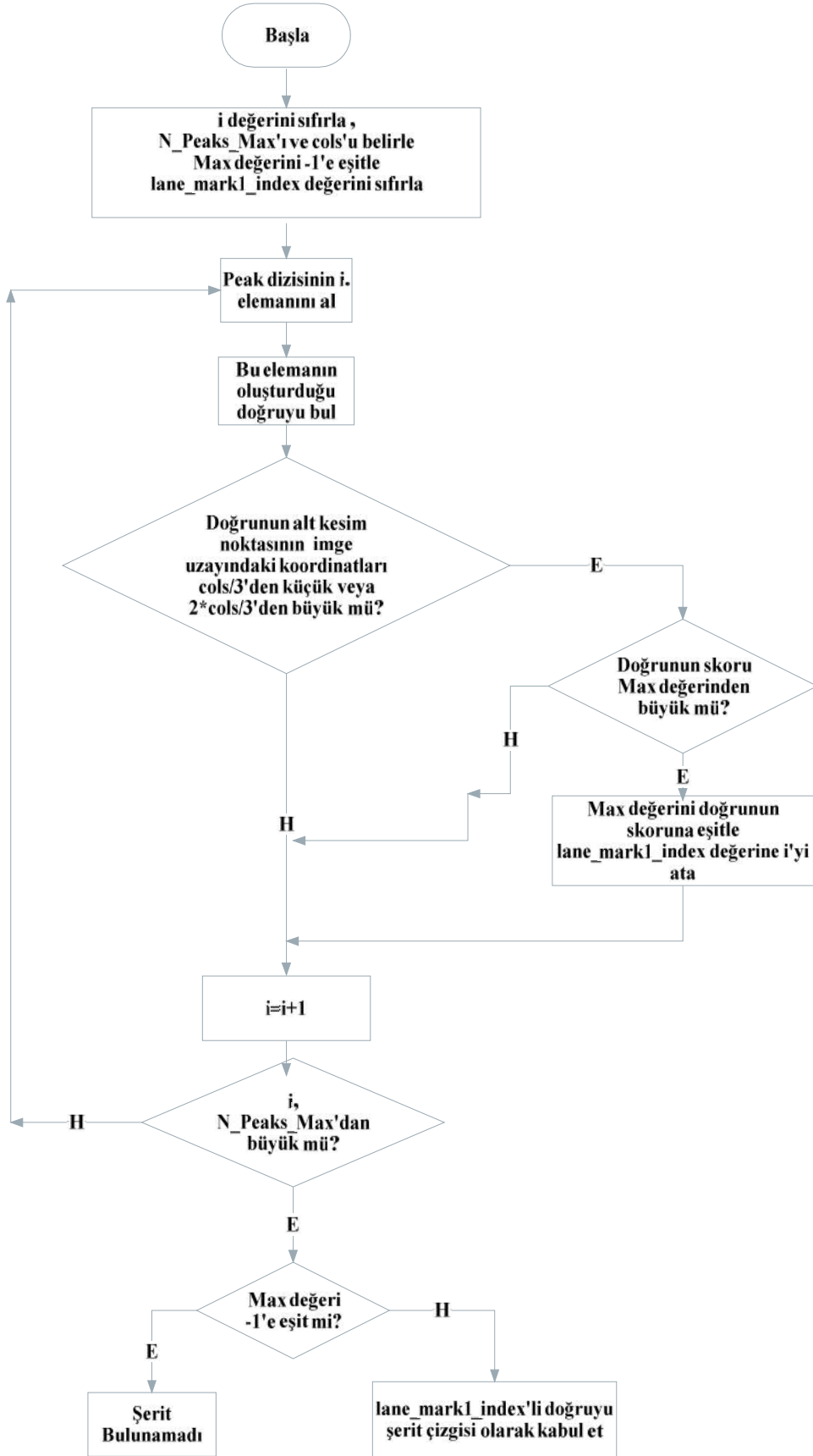


Şekil 2.14: a) Orijinal imge b) Bütün Hough çizgileri c) Bulunan şerit çizgileri

Eğer şartları sağlayan bir şerit çifti yok ise algoritma bu sefer tek şerit çizgisi olup olmadığını kontrol eder. Bir doğrunun şerit çizgisi olarak sayılabilmesi için aşağıdaki şartları sağlaması gerekmektedir.

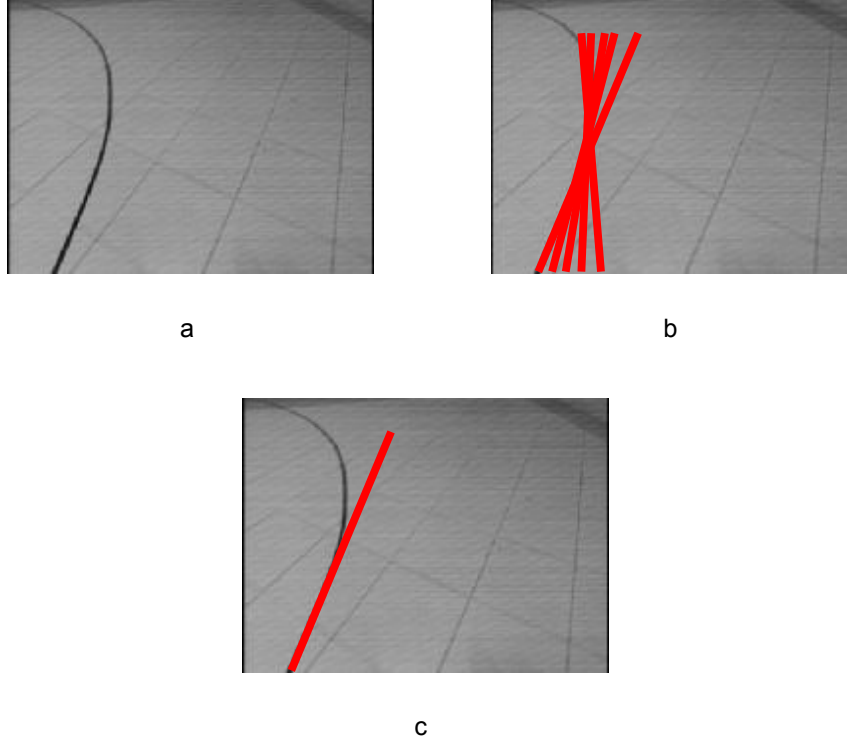
- Doğrunun imgedeki alt kesim noktasının koordinatlarının belirli sınırlar içerisinde olması
- Doğrunun en büyük skorlu doğru olması

Şekil 2.15’de tek şerit bulunmasının akış şeması verilmiştir.



Şekil 2.15: Tek şerit çizgisinin bulunmasının akış şeması

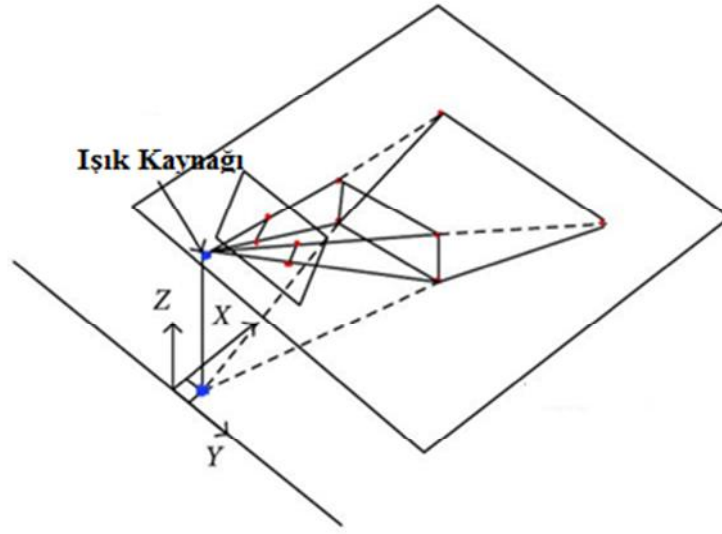
Şekil 2.16’de elde örnek bir imgede elde edilen tüm Hough çizgileri ve bulunan tek şerit çizgisi gösterilmektedir.



Şekil 2.16: a) Orijinal İmge b) Bütün Hough Çizgileri c) Bulunan şerit çizgisi

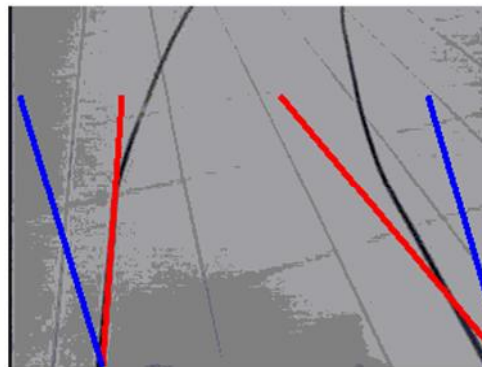
2.2.5. Ters perspektif dönüşümü

Perspektif bozunumu kameralarda ve fotoğraf makinelerinde yamuksal bir görünüşün bir dikdörtgensel bir görünüme dönüştürülmesi sonucu ortaya çıkan sıkça karşımıza çıkan bir sorundur. Kameranın görüş alanının bir yamuk olmasına rağmen kameradan alınan görüntü bir dikdörtgen halindedir. Bu durum Şekil 2.17’de açıkça gösterilmiştir. Perspektif etkisini yok etmek amacıyla birçok yöntem kullanılmıştır [43]. Ayrıca bu perspektif etkisi şerit tespitinde de kullanılabilir [44,45].



Şekil 2.17: Perspektif hatası

Perspektif dönüşümü sayesinde imge koordinatlarındaki çizgiler gerçek dünya koordinatlarına dönüştürülmüştür. Bu dönüşümü yapılırken öncelikli olarak kameranın görüş açısı olan yamuğun sınırları belirlenmiştir. Bunun yapılması için kameranın görüş açısının sınırlarına değişik elemanlar koyularak, elemanlar arasındaki uzaklık bir metre ile ölçülüp yamuğun sınırları kaydedilmiştir. Tez çalışması sırasında sadece resmin alt kesim ve üst kesim noktalarındaki noktaların ters perspektif dönüşümünün alınması yeterli olmuştur. Şekil 2.18’de tez çalışması sırasında yapılan ters perspektif dönüşümü ile şeritlerin gerçek dünya üzerindeki koordinatları ile imge uzayındaki koordinatları gösterilmiştir.



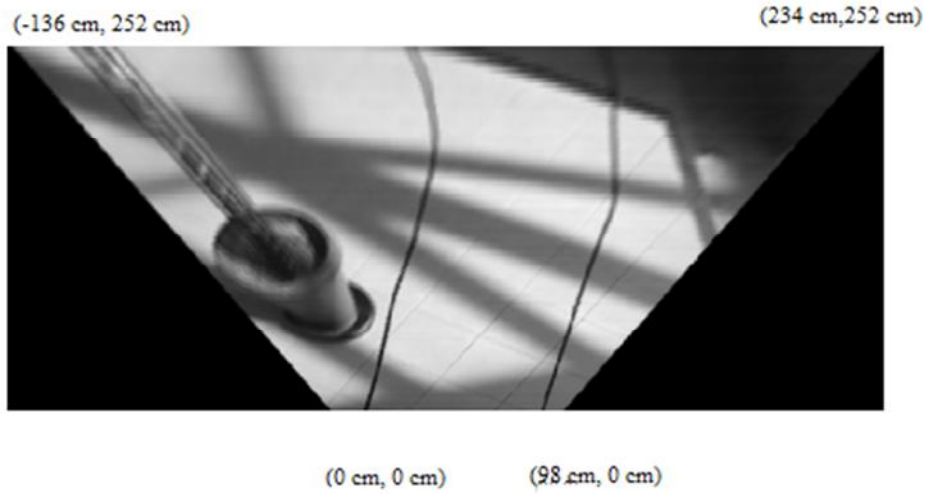
Şekil 2.18: Şeritlere uygulanan perspektif dönüşümü

Kırmızı renkte olan çizgiler imge uzayında bulunan çizgiler iken, mavi renkte olan çizgiler ters perspektif dönüşümü yapılarak elde edilen çizgilerdir. Görüldüğü gibi

imge uzayında çizgiler gittikçe daralır halde görülürken ters perspektif dönüşümü alındığında çizgilerin birbirine paralel olduğu görülmüştür. Şekil 2.19’da orijinal imge gösterilirken, Şekil 2.20’de ters perspektif dönüşümü bütün imge üzerine uygulanmıştır.



Şekil 2.19: Ters perspektif dönüşümü için orijinal imge



Şekil 2.20: Ters perspektif dönüşümü alınmış imge

Şekil 2.20’de alınan bütün görüntünün ters perspektif dönüşümü yapılmıştır. Şekilden de görüleceği gibi yamuğun dışında kalan alanlar kameranın görüş açısının dışında kalmaktadır ve bu yüzden siyah renkli olarak görünmektedir. Buna göre kamera resim üzerinde siyah ile görünen alanları perspektif hatası yüzünden algılayamamaktadır.

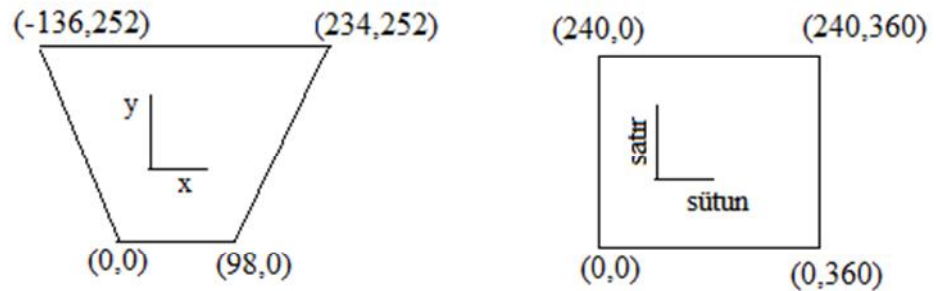
Perspektif dönüşümünü tüm görüntü üzerinde uygulayabilmek için bir H matrisinden faydalanılmıştır. H matrisi elde edilirken görüş alanının oluşturduğu yamuğun dört köşesi ile oluşan görüntünün dört köşesinin birbirine olan benzerliği (homografi) kullanılabilir. İki dörtgen şekil arasında, ters perspektif dönüşümü yapılırken 3x3'lük benzerlik matrisi kullanılması yeterli olmaktadır. H matrisi aşağıdaki gibi gösterilebilir;

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Elde edilen 3x3'lük H benzerlik matrisi ile imge matrisinin oluşturan elemanların orijin noktasına (imgenin sol alt köşesi) göre indeks değerleriyle oluşturulan vektör çarpılarak o noktanın gerçek dünya koordinatlarındaki olması gereken yer elde edilmiştir. Aşağıda bu denklem verilmiştir;

$$\begin{bmatrix} x^* \\ y^* \\ k \end{bmatrix} = H^* \begin{bmatrix} \text{sütun indeks} \\ \text{sıra indeks} \\ 1 \end{bmatrix} \quad (2.1)$$

Bu denklemdeki indeks değerleri ile x, y değerleri Şekil 2.21'de açıklanmıştır.



Şekil 2.21: İmge uzayı ve perspektif etkisi

Bu vektörün ile H matrisi çarpılarak elde edilen vektör üçüncü elemanına göre normalize edilerek gerçek dünya koordinatlarındaki değerler elde edilebilir. Normalizasyon işlemi aşağıda görüldüğü gibidir.

$$x=x^*/k \quad (2.2)$$

$$y=y^*/k \quad (2.3)$$

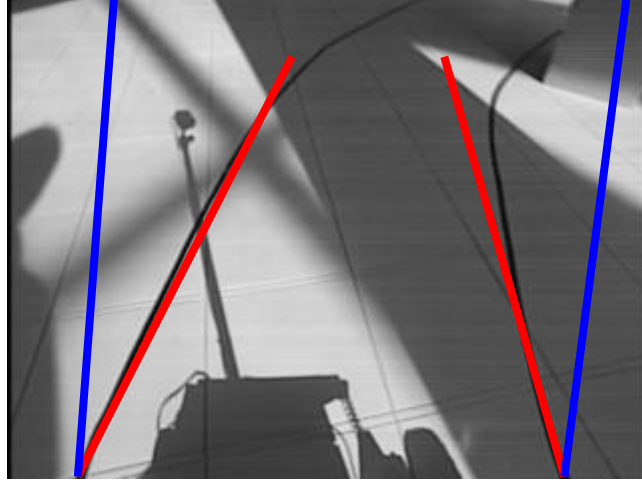
H matrisinin değerlerini elde etmek için köşe noktaları kullanılır. Seçilen orijin noktasına göre imge uzayındaki uzaklıklar satır veya sütun sayısına göre bakarak belirlenir. Gerçek dünya koordinatları da ölçüm yapılarak belirlenir ve bu değerler kullanılarak H matrisinin elemanları elde edilir.

Tez çalışması sırasında elde edilen benzerlik (homografi) matrisi aşağıda belirtildiği gibidir;

$$H = \begin{bmatrix} 3.6633 & 3.2142 & 0 \\ 0 & 5.8216 & 0 \\ 0 & 0.0179 & 1 \end{bmatrix}$$

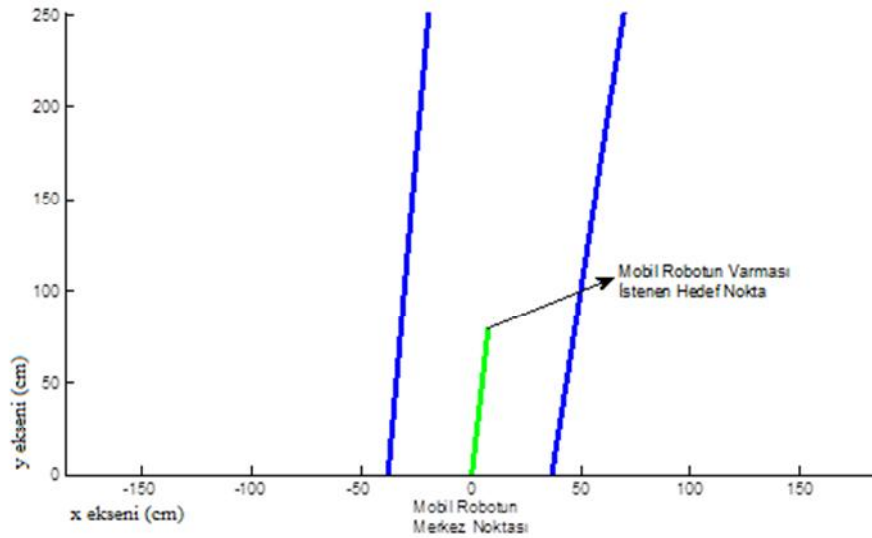
2.2.6. Mobil robotun hareket yönünün kontrolü

Mobil robotun hareket yönünü belirlemek için şerit çizgileri bulunduktan sonra, bulunan şerit çizgilerinin alt kesim ve üst kesim noktalarının perspektif dönüşümü alınır. Böylelikle gerçek dünya koordinatlarındaki şerit çizgilerine ulaşılır. Bu ulaşılan şerit çizgilerinin eksenindeki 80 cm ilerisinde, orta noktası mobil robotun yol üzerinde gitmesi gereken nokta olarak belirlenmiştir. Bu durum Şekil 2.22’de gösterilmiştir. Şekil üzerinde yer alan kırmızı renkteki çizgiler algoritmanın belirlediği şerit çizgileri olurken, mavi renkteki çizgiler ise bulunan şerit çizgilerinin ters perspektif dönüşümü yapılmış halidir.



Şekil 2.22: Şerit çizgilerinin ters perspektif dönüşümü

Aynı görüntü için varılması istenen hedef noktanın X ve Y eksenleri kameranın tüm görüş alanı olması halinde robotun merkeziyle oluşturduğu doğrunun grafiği Şekil 2.23’de gösterilmiştir. Grafikteki (0,0) noktası mobil robotun merkez noktasını göstermektedir. Mavi çizgiler şerit çizgisini ifade ederken yeşil çizgi ise 80 cm ileride mobil robotun varması gereken hedef nokta olarak grafikte gösterilmiştir. Bu yeşil çizgiyle gösterilen doğrunun açısı mobil robotun yönelim açısı olarak belirlenmiştir.

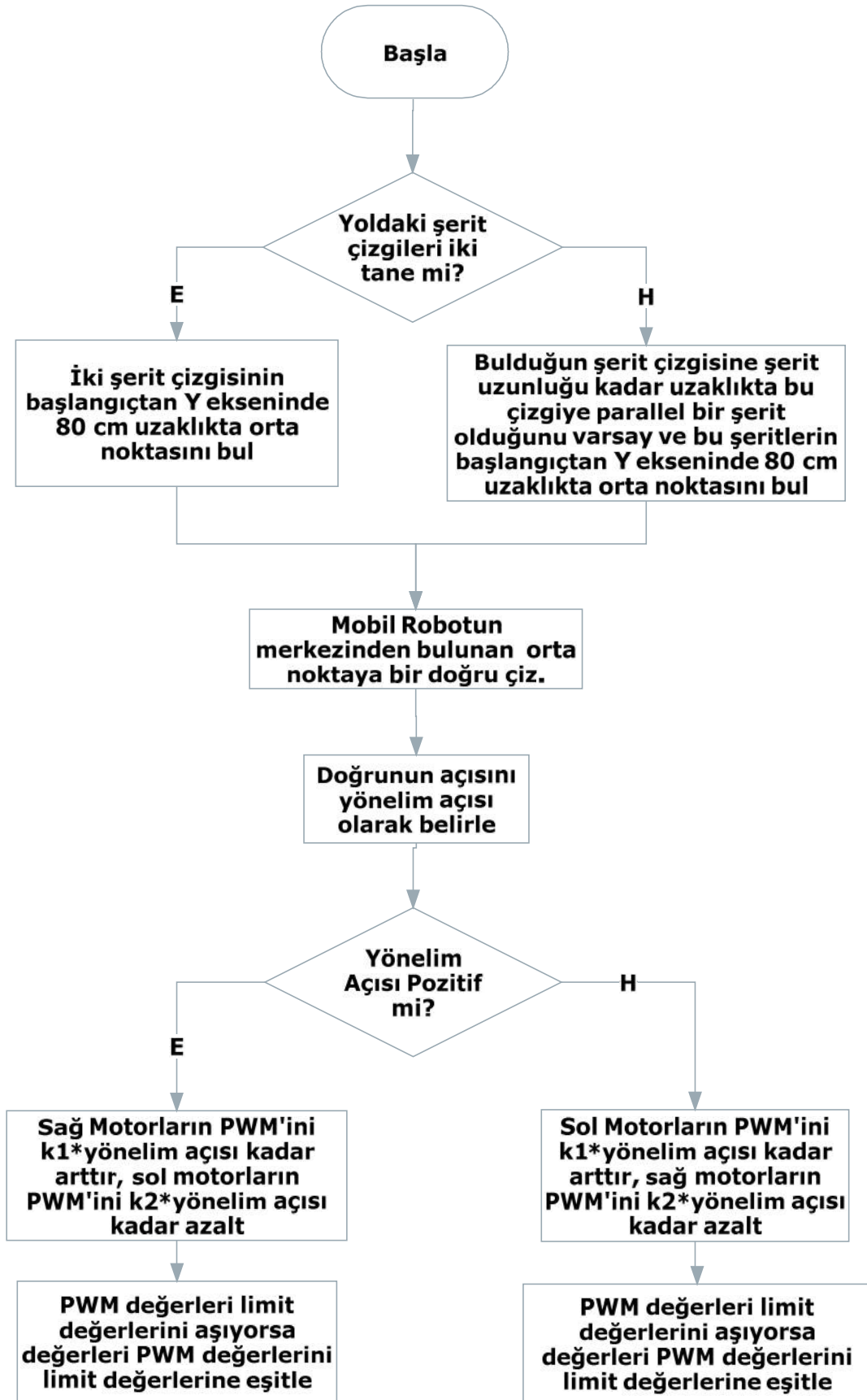


Şekil 2.23: Mobil robotun varması istenen hedef nokta

Mobil robotun hareketi birbirinden bağımsız olarak çalışan dört adet motorla sağlanmaktadır. Mobil robotun sağa ya da sola dönmesi ise bu motorların kontrolünü

sağlayan sürücülere uygulanan PWM sinyalinin doluluk boşluk oranının değiştirilmesi ile ayarlanır. Eğer sağ taraftaki motorlara doluluk boşluk oranı daha yüksek bir PWM sinyali uygulanırsa mobil robot sola doğru, eğer sol taraftaki motorlara doluluk boşluk oranı daha yüksek bir PWM sinyali uygulanırsa mobil robot sağa doğru hareket etmektedir.

Sistemin hareketini sağlayan motorların kontrolü için oransal kontrol kullanılmıştır. Oransal kontrolde, çıkış, referans alınan değerinin değişimi oranında değiştirilir [46]. Başlangıçta dördü de aynı hıza dönen motorların hangi ikisinin daha hızlı çalışacağına karar vermek için mobil robotun yönelim açısının bulunması gerekmektedir. Eğer bulunan açı pozitif ise mobil robot sola negatif ise sağa hareket edecektir. Motorların hızlarının arttırılması işlemi için yönelim açısı referans olarak alınmış olup, bu değere bağlı olarak bir P (oransal) kontrol yapılmıştır. Yapılan kontrolün akış şeması Şekil 2.24’de verilmiştir.



Şekil 2.24: Mobil robotun yönünün kontrolü akış şeması

Deneme yanılma yoluyla k_1 değeri 3, k_2 değeri 1,5, minimum PWM değeri 5, maksimum PWM değeri 95 olarak bulunmuştur.

Ayrıca mobil robot döndürülürken PWM sinyalindeki artış kademeli olarak yapılarak daha yumuşak dönüşler sağlanmıştır. Fakat bu kademeli artış sırasında artış miktarının doğru olarak ayarlanması gerekmektedir. Eğer bu artış miktarı yetersiz veya aşırı olursa mobil robot yoldaki şerit çizgilerinin dışına çıkabilir. Bu artış miktarının ayarlanmasında mobil robotun boyut bilgileri ve manevra kabiliyeti önemlidir. Tez çalışması sırasında yapılan testler ile her kademedeki artış ve azalış miktarının 5 olması durumunda mobil robotun şerit çizgilerinden çıkmadan parkurda rahatlıkla dönebildiği gözlenmiştir.

2.3. Optimizasyon

Şerit takibinde kullanılan Hough dönüşümünün getirdiği hesapsal yük gerçek zamanlı şerit takibinin önündeki en büyük engel olarak karşımıza çıkmaktadır. Şerit takibinin gerçek zamanlı olabilmesi için algoritma ve kod üzerinde birçok optimizasyon yapılmıştır.

Tez çalışması sırasında geliştirilen kodun üzerinde çalışacağı işlemciye uygun veri tiplerini seçilmesi de kodun hızlı çalışması için önemlidir. Bu işlem bir kodlama tekniği olduğu için optimizasyon sayılamayabilir fakat kodun hızlı çalışabilmesi için gerekli bir işlemdir.

Kameradan alınan görüntünün sadece yarısının Hough dönüşümünün alınıp işlenmesi de hesapsal yükümüzü azaltır. Mobil robotun ilerisini değil o anda bulunduğu şeritleri bulması gerekmektedir. Bu yüzden kameradan alınan görüntünün en ilerisinin işlenmesine gerek yoktur ve görüntünün yarısının işlenmesi yeterli olabilmektedir. Bu işlem de bir kodlama tekniğidir ve optimizasyon sayılamayabilir fakat işlem yükünü azaltılması nedeniyle kodun daha hızlı çalışmasını sağlar.

Tez çalışması sırasında literatürde bulunan optimizasyonlardan iki tanesi kullanılmıştır. Bunlardan biri Hough dönüşümünde kullanılacak açılışın filtre

çıkışından elde edilmesi diğeri ise Hough dönüşümünde kullanılacak olan trigonometrik fonksiyonların değerlerinin bir LUT'dan elde edilmesidir.

2.3.1. Açı bilgisinin filtre çıkışından alınması

Şerit takibinde kullanılan Hough dönüşümünde, önceki kısımlarda anlatıldığı gibi eğer noktanın piksel değeri belirli bir eşiğin altında ve aynı zamanda bir kenar noktası ise bütün Θ (açı) değerleri için, r (orijinden olan uzaklık) değerlerinin hesaplanması gerekmektedir. Bu işlem için kurulması gereken döngünün hangi aralıklarını ve döngüdeki artış miktarını Θ değeri belirler. Döngünün sınırları Θ değerinin sınırları, döngüdeki artış miktarı ise Θ değerinin çözünürlüğüdür.

Tez çalışması sırasında Θ değerinin sınırları -85° ile $+85^\circ$ ve Θ değerinin çözünürlüğü 1° olarak belirlenmiştir. Bu yüzden her bir kenar noktası için toplamda $(\Theta_{\max} - \Theta_{\min}) / \Theta_{\text{res}}$ değeri kadar döngüde kalmak gerekmektedir ve bu yukarıda belirttiğimiz değerler için işlemin 171 defa yapılması gerekir. Bu işlem her bir kenar noktası için yapılacağı için bize çok büyük hesapsal yük getirir ve bu hesapsal yük algoritmanın yavaşlamasına dolayısıyla da gerçek zamanlı olarak çalışamamasına neden olur. Bu hesapsal yükü azaltmak için ya aranılan açı değerinin sınırlarının azaltılması ya da çözünürlüğün arttırılması gerekmektedir. Çözünürlük arttırıldığında, algoritmanın hassasiyeti azalacağı için algoritma yanlış sonuçlar verebileceğinden dolayı, çözünürlüğün arttırılması yerine sınırların azaltılması işlemi daha doğru olacaktır.

Sınırların azaltılması için Θ değerinin alabileceği değerlerin öncelikli olarak tahmin edilebilmesi gerekmektedir. Bu işlem için basit yatay ve düşey kenar filtreleri kullanılmıştır. Belirli bir eşiğin altındaki noktaya uygulanan düşey kenar filtresiyle elde edilen değer eğer belli bir değerin üzerinde ise nokta kenar noktası olarak kabul edilip (G_y), aynı noktaya bir de yatay kenar filtresi uygulanarak bir değer elde edilmiştir (G_x). Bu iki değer oranın ark tanjant (arktan) değeri ise Θ değerini vermektedir. Hough dönüşümünde kullanılacak olan Θ değeri de bu elde ettiğimiz Θ değeri civarlarında olacaktır. Θ değerinin, artık -85° ile $+85^\circ$ arasında değil elde ettiğimiz G_y , G_x oranından bulunan değerin civarlarında olduğu bilinmektedir. Bu yüzden elde ettiğimiz Θ değeri civarındaki $\pm 6^\circ$ 'lik aralıkta Hough dönüşümünün

bulunması yeterli olur. Böylelikle her bir nokta için 171 defa döngüde kalmak yerine 13 defa döngüde kalınır ve bu sayede ciddi bir kod optimizasyonu sağlanır. Tablo 2.1’de bu optimizasyon yapılmadan önce saniyede işlenen imge sayısı ile bu optimizasyon yapıldıktan sonra saniyede işlenen imge sayısı verilmiştir.

Tablo 2.1’de görülebileceği gibi açı bilgisinin kenar filtresi çıkışından elde edilmesiyle algoritmanın saniyede işlediği imge sayısında yaklaşık olarak üç kat artış sağlanmıştır. Bu aşama görüntünün gerçek zamanlı olarak çalışabilmesi için çok önemli bir adım alınmıştır.

2.3.2. Trigonometrik fonksiyonların değerlerinin LUT’dan elde edilmesi

Hough dönüşümünde kullanılan trigonometrik fonksiyonlar tanjant, ark tanjant, sinüs, kosinüs fonksiyonlarıdır. İşlemci bu fonksiyonların değerlerini elde ederken çok zaman kaybetmektedir ve bu yüzden bu değerlerin her defasında tekrar hesaplanması yerine bu değerlerin programın başlangıcında sadece bir defa hesaplanıp o değerlerin bir LUT’a aktarılmasıyla da kod üzerinde bir optimizasyon yapılmıştır.

Trigonometrik fonksiyonlar içerisinde en çok ark tanjant işlemi zaman almaktadır. Bu işlem Θ değerinin filtre çıkışından hesaplanmasında kullanıldığı için her bir kenar noktası için kullanılmaktadır. Her defasında bu değer hesaplanması yerine bu değer bir LUT’dan elde edilmesi ile optimizasyon sağlanmıştır. Özellikle de ark tanjant işlemi trigonometrik fonksiyonlar arasında hem en zor yapılan işlem, hem de en çok kullanılan işlem olduğu için LUT ile yapılan optimizasyon algoritmaya yaklaşık olarak üç kat hız kazandıran işlemdir.

Ayrıca her bir r değeri Θ değerinden $r = x\cos(\Theta) + y\sin(\Theta)$ şeklinde elde edileceği için ve bu işlem imgedeki bütün noktalar için yapılacağından dolayı sinüs ve cosinüs fonksiyonlarının değerlerinin de LUT’dan elde edilmesi çok önemli bir adımdır. Tablo 2.1’de bu optimizasyon yapılmadan önce saniyede işlenen imge sayısı ile bu optimizasyon yapıldıktan sonra saniyede işlenen imge sayısı verilmiştir.

Tablo 2.1 : Trigonometrik Fonksiyonların Değerlerinin LUT'dan elde edilmesi

LUT Kullanılmadan Önce		LUT Kullanıldıktan Sonra	
Açı Bilgisi Filtre Çıkışından Alındığında	Açı Bilgisi Filtre Çıkışından Alınmadığında	Açı Bilgisi Filtre Çıkışından Alındığında	Açı Bilgisi Filtre Çıkışından Alınmadığında
11,25fps	2,5fps	30fps	7,5fps

2.3.3. Hafıza organizasyonu

Tez çalışması sırasında Visual DSP ortamında geliştirilen kod BF561 işlemcisi üzerinde çalıştırılmıştır. Çift çekirdeğe sahip bu işlemci sayesinde bir çekirdek görüntüyü almak için bir çekirdek de görüntüyü işlemek için kullanılmıştır. Bu yüzden iki çekirdeğin de görüntü verilerinin saklandığı RAM hafızasına erişmesi gerekmektedir. Görüntü verisi alınırken DMA tanımlayıcı (descriptor) moda kullanılmış ve veriler bir tampon (buffer) mantığı kullanılarak sFrame0 ve sFrame1 adlı değişkenlere yazılmıştır. Bu sayede DMA bir görüntü değerinin saklandığı değişkeni hafızaya yazarken, diğer görüntünün bulunduğu değişken işlemci tarafından işlenmektedir.

Verilerin saklandığı 256 MB'lık SDRAM hafızası 4 farklı kısımdan (BANK) oluşmaktadır ve 120 MHZ hızında çalışmaktadır. İki çekirdeğin de erişeceği alan aynı RAM BANK'ında bulunursa, hafızadan veri alınması veya hafızaya veri yazılması sırasında meydana gelen gecikme yüzünden işlem hızı oldukça yavaşlamaktadır. Bu yüzden alınan sFrame0 görüntüsü BANK1'e yazılırken, sFrame1 görüntüsü ise BANK1'e yazılmıştır. Ayrıca görüntüyü işlerken elde edilen Hough dizisinin de boyutu büyük olduğu için L2 veya L1 hafızaya sığmamaktadır. Bu yüzden Hough dizisi de SDRAM BANK3'e tanımlanmıştır. Böylelikle iki çekirdek birbirinden bağımsız olarak hafızada istenilen yerlere erişmektedir.

L1 hafıza ise işlemcinin çalışma frekansı olan 600 MHZ ile çalıştığı için sistemde bulunan en hızlı hafıza birimidir. Fakat bu hafızanın boyutu 64 KB ile sınırlı olduğu

iin kod sırasında elde edilen Hough dizisi veya peak dizileri bu hafızada tanımlanamamaktadır. Bu hafızada sadece LUT'lar tanımlanmıştır.

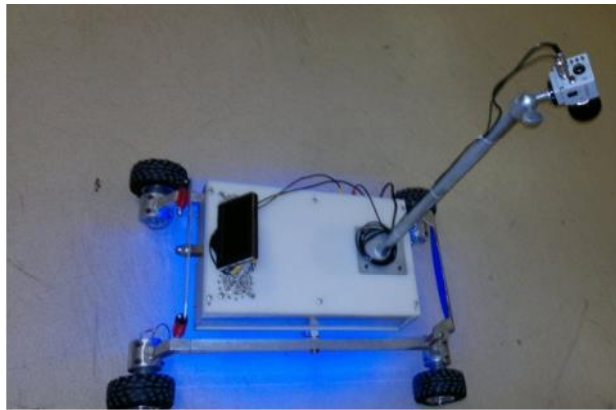
İki ekirdeğın ortak kullanımına sunulan, minimum gecikmeyle erişilebilen, 300 MHZ'de alışan 128 KB'lık L2 hafızada ise işlenen görüntü ile peak dizileri tutulmuştur. Böylelikle sıralama işlemi gibi yüksek hesap yükü bulunan işlem yapılabilcek en hızlı şekilde yapılmıştır. Bu hafızada sadece işlenen görüntü olduėu iin sadece işlenen görüntüyü ekrana yazdıracak olan ekirdeğın bu alana erişmesi yeterli olmaktadır.

3. MOBİL ROBOTUN TASARIMI

3.1. Giriş

Bu bölümde, tez çalışmasının konusu olan otonom hareket eden mobil robotun mekanik olarak tasarımı ve mobil robotu kontrol etmek amacıyla tasarlanmış olan kontrol devreleri açıklanmaktadır. Bu amaç doğrultusunda, tasarlanan devrelere ait PCB tasarımlar ve devrelerin fotoğrafları ile mekanik parçalara ait tasarımlar ve mekanik parçaların fotoğrafları anlatım sırasında kullanılmıştır. PCB tasarımlar ve Mekanik tasarımlar alt bloklara bölünerek buralarda kullanılan elemanların özellikleri ve alt blokların çalışma mantığı detaylı olarak açıklanmıştır.

Tez çalışması sırasında, temel olarak iki farklı devre grubu geliştirilmiştir. Bunlar mobil robotun hareketini motorların kontrolünü sağlayan sürücü devreleri ile pilden gelen gerilimi, görüntü işleme kartı, kamera, LCD ve şerit LED'lerin ihtiyaç duyduğu gerilime dönüştüren regülatör devreleridir. Ayrıca tez çalışması sırasında geliştirilen mobil robot platformu mekanik olarak dört farklı gruptan oluşmaktadır. Bunlar, mobil robotun hareketini sağlayan motorları tutan alüminyum tutucular, mobil robotun ana iskeleti, mobil robotun esnek hareketi sağlayan amortisör katmanını ve görüntü işleme kartının içerisinde bulunduğu plastik kutudur. Bütün bu mekanik ve elektronik kısımlar tezin bu kısmında anlatılacaktır. Şekil 3.1'de geliştirilen mobil robot platformu gösterilmiştir.



Şekil 3.1: Geliştirilen Mobil robot platformu

3.2. Elektronik Devrelerin Tasarımı

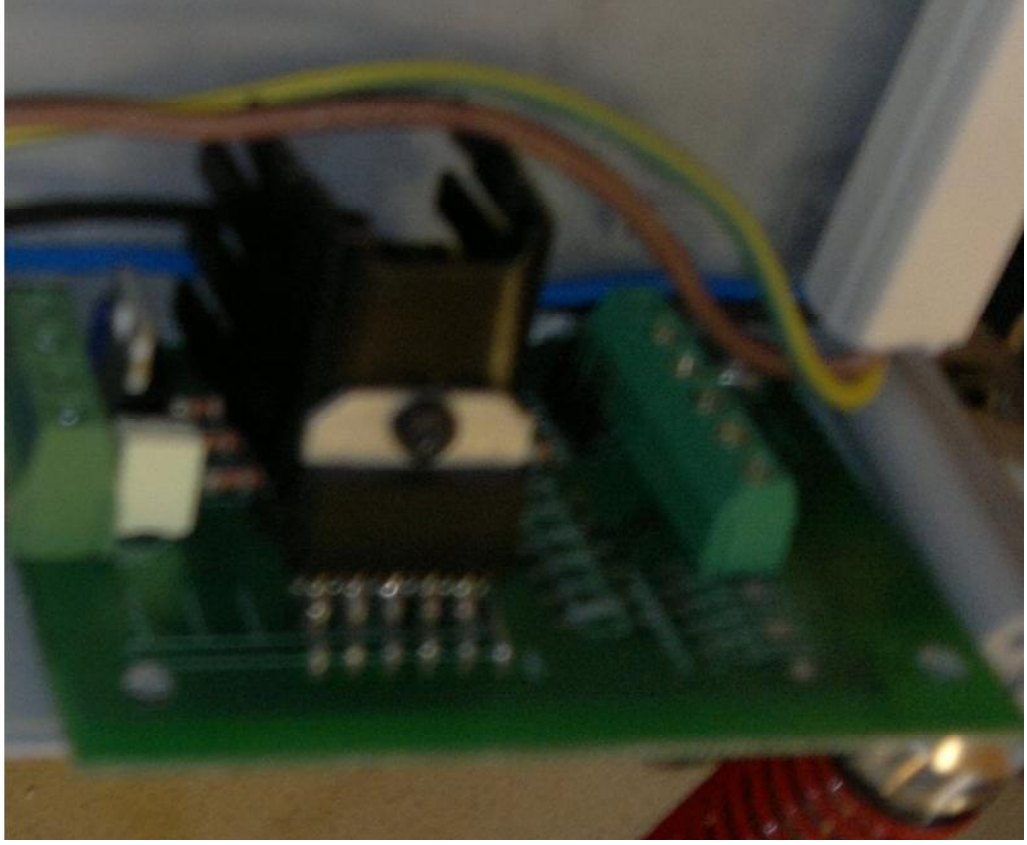
Geliştirilen mobil robot platformuna ait motor kontrol ve sürücü devresi tek bir prototip içerisinde toplanmıştır. Bu devre üzerinde mobil robotun hareketini sağlayan fırçasız dc motorun karttan gelen uygun bir PWM değerine göre sürülmesini sağlayan bir sürücü entegresi, görüntü işleme kartının çıkışlarını sürücü devresinde gelebilecek olası ters gerilimlerden korumak amacıyla bu çıkışları devreden yalıtan bir optokuplör entegresi ve bu entegreden çıkış alabilmek için pilden gelen 24 V'luk gerilim değerini 5V'luk gerilime dönüştüren bir adet regülatör devresi kullanılmıştır. Sürücü devresine giriş ve çıkış klemensler ile sağlanmıştır.

Geliştirilen mobil robot platformunda kullanılan görüntü işleme kartı, kamera, LCD, şerit LED'lerin beslemesinin sağlanması için ise pilden gelen gerilimi bu elemanların ihtiyaç duyduğu gerilim değerlerine düşürmek için regülatör devresi kullanılmıştır. Bu devre üzerinde 12V, 7V ve 9V olmak üzere üç farklı regülatör grubu bulunmaktadır. Görüntü işleme kartı 7V ile çalışırken kamera, LCD, şerit LED'ler ise 12V ile çalışmaktadır. Ayrıca gerilimin düşürülmesi nedeniyle ortaya çıkan sıcaklığın azaltılması için devre üzerinde bir adet soğutucu ve tam olarak bu devrenin üst hizasında yer alan iki adet fan bulunmaktadır.

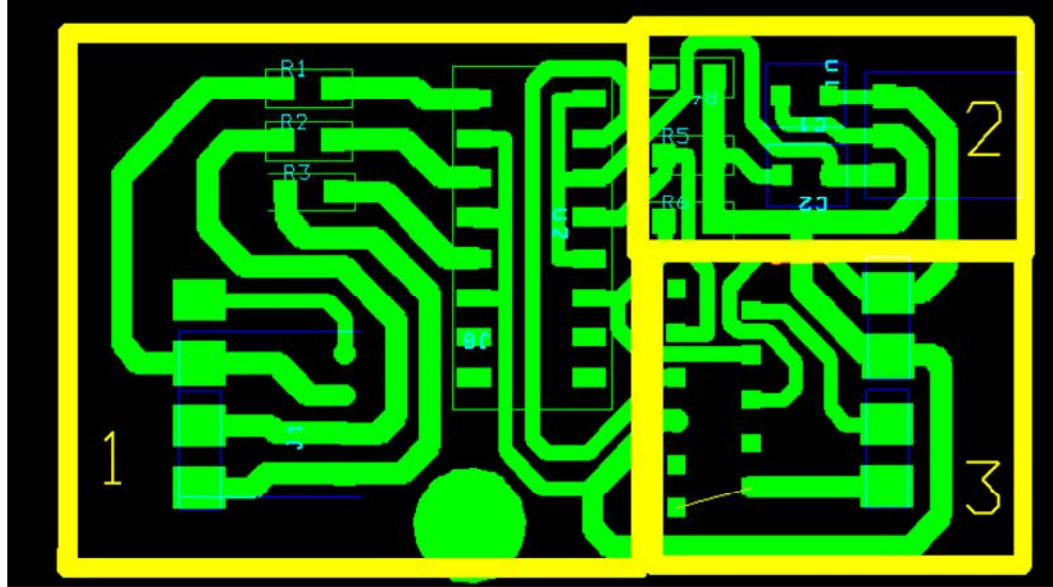
3.2.1. Motor sürücü devresi ve PCB tasarımı

Motor sürücü devresi, geliştirilen mobil robot sisteminin hareketini sağlayan birbirinden bağımsız dört adet fırçasız D.C motorun hızlarını ve dönüş yönlerini kontrol etmek için tasarlanmıştır. Motorların anlık hızlarının ve dönüş yönlerinin kontrolü görüntü işleme kartının timer (sayıcı) çıkışlarından elde edilen PWM sinyali ile sağlanmaktadır. Mobil robotumuz hep ileri yönlü hareket yapacağı için ayrıca görüntü işleme kartının timerları sınırlı olduğu için dönüş yönünün kontrolüne ihtiyaç duyulmamıştır.

Motor sürücü devresi Şekil 3.3'den görülebileceği üzere, anlaşılır bir şekilde açıklanmak amacıyla PCB çizim üzerinde bölümlere ayrılıp her bir bölüm ayrı ayrı numaralandırılmıştır.Devrenin orijinal hali ise Şekil 3.2'de gösterilmiştir.



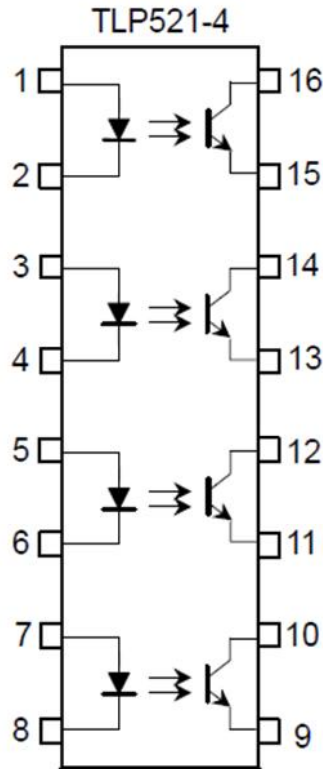
Şekil 3.2: Motor sürücü ve kontrol devresi



Şekil 3.3: Motor sürücü ve kontrol devresi PCB çizimi

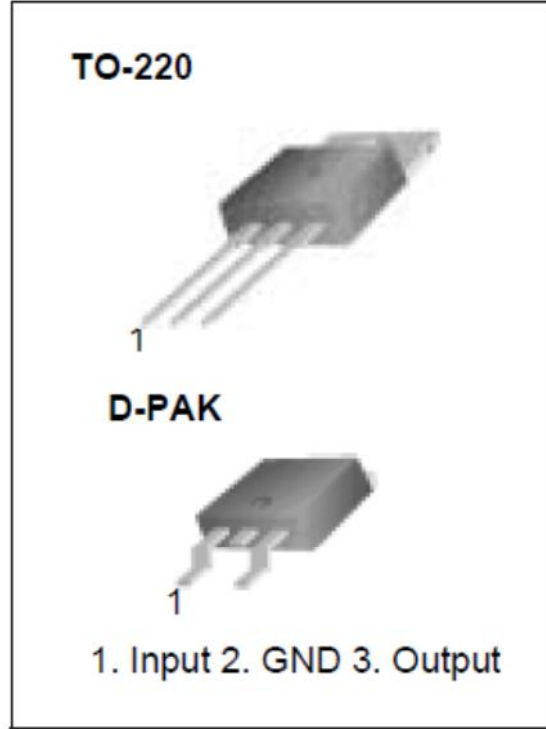
Bu bölümlerin işlevleri ve kullanılan devre elemanlarının özellikleri aşağıdaki gibidir;

1)Bu bölümde, görüntü işleme kartı üzerindeki BF561 işlemcisinin timer çıkışlarından elde edilen PWM sinyalinin devreye giriş olarak uygulandığı kısımdır. Bu amaçla devre üzerinde motorların hızını ayarlamak, motorların dönüş yönünü ayarlamak ve motorları ani olarak durdurmak için üç adet giriş bulunmaktadır. Görüntü işleme kartının timer çıkışlarını sürücülerden gelebilecek olası ters gerilimlere karşı korumak için timer çıkışları, optokuplör entegresi kullanılarak devreden yalıtılmıştır. Burada optokuplör olarak TLP521-4 entegresi kullanılmış olup, bu entegrenin kullanılmasının sebebi yüksek frekanslarda da çalışabilmesidir. Optokuplör entegresinin transistör kısmının emiter pininden çıkış alınarak girişten lojik 1 verildiğinde çıkıştan da lojik 1 alınması sağlanmıştır. Görüntü işleme kartının timer çıkışları 3.3V olmasına rağmen sürücü entegresinin girişine 5V PWM verilmesi gerekmektedir. Bu yüzden optokuplör entegresinin transistör kısmının emiter pinine +5V verilmiştir. Ayrıca görüntü işleme kartının timer çıkışları optokuplör entegresinin LED kısmına giriş olarak verilirken 1.8K'lık bir direnç üzerinden geçirilmiştir. TLP521-4 entegresine ait katalog bilgileri Şekil 3.4'de belirtildiği gibidir.



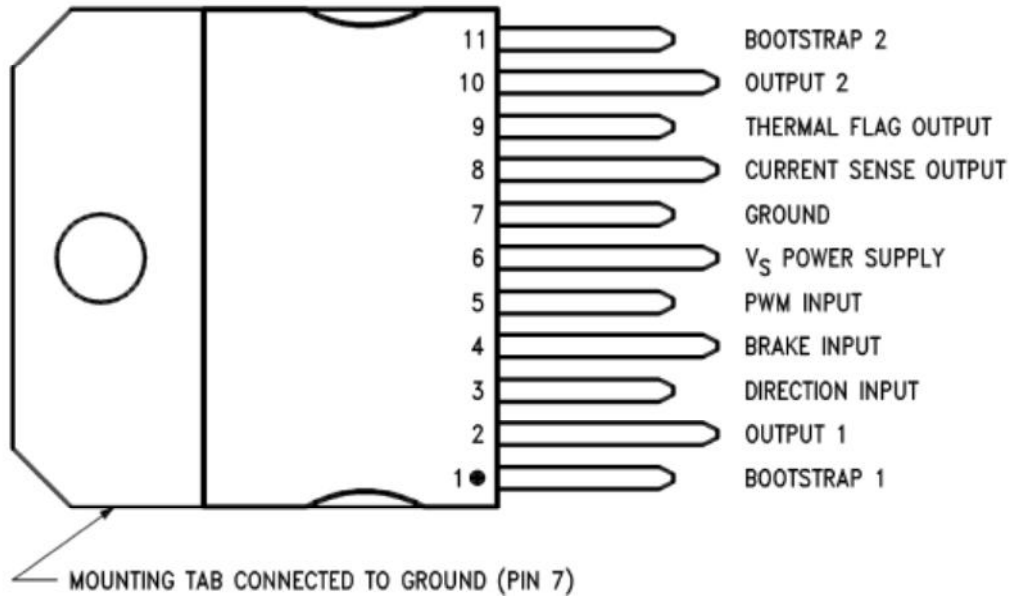
Şekil 3.4: TLP521-4 entegresi [47]

2)Bu bölümde, optokuplör entegresinin çıkışlarını 5V'a çekebilmek için pilden gelen 24V'luk gerilimi 5V'a düşüren bir adet LM7805 regülatör entegresi bulunmaktadır. Bu entegrenin çıkışından alınan 5V optokuplör entegresinin transistör tarafının kollektör pinine verilmiştir. LM7805 entegresine ait katalog bilgileri Şekil 3.5'de belirtildiği gibidir.



Şekil 3.5: 7805 entegresi [48]

3)Bu bölümde, mobil robotun hareketini sağlayan D.C. motorların hızını ve hareket yönünü görüntü işleme kartından gelen timer çıkışlarıyla kontrol etmek amacıyla LMD18200T entegresi kullanılmıştır. Bu entegrenin kullanılmasının nedenleri ise yüksek frekanslarda çalışabilmesi, motorların çekebileceği maksimum akıma dayanabilmesidir.LMD18200T entegresine ait katalog bilgileri Şekil 3.6'da belirtildiği gibidir.



Şekil 3.6: LMD18200T entegresi [49]

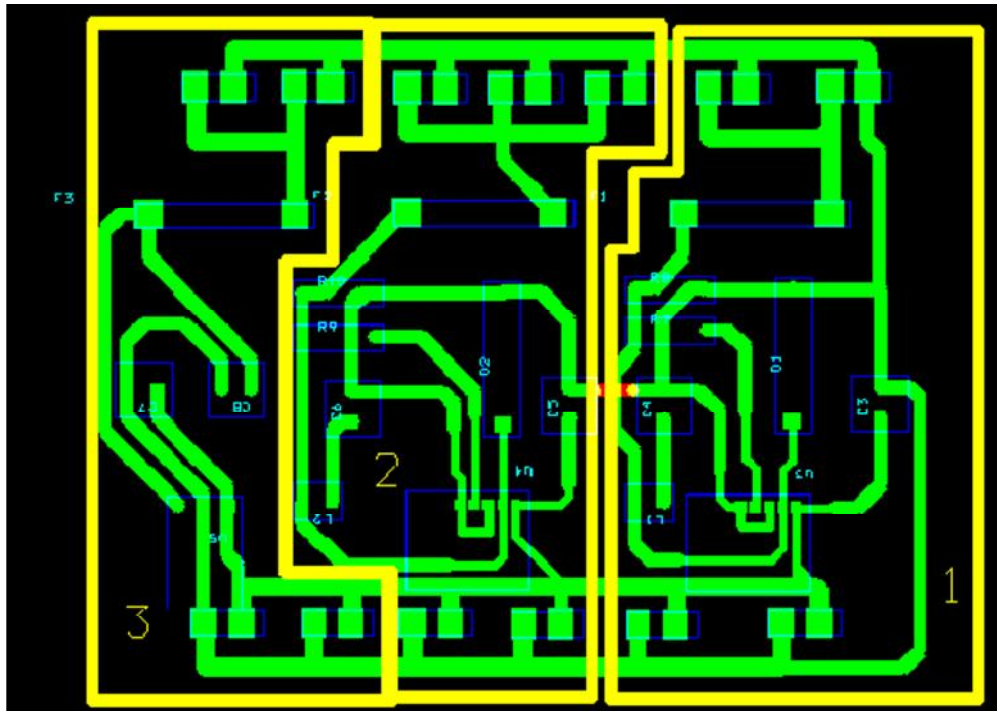
3.2.2. Regülatör devresi ve PCB tasarımı

Regülatör devresi, pilden elde edilen 24V'luk gerilimi, geliştirilen mobil robot sisteminin kontrolünü sağlayan görüntü işleme kartı, kamera, LCD ve şerit LED'lerin ihtiyaç duyduğu gerilimlere dönüştürmek için kullanılmıştır. Devre üzerinde kamera, LCD ve şerit LED'lerin beslemesi için bir adet 12V, görüntü işleme kartının beslemesi için bir adet 7V ve genel amaçlı kullanım için bir adet 9V olmak üzere üç adet çıkış bulunmaktadır. Her bir voltaj değerine birden çok ihtiyaç duyulacağı için çıkışlar iki adet klemenden verilmiştir. Ayrıca motor sürücü devrelerine gidecek olan pil gerilimi de bu devre üzerinde çoğullanmıştır. Devrenin ısınmasını önlemek amacıyla bir adet alüminyum soğutucu ile beraber devrenin tam üst hizasında kutunun üst kısmında yer alan iki adet fan kullanılmıştır.

Regülatör devresi Şekil 3.8'den görülebileceği üzere, anlaşılır bir şekilde açıklanmak amacıyla PCB çizim üzerinde bölümlere ayrılıp her bir bölüm ayrı ayrı numaralandırılmıştır. Şekil 3.7'de devrenin orijinal hali verilmiştir.



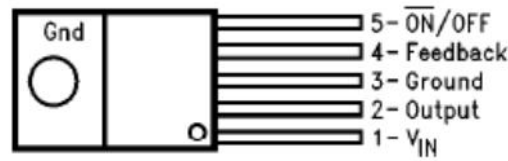
Şekil 3.7: Regülatör devresi



Şekil 3.8: Regülatör devresi PCB çizimi

Bu bölümlerin işlevleri ve kullanılan devre elemanlarının özellikleri aşağıdaki gibidir;

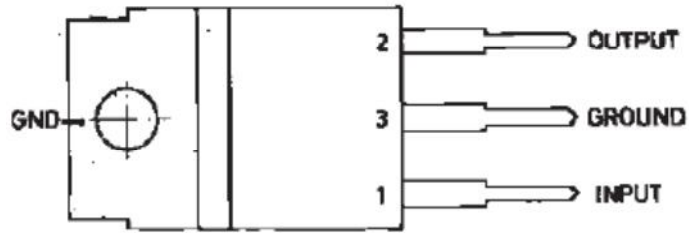
1) Bu bölümde, pilden gelen 24'luk gerilim görüntü işleme kartının ihtiyaç duyduğu 7V'luk gerilime dönüştürülmüştür. Direk 7V'luk özel bir regülatör entegresi olmadığı için ayarlanabilir bir regülatör olan LM2576-ADJ entegresi kullanılmıştır. Çıkış voltajının ayarlanması ise LM2576-ADJ kataloğunda yer alan formüle göre elde edilen R_1 ve R_2 ve dirençlerinin oranıyla sağlanmıştır. Bu entegrenin kullanılmasının nedenleri ise kartın çalışmak için ihtiyaç duyduğu akıma dayanabilmesi, switch mod çalıştığı için lineer regülatörlere göre daha az ısınması olarak gösterilebilir. Ayrıca regülatör çıkışlarına seri olarak bir adet 2.5A'lık sigorta bağlanmıştır. Bu sigorta ile görüntü işleme kartının olası bir aşırı akımdan korunması amaçlanmıştır. LM2576-ADJ entegresine ait katalog bilgileri Şekil 3.9'da belirtildiği gibidir.



Şekil 3.9 : LM2576 entegresi [50]

2) Bu bölümde, pilden gelen 24'luk gerilim kamera, LCD, şerit LED'lerin ihtiyaç duyduğu 12V'luk gerilime dönüştürülmüştür. Direk 12V'luk özel bir regülatör entegresi olmadığı için 7V'da olduğu gibi ayarlanabilir bir regülatör olan LM2576-ADJ entegresi kullanılmıştır. Çıkış voltajının ayarlanması ise LM2576-ADJ kataloğunda yer alan formüle göre elde edilen R_1 ve R_2 ve dirençlerinin oranıyla sağlanmıştır. Bu entegrenin kullanılmasının nedenleri ise kameranın, LCD, şerit LED'lerin çalışmak için ihtiyaç duyduğu akıma dayanabilmesi, switch mod çalıştığı için lineer regülatörlere göre daha az ısınması olarak gösterilebilir. Ayrıca regülatör çıkışlarına seri olarak bir adet 2.5A'lık sigorta bağlanmıştır. Bu sigorta ile regülatörün çıkışına bağlı elemanların olası bir aşırı akımdan korunması amaçlanmıştır. LM2576-ADJ entegresine ait katalog bilgileri Şekil 3.9'da belirtildiği gibidir.

3) Bu bölümde, pilden gelen 24'luk genel amaçlı kullanım için 9V'luk gerilime dönüştürülmüştür. Direk 9V'luk bir gerilim regülatörü olan LM7809 entegresi kullanılmıştır. Bu entegrenin kullanılmasının nedeni ise kolay kurulabilir olması LM2576 entegresine göre daha az eleman içermesi ve genel amaçlı kullanımda çok fazla akıma ihtiyaç duyulmamasıdır. Ayrıca regülatör çıkışlarına seri olarak bir adet 0.5A'lık seri bir sigorta bağlanmıştır. LM7809 entegresine ait katalog bilgileri Şekil 3.10'da belirtildiği gibidir.



Şekil 3.10: LM7809 entegresi [51]

3.2.3. BF561 işlemcisi

ADSP BF-561 işlemcisi Analog Devices firmasının ürettiği çift çekirdeğe sahip bir işlemcidir. Bu çift çekirdek sayesinde işlemcide yapılan işlemler, işlemci tarafından çok hızlı bir şekilde gerçekleştirilebilir. Bu işlemcinin tek dezavantajı olarak işlemci tipinin sabit noktalı işlemci olması gösterilebilir.

İşlemciye ait bazı temel özellikler aşağıda sıralanmıştır.

- Her iki çekirdek birbirinden bağımsız olarak 600MHZ hızında çalışabilir.
- 328 KB çip üzerinde dâhili hafıza
- Her bir çekirdek için 64 KB L1 veri hafızası
- Her bir çekirdek için 32 KB L1 komut hafızası
- Ortak kullanım için 128 KB paylaşılmış L2 hafızası
- 24 adet Direkt Hafız Erişimi (DMA)
- 12 tane genel amaçlı 32 bit timer
- 48 adet GPIO
- Haberleşme için SPI ve UART portları

- 0.5x ‘den 64x’e çalışma frekansını arttırmak için kullanılan PLL
- 2 adet PPI

PPI ile video encoderden alınan çıkışlar PPI ile okunup DMA kullanılarak hafızaya yazılmıştır. Ayrıca timerlar da mobil robotun hareketini sağlayan motorların kontrolünün sağlanması amacıyla PWM oluşturmak için kullanılmıştır.

Bu işlemci çift çekirdek olduğu için her iki çekirdek birbirinden ayrı olarak programlanabilir. Bir çekirdek görüntü işleme işlemlerini yaparken diğer çekirdeğin görüntünün alınması ve ekrana yazdırılması işlemlerini yapabilmektedir. Bu da tez çalışması sırasında yapılan görüntü işlemenin doğru sonuçlar verip vermediğini görmek ve yanlışa hangi parametrelerin değiştirileceğini anlamak için kullanılmıştır.

3.2.4. Görüntü işleme kartı (ADSP BF561 EZ-KIT LITE)

ADSP BF-561 EZ-KIT LITE geliştirme kartı Analog Devices firmasının ürettiği içerisinde BF-561 işlemcisi bulunan bir geliştirme kartıdır. Bu kart sayesinde bir bilgisayarda Visual DSP ortamında geliştirilen programlar bir USB kablosu ile BF-561 işlemcisi üzerinde çalıştırılabilir.

Kart üzerinde video ve sinyallerinin alınması ve verilmesi için konnektörler bulunmaktadır. Konnektörlerden alınan veya verilecek olan sinyalleri dijitalden analoga veya analogdan dijitalle dönüştürmede kullanılan encoder ve decoder entegreleri bulunmaktadır. Analog olarak alınan görüntü ve ses sinyalleri bu decoder entegreleri kullanılarak sayısala dönüştürülerek bu sinyaller işlemcinin işleyebileceği hale getirilmiştir. Aşağıda geliştirme kartının bazı özellikleri ve kart üzerinde yer alan elemanlar verilmiştir.

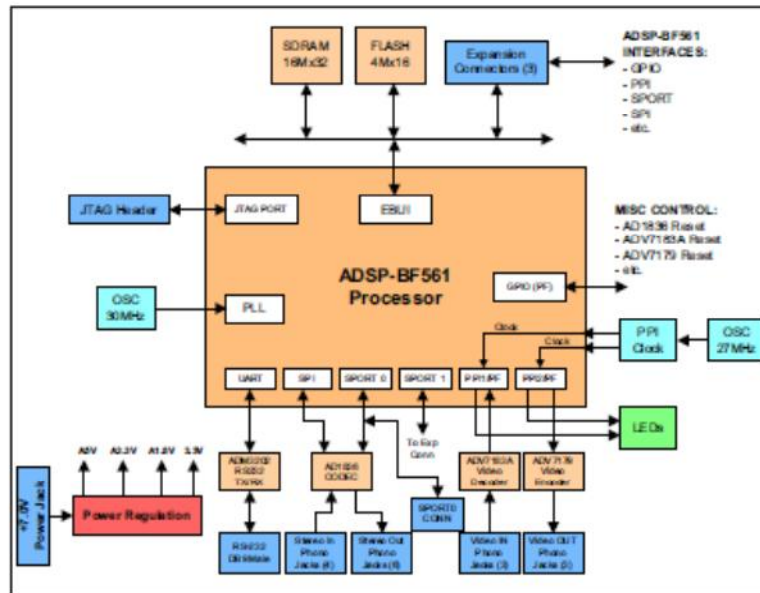
- JTAG ICE 14-pin
- ADV7183A video encoder
- ADV7179 video decoder
- 20 LED (1 power, 1 reset, 1 USB, 16 turuncu)
- 5 Button (1’i reset 4’ü programlanabilir)
- 1 adet rs232 çıkışı

- 7V adaptör girişi ve on board regülatör
- Bilgisayara bağlantı için USB girişi
- 90 pin genişleme arayüzü
- 256 MB SDRAM

Bu geliştirme kartının resmi Şekil 3.11’de verilmiştir. Ayrıca yine bu geliştirme kartının blok şeması da Şekil 3.12’de verilmiştir.



Şekil 3.11: ADSP BF-561 EZ-KIT LITE [52]



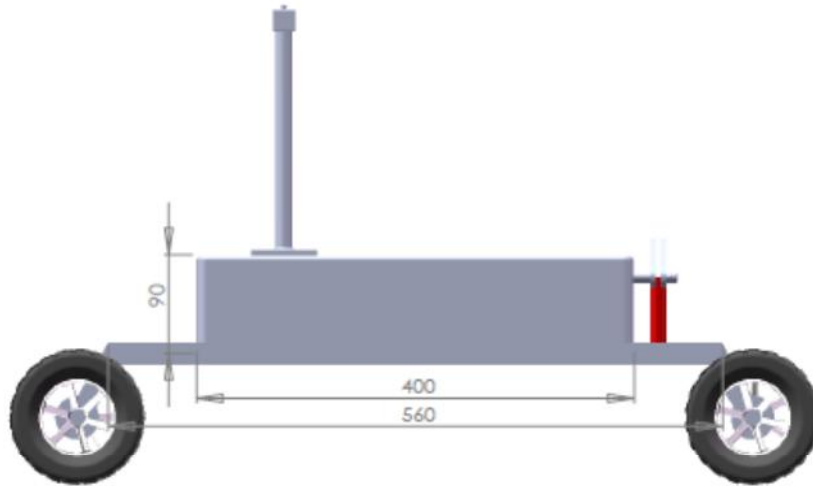
Şekil 3.12: ADSP BF-561 EZ-KIT LITE Şeması [52]

3.3. Mekanik Tasarım

Geliştirilen mobil robot platformunun mekanik kısımlar üç parça halinde toplanmıştır. İlk parça görüntü işleme kartı, motor sürücü devreleri, regülatör devresini ve sistemin enerjisinin sağlayan pili içerisinde bulunduran plastik kutudur.

Geliştirilen mobil robot platformunun ana gövdesi ise alüminyumdan yapılmış olup plastik kutu ve motor tutucuları bu ana gövdenin üzerine monte edilmiştir. Kutu ana gövde üzerine vida ile montajlanırken motor tutucular alüminyum kaynak yapılmıştır. Ayrıca ana gövdede mobil robotun iki tarafının birbirinden bağımsız hareket etmesini sağlayan iki adet amortisör bulunmaktadır. Bu sayede mobil robot sağ ve sol tarafta birbirinden bağımsız olarak yukarı yönde hareket edebilmektedir. Bu sayede mobil robotun engellerden kolayca geçebildiği görülmüştür.

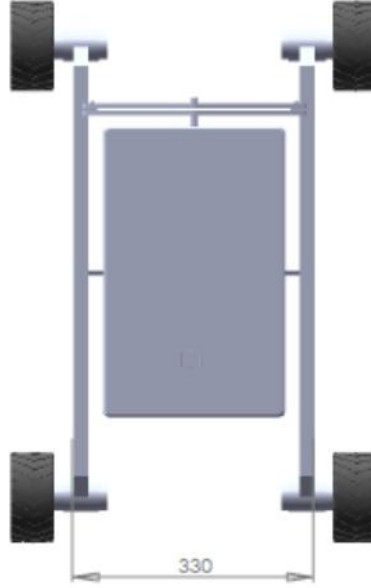
Son olarak da mobil robotun hareketini sağlayan motorların sabit tutulmasını sağlayan yarım yuvarlak şeklinde parçalar geliştirilmiştir. Mobil robotun mekanik çizimi Şekil 3.13’de gösterilmiştir.



Şekil 3.13: Mobil robotun mekanik çizimi

3.3.1. Elektronik kutu

40*25*9 cm ölçülerinde olan bu kutu 4009 elektronik cihaz kutusu olarak da geçmekte ve elektronik devre kutu satan yerlerden temin edilebilmektedir. Kutunun plastik olması kolay devrelerin daha kolay montajlanmasını sağlamıştır. Şekil 3.13’de elektronik kutu gösterilmiştir.



Şekil 3.14: Elektronik Kutu

3.3.2 Ana gövde

Geliştirilen mobil robot platformunun ana gövdesi ise alüminyumdan yapılmış olup plastik kutu ve motor tutucuları bu ana gövdenin üzerine monte edilmiştir.

3.3.3. Motor tutucuları

Alüminyumdan oluşan bu parçalar motorların gövdelerinin sabit kalmasını sağlarken aynı zamanda alüminyum olduğundan dolayı motorların ısınmasını da önlemektedir.

4. DENEYSEL SONUÇLAR

4.1. Giriş

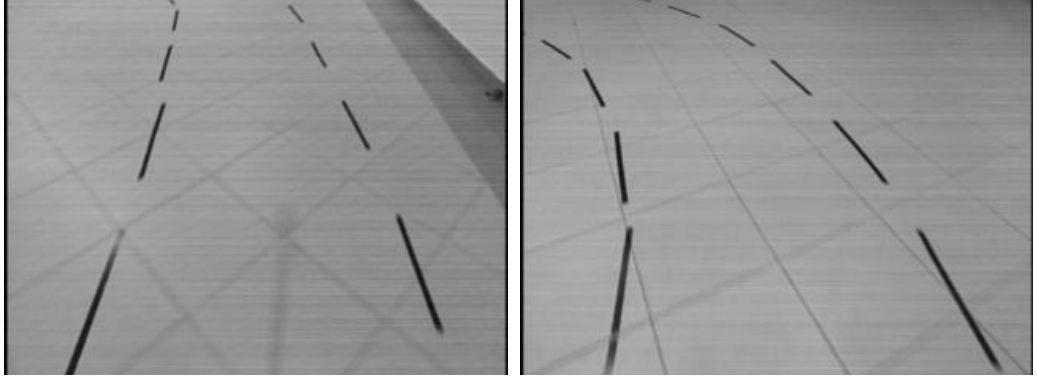
Bu bölümde, tez çalışması sırasında geliştirilen kod değişik ortam koşullarında denenmiş ve sonuçlar gösterilmiştir.

Geliştirilen sistem sırasıyla şerit çizgilerinin kesikli çizgiler olması , ortamdaki ışık şiddetinin az olması , ortamdaki ışık şiddetinin fazla olması durumlarında denenmiştir. Elde edilen sonuçlar sırasıyla Bölüm 4.2 , 4.3 ve 4.4’de gösterilmiştir.

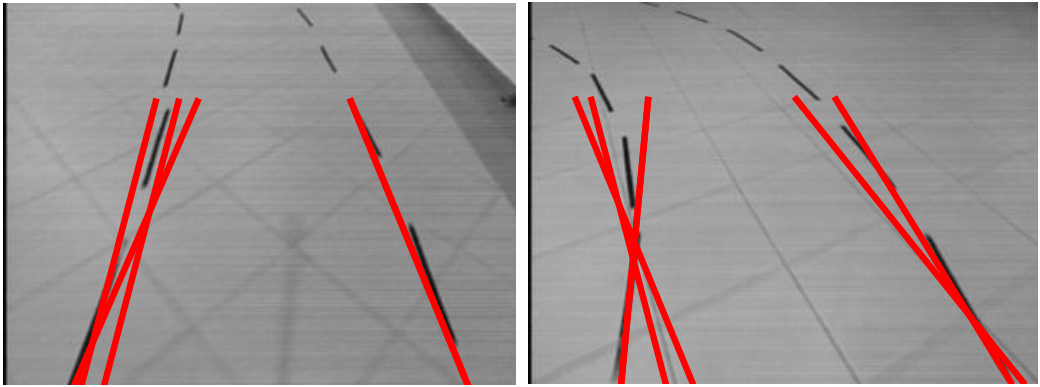
Geliştirilen sistem daha sonrasında şerit çizgilerinden bir tek şerit çizgisi olması durumunda denenmiştir. Gerçek yollarda şerit çizgilerinden sadece bir tanesi olabilir ve bu durum algoritmanın testinde önemli bir aşamadır. Sağda veya solda tek şerit olması durumu karşısında algoritmanın elde ettiği sonuçlar bölüm 4.5’de anlatılmıştır.

4.2. Şerit Çizgilerinin Kesikli Olması Durumu

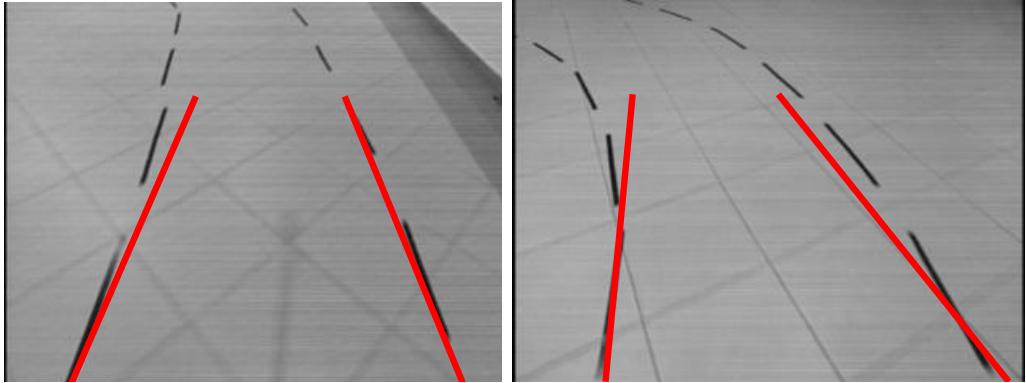
Tez çalışması sırasında, yapılan deneylerden biri de şerit çizgilerinin kesikli çizgilerden oluşması durumudur. Eğer şerit çizgileri kesikli çizgilerden oluşursa algoritmanın yanılıp yanılmadığı denenmiştir. Kesikli çizgilerin boyları kısa olduğu için yol üzerindeki yer alan karoları da şerit bilgisi olarak görebileceğinden algoritmanın yanılması ve mobil robotun şeritler dışına çıkması olası bir durumdur. Fakat kod geliştirilirken kullanılan Hough dönüşümü, aynı doğru üzerindeki bütün pikselleri belirlediği için boşluklardan ve gürültüden az etkilenmektedir. Şekil 4.1’de şerit çizgilerinin kesikli olması durumu için elde edilen orijinal imgeler sırasıyla verilmiştir. Şekil 4.2’de bu orijinal imgeler için elde edilen bütün Hough çizgileri sırasıyla verilmiştir. Şekil 4.3’de aynı orijinal imgeler için bulunan şerit çizgileri gösterilmektedir.



Şekil 4.1: Şerit çizgilerinin kesikli olması durumunda alınan iki örnek görüntü



Şekil 4.2: Kesikli şerit çizgileri için alınan iki örnek görüntüden elde edilen bütün Hough çizgileri



Şekil 4.3: Kesikli şerit çizgileri için alınan iki örnek görüntüden eliminasyon sonrası tespit edilen şerit çizgileri

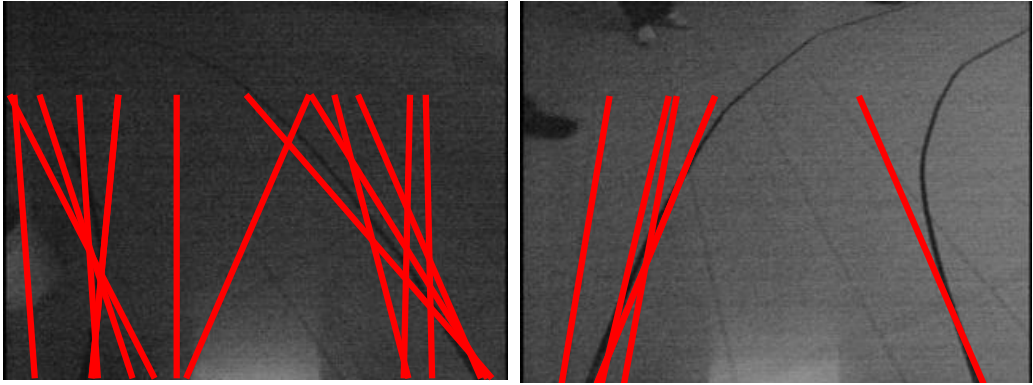
Görüldüğü gibi yolun kesikli çizgilerden oluşması durumunda dahi algoritma başarılı olarak çalışmakta ve mobil robot belirlenen çizgilerin dışarısına çıkmamaktadır.

4.3. Ortamdaki Işık Şiddetinin Az Olması Durumu

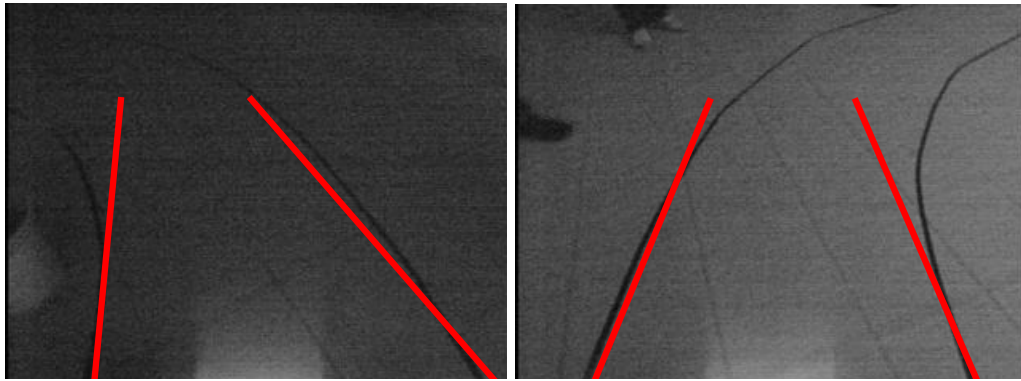
Tez çalışması sırasında, yapılan deneylerden biri de ortamdaki ışık şiddetinin az olması durumudur. Eğer ortamdaki ışık şiddeti az olursa algoritmanın yanılıp yanılmadığı denenmiştir.



Şekil 4.4: Ortamda yetersiz ışık olması durumunda alınan iki örnek görüntü



Şekil 4.5: Ortamda yetersiz ışık olması durumu için alınan iki örnek görüntüden elde edilen bütün Hough çizgileri



Şekil 4.6: Ortamda yetersiz ışık olması durumu için alınan iki örnek görüntüden eliminasyon sonrası tespit edilen şerit çizgileri

Yetersiz ışık altında ortamdaki şerit çizgileri siyah olduğu için görünmez bir hal alacağı için algoritmanın yanılması ve mobil robotun şeritler dışına çıkması olası bir durumdur. Ayrıca ortamdaki ışık şiddeti az olduğu için yoldaki diğer cisimler de siyah olarak görüneceğinden dolayı diğer cisimler de şerit çizgisi olarak görünebilir. Ayrıca bu yüzden bulunan doğru miktarı diğer durumlara göre daha fazla olmaktadır.

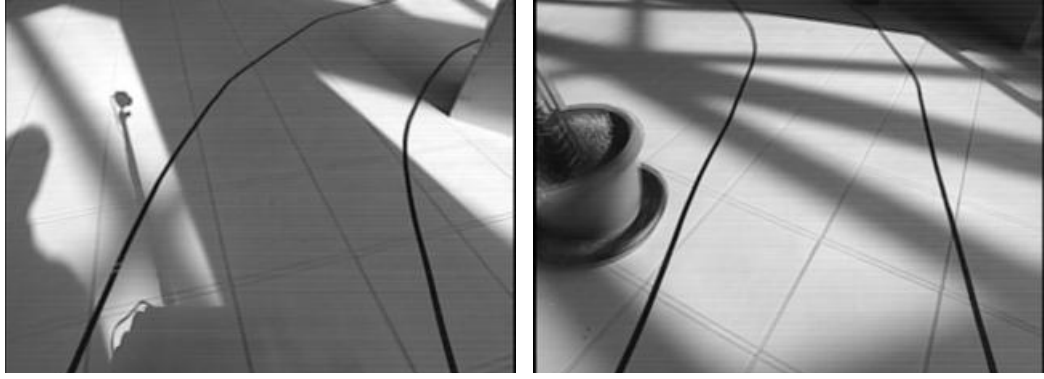
Şekil 4.4’de elde edilen orijinal imgeler sırasıyla gösterilmiştir. Şekil 4.5’de bu orijinal imgeler için elde edilen bütün Hough çizgileri sırasıyla gösterilmiştir. Şekil 4.6’da eliminasyondan sonra elde edilen şerit çizgileri görünmektedir. Şekillerde görüldüğü gibi ortamdaki ışık şiddeti az olursa, gürültüden dolayı görüntüde oluşan siyah noktalar da şerit çizgisi olarak görünmektedir. Bu yüzden şerit çizgisi olmaya aday çizgiler diğer durumlara göre sayıca daha fazla olmaktadır. Eliminasyonun başarılı bir şekilde çalıştığı görülmektedir.

Görüldüğü gibi ortamdaki ışık miktarının az olması durumunda da algoritma başarılı olarak çalışmakta ve mobil robot belirlenen çizgilerin dışarısına çıkmamaktadır.

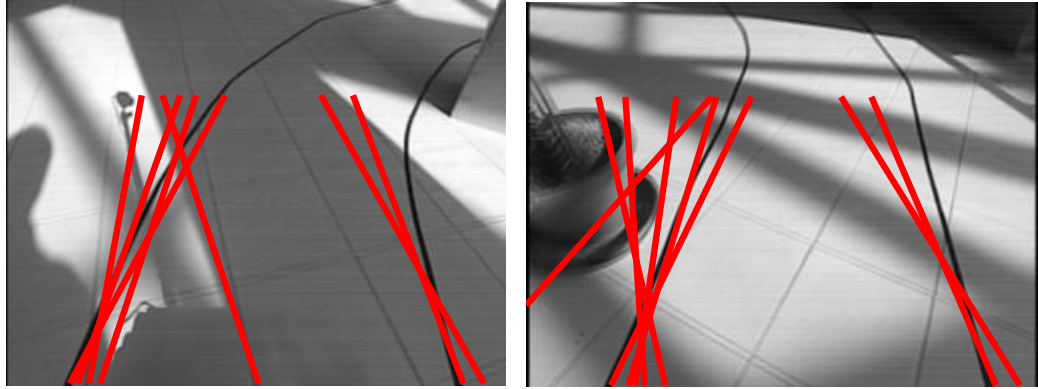
4.4. Ortamda Fazla Işık Olması Durumu

Tez çalışması sırasında, yapılan deneylerden biri de ortamdaki ışık şiddetinin fazla olması durumudur. Eğer ortamdaki ışık şiddeti fazla olursa algoritmanın yanılıp yanılmadığı denenmiştir. Fazla ışık altında ortamdaki çizgiler parladığı için, çizgiler görünmez bir hal alabilir ve bu da algoritmanın yanılmasına ve mobil robotun şeritler dışına çıkmasına neden olabilir. Ayrıca bu ışık şiddetinin fazla olması durumu cisimlerin yol üzerinde gölgelerinin oluşmasına neden olur. Bu durumda oluşan gölgeler de şerit çizgisi olarak görünebilir.

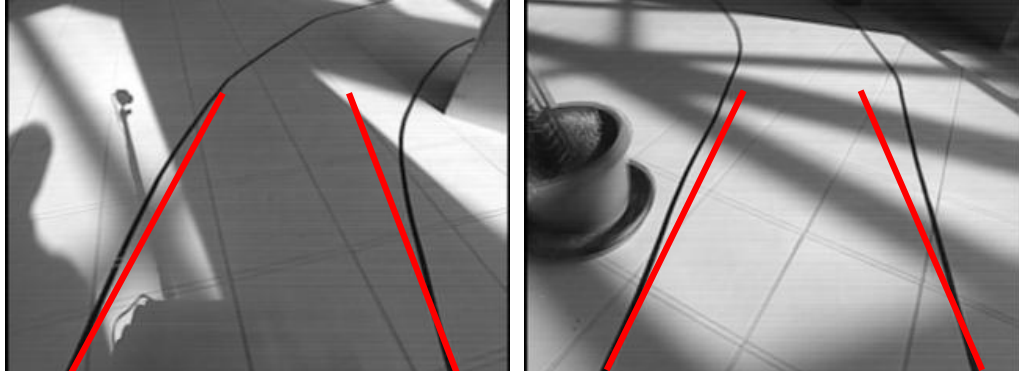
Şekil 4.7’de orijinal imgeler sırasıyla gösterilmiştir. Şekil 4.8’de verilen orijinal imgeler için elde edilen bütün Hough çizgileri sırasıyla verilmiştir. Şekil 4.9’da eliminasyondan sonra elde edilen şerit çizgileri görünmektedir.



Şekil 4.7: Ortamda fazla ışık olması durumunda alınan iki örnek görüntü



Şekil 4.8: Ortamda fazla ışık olması durumu için alınan iki örnek görüntüden elde edilen bütün Hough çizgileri



Şekil 4.9: Ortamda fazla ışık olması durumu için alınan iki örnek görüntüden eliminasyon sonrası tespit edilen şerit çizgileri

Görüntülerde bulunan yoldaki gölgelere ve çok sayıda parlamalara rağmen algoritma başarılı bir biçimde şerit bilgisini bulabilmektedir. Görüldüğü gibi ortamdaki ışık miktarı fazla olması durumunda dahi algoritma başarılı olarak çalışmakta ve mobil robot belirlenen çizgilerin dışarısına çıkmamaktadır.

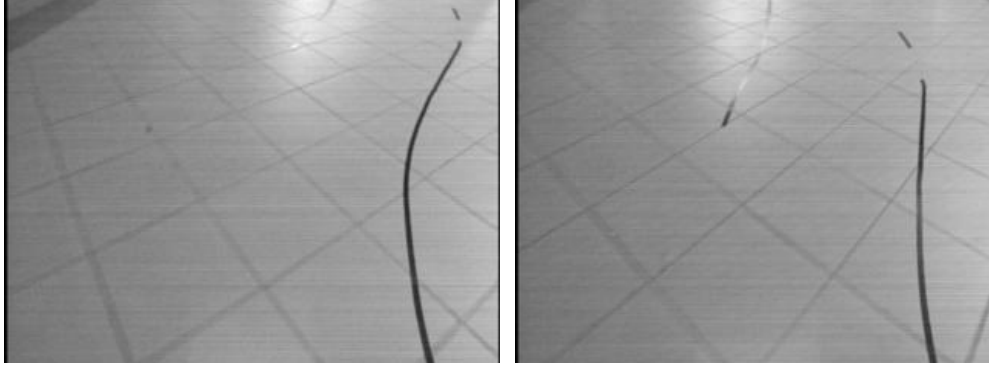
4.5. Tek Şerit Olması Durumu

Tez çalışması sırasında, yapılan deneylerden biri de şerit çizgilerinden birinin eksik olması durumudur. Yol üzerinde bulunan şerit çizgilerinden biri olmadığında algoritmanın yanılıp yanılmadığı denemiştir. Yolda tek şerit olması durumunda algoritma yolda öncelikli olarak iki şerit aradığı için, yol üzerinde bulunan karoları ikinci şerit olarak kabul edebilir ve bu da algoritmanın yanılmasına ve mobil robotun şeritler dışına çıkmasına neden olabilir.

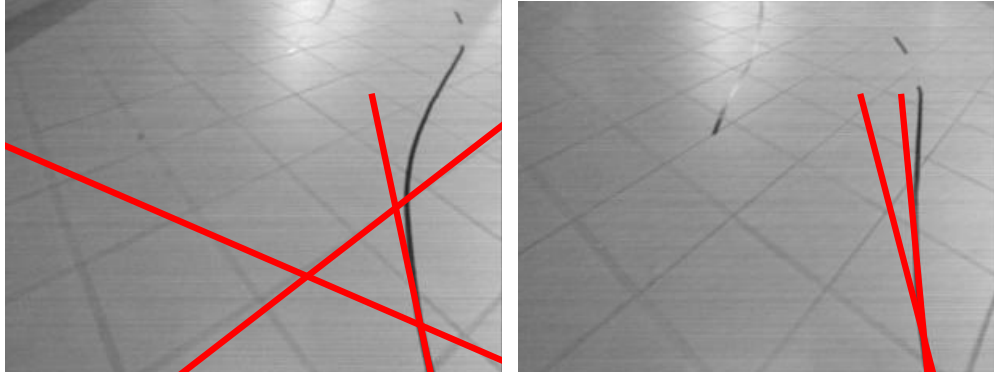
Yolda tek şerit olması durumunda algoritma yoldaki şerit çizgisinin sağ şerit çizgisi mi sol şerit çizgisi mi olduğuna karar verir. Bunun için doğrunun alt kesim noktasının gerçek dünya koordinatlarındaki yeri kontrol edilir. Bulunan bu nokta robotun solunda kalıyor ise bulunan şerit çizgisinin sol şerit, aksi halde sağ şerit olduğu kabul edilir. Eğer bulunan şerit sağ şerit ise algoritma şerit uzunluğu kadar solda, bulduğu şerit çizgisine paralel bir çizgi olduğunu kabul eder ve yolu ortalamaya çalışır. Eğer bulunan şerit sol şerit ise algoritma şerit uzunluğu kadar sağ, bulduğu şerit çizgisine paralel bir çizgi olduğunu kabul eder ve yolu ortalamaya çalışır.

4.5.1. Sağda tek şerit olması durumu

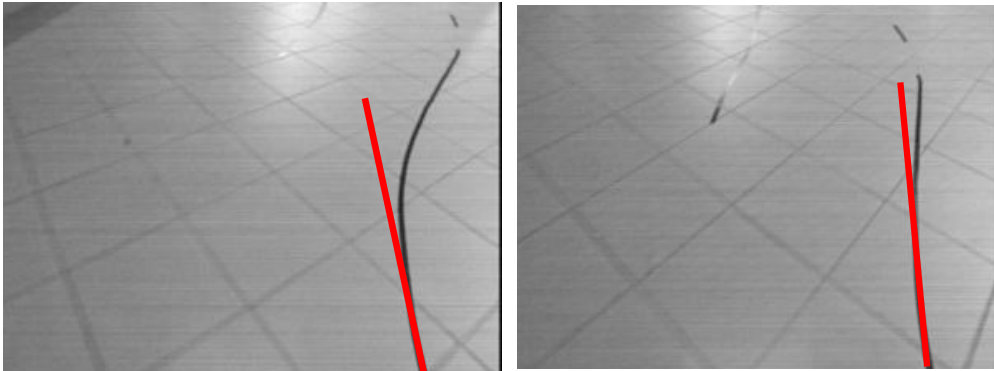
Şekil 4.10'da orijinal imgeler sırasıyla gösterilmiştir., Şekil 4.11'de bu orijinal imgeler için elde edilen bütün Hough çizgileri sırasıyla gösterilmiştir. Şekil 4.12'de eliminasyondan sonra elde edilen şerit çizgileri görünmektedir. Görüldüğü gibi yolda tek şerit çizgisi olması durumunda dahi algoritma başarılı olarak çalışmakta ve mobil robot belirlenen çizginin dışına çıkmamaktadır.



Şekil 4.10: Sağda tek şerit olması durumunda alınan iki örnek görüntü



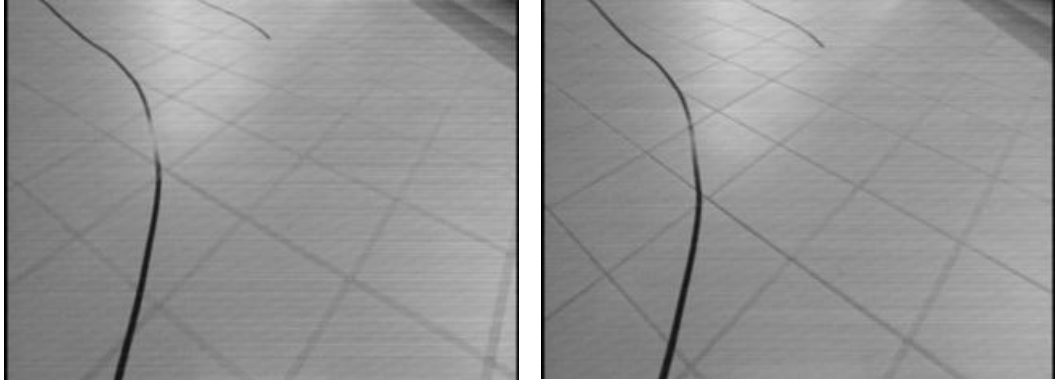
Şekil 4.11: Sağda tek şerit olması durumu için alınan iki örnek görüntüden elde edilen bütün Hough çizgileri



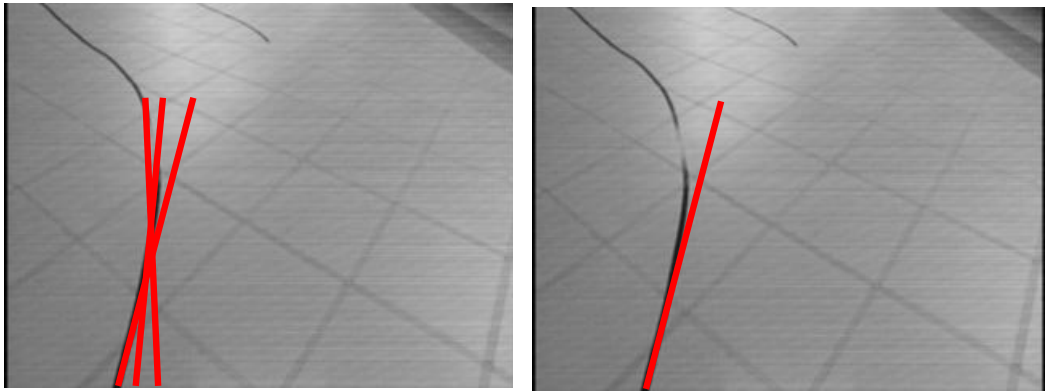
Şekil 4.12: Sağda tek şerit olması durumu için alınan iki örnek görüntüden eliminasyon sonrası tespit edilen şerit çizgileri

4.5.2. Solda tek şerit olması durumu

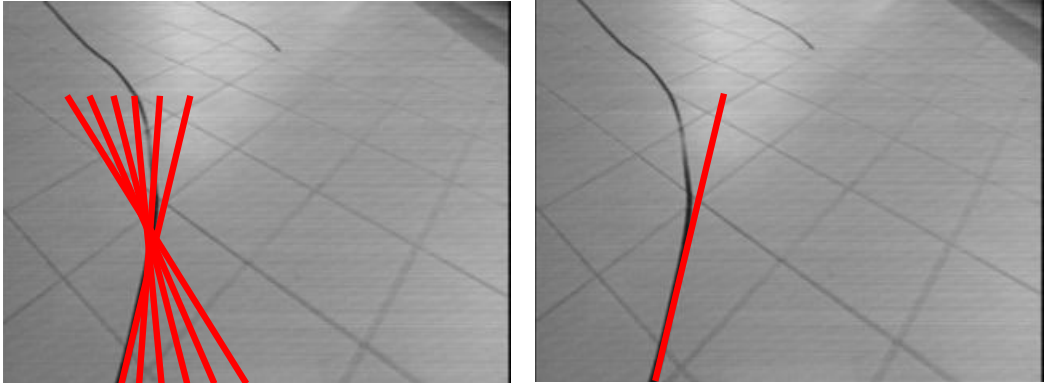
Şekil 4.13’de orijinal imgeler sırasıyla gösterilmiştir. Şekil 4.14’de bu orijinal imgeler için elde edilen bütün Hough çizgileri verilmiştir. Şekil 4.15’de eliminasyondan sonra elde edilen şerit çizgileri görünmektedir.



Şekil 4.13: Solda tek şerit olması durumunda alınan iki örnek görüntü



Şekil 4.14: Solda tek şerit olması durumu için alınan iki örnek görüntüden elde edilen bütün Hough çizgileri



Şekil 4.15: Solda tek şerit olması durumu için alınan iki örnek görüntüden eliminasyon sonrası tespit edilen şerit çizgileri

Görüldüğü gibi yolda tek şerit çizgisi olması durumunda dahi algoritma başarılı olarak çalışmakta ve mobil robot belirlenen çizginin dışarısına çıkmamaktadır.

SONUÇLAR ve ÖNERİLER

Bu bölümde, tez çalışması sırasında geliştirilen otonom hareket eden mobil robot platformunun elde ettiği sonuçlar ve sistemin geliştirilebilmesi için gelecekte neler yapılabileceği tartışılmıştır.

Tez çalışması sırasında geliştirilen otonom hareket eden mobil robot platformunun bütün ortamlarda yazılımsal olarak mükemmel bir şekilde çalıştığı, şerit çizgisini kaçırmadığı gözlenmiştir. Fakat geliştirilen sistemin mekanik olarak bazı sorunları bulunmaktadır. Bu bölümde bu sorunlara kısaca değinilecek olup sorunların çözümleri için bazı yöntemler sunulacaktır.

Geliştirilen mobil robot platformunun hareketini sağlayan DC motorların güçleri çok yüksek olmadığı için mobil robotun maksimum hızı 4 km/s olmaktadır. Daha güçlü DC motorlar takılarak mobil robotun hızı artırılabilir.

Mobil robotun engellerden rahat geçmesi için oluşturulan amortisörlü mekanizma gayet başarılı bir biçimde çalışmaktadır ve mobil robot karşısına çıkan engellerden rahatlıkla geçebilmekte hatta merdiven inip çıkabilmektedir. Fakat geçebileceği engellerin boyunu tekerleğin yarıçapı belirlemektedir. Daha yüksek çaptaki tekerlekler kullanılarak mobil robotun engellerden geçme kabiliyeti artırılabilir.

Mobil robotun ana kasası ve motor tutucuları alüminyumdan oluşturulmuştur. Bu elemanların birbirine alüminyum kaynak yapılarak tutturulmuştur. Alüminyum kaynak hassas bir kaynak türü olduğu için eksenler iyi tutturulamamış olup, eksenlerde bazı kaymalar meydana gelmiştir. Bu yüzden mobil robotun hareketini sağlayan motorlara aynı oranda PWM verildiğinde mobil robotun sağa doğru yönlendiği görülmüştür. Bunu çözmek için sağ taraftaki motorlara doluluk boşluk oranı daha yüksek olan PWM sinyali gönderilmiştir. Alüminyum yerine çelik malzeme kullanılarak ve lazerle çelik kaynak yapılarak bu sorun çözülebilir.

Mobil robotun enerjisi 24V 5A'lık bir LI-PO pil kullanılarak sağlanmıştır. Sistemdeki görüntü işleme kartı tek başına 2A çekmektedir ve her bir motor tek başına 1A çekmektedir. Bu da 5A olan pilin çabuk bitmesine yol açmaktadır. Amper değeri daha yüksek bir pil kullanılarak bu sorun giderilebilir.

Gelecekteki çalışmalarda mobil robot üzerine bir radar koyularak mobil robotun engelleri algılaması ve engellerden kaçması sağlanabilir. Ayrıca yine değişik sensörler kullanılarak mobil robotun konumu belirlenebilir ve mobil robota hedef verilip mobil robotun bu hedefe otonom olarak varması sağlanabilir.

KAYNAKLAR

- [1] Autonomous car http://en.wikipedia.org/wiki/Autonomous_car (**Ziyaret Tarihi: 22.11.2011**)
- [2] McGillem, C.D., Rappaport, T.S., "Infra-red Location System for Navigation of Autonomous Vehicles", *IEEE Transactions on vehicular technology*, 38, 132-139, (1988)
- [3] Dickmanns, E.D., Mysliwetz, B., Christians, T., "An Integrated Spatio-Temporal Approach to Automatic Visual Guidance of Autonomous Vehicles", *IEEE Transactions on Systems*, 20, 1273-1284, (1990)
- [4] Gregor, R., Lutzeler, M., Pellkofer, M., Siedersberger, K.-H., Dickmanns E.D., "A Perceptual System for Autonomous Vehicles", *IEEE Transactions on Intelligent Transportation Systems*, 3, 48-59, (2002)
- [5] Antonelli, G., Chiaverini, S., "Kinematic Control of Platoons of Autonomous Vehicles", *IEEE Transactions on Robotics*, 22, 1285-1292, (2006)
- [6] Frazzoli, E., Munther A.D., Feron, E., "Real-Time Motion Planning for Agile Autonomous Vehicles", *Journal Of Guidance Control And Dynamics*, 25, 116-129, (2002)
- [7] Krogh, B., Thorpe, C., "Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles", *Conference on Robotics and Automation Proceedings*, IEEE International, 1664-1669, April (1986)
- [8] Shiller, Z., Chen, J.C., "Optimal Motion Planning of Autonomous Vehicles in Three Dimensional Terrains", *Conference on Robotics and Automation*, IEEE International, 198-203 Cincinnati, 13-18 May (1990)
- [9] Kanayama, Y., Nilipour, A., Lelm, C.A., "A Locomotion Control Method For Autonomous Vehicles", *Conference on Robotics and Automation*, IEEE International, 1315-1317 Philadelphia, 13-18 Nis (1998)
- [10] Nelson, W., "Continuous-Curvature Paths for Autonomous Vehicles", *Conference on Robotics and Automation*, IEEE International, 1260-1264 Scottsdale, 14-19 May (1989)
- [11] Kanayama, Y., Kimura, Y., Miyazaki, F., Noguchi, T., "A Stable Tracking Control Method for an Autonomous Mobile Robot", *Conference on Robotics and Automation*, IEEE International, 384-389 Cincinnati, 13-18 May (1990)

- [12] Aguiar, A.P., Hespanha, J.P., “Trajectory-Tracking and Path-Following of Underactuated Autonomous Vehicles With Parametric Modeling Uncertainty”, *IEEE Transactions on Automatic Control*, 52, 1362-1379, (2007)
- [13] Gomez, B.F., Cuesta, F., Ollero, A., “Parallel And Diagonal Parking In Nonholonomic Autonomous Vehicles”, *Engineering Applications of Artificial Intelligence*(2000)
- [14] Vaidyanathan, R., Hocaoglu, C., Prince, T.S, Quinn R. D, “Evolutionary Path Planning For Autonomous Airvehicles Using Multiresolution Path Representation”, *Conference on Intelligent Robots and Systems*, IEEE International, 69-76 Maui, 29 Sep-03 Nov (2001)
- [15] Beard, R.W., McLain, T.W., Goodrich, M.A., Anderson, E.P., “Coordinated Target Assignment and Intercept for Unmanned Air Vehicles”, *Ieee Transactions On Robotics And Automation*, 18, 911-922, (2002)
- [16] Thomas, P.W., Richard, J.P., Andrew J.K., “Vision-Based State Estimation for Autonomous Micro Air Vehicles”, *Journal Of Guidance, Control, And Dynamics*, 30, 816-826, (2007)
- [17] Kanade, T., Amidi, O., Ke, Q., “Real-Time and 3D Vision for Autonomous Small and Micro Air Vehicles”, *Conference on Decision and Control*, IEEE International, 1655-1662 USA, 14-17 Dec (2004)
- [18]<http://www.nytimes.com/2010/10/10/science/10google.html?pagewanted=all>(**Ziyaret Tarihi: 22.11.2011**)
- [19]http://www.aselsan.com.tr/urun.asp?urun_id=236&lang=tr(**Ziyaret Tarihi: 22.11.2011**)
- [20] Qing, L., Zheng, N., Hong, C., “Springrobot: A Prototype Autonomous Vehicle And Its Algorithms For Lane Detection”. *IEEE Transactions on Automatic Control*, 5, 300-308, (2007)
- [21] Yu, B., Jain, A.K., “Lane Boundary Detection Using A Multiresolution Hough Transform”, *Conference on Image Processing*, IEEE International, 748-751 Santa Barbara, 26-29 Nov (1997)
- [22] McCall, J.C., Trivedi, M.M., “ Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation”, *IEEE Transactions on Intelligent Transportation Systems* , 7, 20-37, (2006)
- [23] Kang, D.J., Jung, M.H., “Road Lane Segmentation Using Dynamic Programming For Active Safety Vehicles”, *Pattern Recognitions Letters* , 24, 3177-3185, (2003)
- [24] Wang, Y., Teoh, E.K., Shen, D., “Lane detection and tracking using B-Snake”, *Image and Vision Computing*, 22, 269-280, (2004)

- [25] Mandalia, H.M., Salvucci D.D.M., “Using Support Vector Machines For Lane-Change Detection”, *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 49, 1965-1969, (2005)
- [26] Borkar, A., Hayes, M., Smith, M.T., “Robust Lane Detection And Tracking With Ransac And Kalman Filter”, *IEEE International Conference on Image Processing*, 7, 3261-3264, (2009)
- [27] Nedeveschi, S., Schmidt, R., Graf, T., Danescu, R., Frentiu, D., Marita, T., Oniga, F., Pocol, C., “3D Lane Detection System Based on Stereovision”, *IEEE Transactions on Intelligent Transportation Systems*, 161-166, (2004)
- [24] Wang, Y., Teoh, E.K., Shen, D., “Lane detection and using Catmull-Rom Spline”, *IEEE Transactions on Intelligent Vehicles*, 51-57, (1998)
- [29] Sun, T.Y., Tsai, S.J., Chan, V., “HSI Color Model Based Lane-Marking Detection”, *Conference on Intelligent Transportation Systems*, IEEE International, 1168-1172 Toronto, 17-20 Sept (2006)
- [30] Nedeveschi, S., Schmidt, R., Graf, T., Danescu, R., Frentiu, D., Marita, T., Oniga, F., Pocol, C., “3D Lane Detection System Based on Stereovision”, *Conference on Intelligent Transportation Systems*, IEEE International, 161-166, 03-06 Oct (2004)
- [31] Kreucher, C., Lakshmanan, S., Kluge, K., “A Driver Warning System Based on the LOIS Lane Detection Algorithm”, *Conference on Intelligent Transportation Systems*, IEEE International, 17-22, (1998)
- [32] Lai, A.H.S., Yung, N.H.C., “Lane Detection by Orientation and Length Discrimination”, *Ieee Transactions On Systems*, 30, 539-548, (2000)
- [33] Jochem, M.D., Pomerleau, D.A., Thorpe, E.C., “MANIAC: A Next Generation Neurally Based Autonomous Road Follower”, The Robotics Institute, Carnegie Mellon University, http://scholar.google.com.tr/scholar?hl=tr&q=%2C+MANIAC%3A+A+Next+Generation+Neurally+Based+Autonomous+Road+Follower&btnG=Ara&lr=&as_ylo=&as_vis=0 (Ziyaret Tarihi: 22.11.2011)
- [34] Demigny, D., “On Optimal Linear Filtering for Edge Detection”, *Ieee Transactions On Image Processing*, 11, 728-737, (2002)
- [35] Kanopoulos, N., Vasantahavada, N., Baker, R.L., “Design of an Image Edge Detection Filter Using the Sobel Operator”, *Ieee Journal Of Solid-State Circuits*, 23, 358-367, (1988)
- [36] Cho, S.Y., Chow, T.W. S., Leung C.T., “A Neural-Based Crowd Estimation by Hybrid Global Learning Algorithm”, *Ieee Transactions On Systems*, 29, 535-541, (1999)

- [37] Ballard, D.H., "Generalizing Hough Transform To Detect Arbitrary Shapes", Computer Science Department, University of Rochester, <http://www.sciencedirect.com/science/article/pii/0031320381900091> (Ziyaret Tarihi: 22.11.2011)
- [38] Ioannou, D., Huda, W., Laine, A.F., "Circle Recognition Through A 2D Hough Transform And Radius Histogramming", *Image and Vision Computing*, 17, 15-26 (1999)
- [39] Chen, T.C., Chung, K.L., "An Efficient Randomized Algorithm for Detecting Circles", *Computer Vision and Image Understanding*, 83, 172-191 (2001)
- [40] Lutton, E., Maitre, H., Lopez-Krahe, J., "Contribution to the Determination of Vanishing Points Using Hough Transform", *Ieee Transactions On Pattern Analysis And Machine Intelligence*, 16, 430-438, (1994)
- [41] Illingworth, J., Kittler, J., "The Adaptive Hough Transform", *Ieee Transactions On Pattern Analysis And Machine Intelligence*, 9, 690-698, (2009)
- [42] Chutatape, O., Guo, L., "A Modified Hough Transform For Line Detection And Its Performance", *Pattern Recognition*, 32, 181-192, (1998)
- [43] Bertozzi, M., Broggi, A., Fascioli, A., "Stereo Inverse Perspective Mapping: Theory And Applications", *Image and Vision Computing*, 16, 585-590 (2007)
- [44] Malot, H.A., Bultoff, H. H., Little, J.J., Bohrer, S., "Inverse Perspective Mapping Simplifies Optical Flow Computation And Obstacle Detection", *Biological Cybernetics*, 64 , 177-185 (1992)
- [45] Jiang, G.Y., Choi, T.Y., Hong, S.K., Bae, J.W., Song, B.S., " Lane and Obstacle Detection Based on Fast Inverse Perspective Mapping Algorithm", *Ieee Transactions On Cybernetics*, 4, 2969-2974, (2000)
- [46] Nandam, P.K., Sen, P.C., "Analog and Digital Speed Control of DC Drives Using Proportional-Integral and Integral-Proportional Control Techniques", *Ieee Transactions On Industrial Electronics*, 34, 227-223, (1987)
- [47] TLP521-4 http://www.datasheetcatalog.com/datasheets_pdf/T/L/P/5/TLP521-4.shtml (Ziyaret Tarihi: 22.11.2011)
- [48] 7805 http://www.datasheetcatalog.com/datasheets_pdf/7/8/0/5/7805.shtml (Ziyaret Tarihi: 22.11.2011)
- [49] LMD18200 <http://www.ti.com/general/docs/lit/getliteratue.tspgenericPartNumber=lmd18200®=en&fileType=pdf> (Ziyaret Tarihi: 22.11.2011)
- [50] LM2576 http://www.datasheetcatalog.com/datasheets_pdf/L/M/2/5/LM2576.sht (Ziyaret Tarihi: 22.11.2011)

[51] LM7809 http://www.datasheetcatalog.com/datasheets_pdf/L/M/7/8/LM7809.sht
(Ziyaret Tarihi: 22.11.2011)

[52] BF-561 EZKIT LITE <http://www.analog.com/en/processors-dsp/blackfin/bf561-ezlite/products/product.html> (Ziyaret Tarihi: 22.11.2011)

EKLER

EKA Geliştirilen Görüntü İşleme Kodu

```
#include "main.h"
#include "stdio.h"

// Horizontal and vertical blankinks (in NTSC/PAL frame) in bytes.
#define H_BLANK (unsigned int)
(ADI_ITU656_BLANKING+ADI_ITU656_EAV_SIZE+ADI_ITU656_SAV_SIZE)
#define V_BLANK (unsigned int) (ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH)

void convertToGrayScale(volatile short *ptrImage, unsigned char *ptrOut);
void extractCb(volatile short *ptrImage, unsigned char *ptrOut);
void extractCr(volatile short *ptrImage, unsigned char *ptrOut);
void Sobel_Edge(unsigned char *ptr);
void Initialize_Hough_Array(unsigned volatile short *ptrHT_Array);
void Hough_Transform(unsigned char *ptrIn, unsigned volatile short *ptrHT_Array);
void Hough_Transform_Non_Optimized(unsigned char *ptrIn, unsigned volatile short
*ptrHT_Array);
void Hough_Peaks(unsigned volatile short *ptrHT_Array);
void Find_Lane_Marks(void);
void Draw_Hough_Lines(volatile short *ptrOut, signed volatile char *theta, signed volatile short
*rho, unsigned volatile short nlines, unsigned char color);
void copy_ProcessedImage2Buffer(unsigned char *ptrIn, volatile short *ptrOut);
void deinterlaceFrame(volatile short *ptrIn);
void Imag_Thr(unsigned char *ptr, unsigned char thr);
void Navigate_Robot(void);
void Perspective_Correction(void);

void processFrame(volatile short *ptrImage) {

    unsigned short peak_count;          // # of peaks detected in Hough space

    // NTSC/PAL frame is resized and converted to gray-scaled image
    // Only active field 1 is kept
    // # of pixels per line (columns) is reduced by 2
    convertToGrayScale(ptrImage, &processedImage[0]);

    //extractCb(ptrImage, &processedImage[0]);
    //Sobel_Edge(&processedImage[0]);

    // initialize the accumulator array
    Initialize_Hough_Array(&houghArray[0]);

    // compute the Hough transform of the image
    Hough_Transform_Non_Optimized(&processedImage[0], &houghArray[0]);

    //Hough_Transform(&processedImage[0], &houghArray[0]);
```

```

// start searching the Hough space for peaks
Hough_Peaks(&houghArray[0]);

// find lane marks among the peaks located
Find_Lane_Marks();

// if dispVideo is set to 1 in system.h:
// Draw the lane marks over the image
if (dispVideo==1) {
    Draw_Hough_Lines(&sFrame0[0], &theta_peaks[0], &rho_peaks[0], N_peaks, 0);
    //Draw_Hough_Lines(&sFrame0[0]+(ADI_ITU656_ILF2_START-
ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH/2), &theta_peaks[0], &rho_peaks[0],
N_peaks, 0);
    //Draw_Hough_Lines(&sFrame1[0], &theta_peaks[0], &rho_peaks[0], N_peaks, 0);
    //Draw_Hough_Lines(&sFrame1[0]+(ADI_ITU656_ILF2_START-
ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH/2), &theta_peaks[0], &rho_peaks[0],
N_peaks, 0);
}

// perspective correction for the lanes detected
Perspective_Correction();

// if dispVideo is set to 1 in system.h:
// Draw the lane marks (perspective corrected) over the image
// Draw the robot's heading vector, too.
if (dispVideo==1) {
    Draw_Hough_Lines(&sFrame0[0], &theta_peaks[0], &rho_peaks[0], N_peaks, 1);
    //Draw_Hough_Lines(&sFrame0[0]+(ADI_ITU656_ILF2_START-
ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH/2), &theta_peaks[0], &rho_peaks[0],
N_peaks, 1);
    //Draw_Hough_Lines(&sFrame1[0], &theta_peaks[0], &rho_peaks[0], N_peaks, 1);
    //Draw_Hough_Lines(&sFrame1[0]+(ADI_ITU656_ILF2_START-
ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH/2), &theta_peaks[0], &rho_peaks[0],
N_peaks, 1);
    Draw_Hough_Lines(&sFrame0[0], &theta_peaks[2], &rho_peaks[2], 1, 2);
    //Draw_Hough_Lines(&sFrame0[0]+(ADI_ITU656_ILF2_START-
ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH/2), &theta_peaks[2], &rho_peaks[2], 1,
2);
    //Draw_Hough_Lines(&sFrame1[0], &theta_peaks[2], &rho_peaks[2], 1, 2);
    //Draw_Hough_Lines(&sFrame1[0]+(ADI_ITU656_ILF2_START-
ADI_ITU656_ILF1_START)*(ADI_ITU656_LINE_WIDTH/2), &theta_peaks[2], &rho_peaks[2], 1,
2);
}

// navigate
//Navigate_Robot();

}

```

```

/*****

NTSC/PAL frame is resized and converted to gray-scaled image
Only active field 1 is kept
# of pixels per line (columns) is reduced by 2
For NTSC frame the resulting image is a 240x360 gray-scaled image

*****/
void convertToGrayScale(volatile short *ptrImage, unsigned char *ptrOut) {

// ptrImage is pointer to the NTSC/PAL frame
// ptrOut is the pointer to the output image

    unsigned int i, j;
    u8 *ptrIn;
    int rows=N_Rows;
    int cols=N_Cols;

    // Skip vertical blanking.
    ptrIn=((u8 *)ptrImage) + 1 + V_BLANK;

    // Get the every other luminosity (Y) value.
    for (i=0; i<rows; i++) {
        ptrIn += H_BLANK;      // skip horizontal blanking
        for (j=0; j<cols; j++) {
            *(ptrOut++) = *(ptrIn);
            ptrIn += 4;          // // Y values are bytes: 2-4-6-8...(every other Y
is kept)
        }
    }

}

```

```

/*****

NTSC/PAL frame is resized and blue diffence (Cb) values are extracted
Only active field 1 is kept
# of pixels per line (columns) is reduced by 2
For NTSC frame the resulting image is a 240x360 gray-scaled image

*****/
void extractCb(volatile short *ptrImage, unsigned char *ptrOut) {

// ptrImage is pointer to the NTSC/PAL frame
// ptrOut is the pointer to the output image

    unsigned int i, j;
    u8 *ptrIn;
    int rows=N_Rows;
    int cols=N_Cols;

    // Skip vertical blanking.
    ptrIn=((u8 *)ptrImage) + V_BLANK;

    // Extract the blue difference (Cb) values
    for (i=0; i<rows; i++) {
        ptrIn += H_BLANK;      // skip horizontal blanking
        for (j=0; j<cols; j++) {
            *(ptrOut++) = *(ptrIn);
            ptrIn += 4;        // Cb is byte:1-5-9-...
        }
    }
}

```

```

/*****

NTSC/PAL frame is resized and red diffence (Cr) values are extracted
Only active field 1 is kept
# of pixels per line (columns) is reduced by 2
For NTSC frame the resulting image is a 240x360 gray-scaled image

*****/
void extractCr(volatile short *ptrImage, unsigned char *ptrOut) {

// ptrImage is pointer to the NTSC/PAL frame
// ptrOut is the pointer to the output image

    unsigned int i, j;
    u8 *ptrIn;
    int rows=N_Rows;
    int cols=N_Cols;

    // Skip vertical blanking.
    ptrIn=((u8 *)ptrImage) + 2 + V_BLANK;

    // Extract the red difference (Cr) values
    for (i=0; i<rows; i++) {
        ptrIn += H_BLANK;      // skip horizontal blanking
        for (j=0; j<cols; j++) {
            *(ptrOut++) = *(ptrIn);
            ptrIn += 4;        // Cr values are bytes: 3-7-11-...
        }
    }
}

```

```

/*****

```

Function :Sobel_Edge

Input Argument :(*ptr) pointer to the input image.

Return Argument :The sobel filtered image

Purpose :Filters an image with sobel filter

The output is written over the input image

```

*****/

```

```

void Sobel_Edge(unsigned char *ptr)
{

```

```

    int i;    //row index
    int j;    //column index

```

```

    signed char Gx, Gy;
    unsigned char preline[N_Cols];
    unsigned char left_pixel;
    unsigned char current_pixel;

```

```

    int rows = N_Rows;          // # of rows
    int cols = N_Cols;          // # of columns

```

```

    // Threshold value to be used in sobel filtered image to find edges

```

```

    // EDGE_THRESH is defined in system.h

```

```

    unsigned char edge_thr = EDGE_THRESH;

```

```

    // Edge pixels having a gray level above this threshold will be considered only

```

```

    // GRAY_THRESH is defined in system.h

```

```

    unsigned char gray_thr = GRAY_THRESH;

```

```

    for (i=0;i<cols;i++)
        preline[i]=*ptr++;

```

```

    left_pixel=*ptr;

```

```

    for (i=1;i<rows-1;i++)
    {
        for(j=1;j<cols-1;j++)
        {

```

```

            current_pixel = *(++ptr);

```

```

            Gy = (signed char) left_pixel-*(ptr+1);          // vertical filter [1 0

```

```

-1]

```

```

            //Gx = (signed char) preline[j]-*(ptr+cols); // horizontal filter [1 ; 0 ; -1]

```

```

            if (current_pixel<=gray_thr && abs(Gy)>=edge_thr)

```

```

                *ptr = 255;

```

```

            else

```

```

                *ptr = 0;

```

```

            preline[j-1] = left_pixel;
            left_pixel = current_pixel;

```

```

        }

```

```

        preline[cols-2]=left_pixel;

```

```

        preline[cols-1]=*(++ptr); ptr++;

```

```

    }
}

```

```
/******
```

Function :Initialize_Hough_Array

Input Argument :(*ptrHT_Array) pointer to the Hough Accumulator Array

Return Arguments :None

Purpose :Initializes all the values of the array to zero

```
*****/
```

```
void Initialize_Hough_Array(unsigned volatile short *ptrHT_Array)
```

```
{
    unsigned short n_rho = N_RHO;
    unsigned short n_theta = N_THETA;
    int i, j;
    for (i=0;i<n_rho;i++)
        for (j=0;j<n_theta;j++)
            *ptrHT_Array++ = 0;
}
```

```

/*****

```

Function :Hough_Transform

Input Argument :(*ptrIn) pointer to the input gray scaled image

Return Arguments :(*ptrHT_Array) pointer to the Hough Accumulator Array

Purpose :Computes the HT of a gray scaled image returning the accumulator array

```

*****/

```

```

void Hough_Transform(unsigned char *ptrIn, unsigned volatile short *ptrHT_Array)
{

```

```

    unsigned short int i;    //row index
    unsigned short int j;    //column index

```

```

    unsigned short houghArrayInd;

```

```

    unsigned short rows = N_Rows;    // # of rows
    unsigned short cols = N_Cols;    // # of columns

```

```

    // define the region to be processed
    unsigned short row_min = rows/2;
    unsigned short row_max = rows-1;
    unsigned short col_min = 10;
    unsigned short col_max = cols-col_min;

```

```

    signed char Gx, Gy;    // gradient in x and y directions
    float slope;    // Gy/Gx
    signed short int_slope;

```

```

    // Hough parameters defined in system.h
    signed short rho_min = RHO_MIN;
    signed short rho_max = RHO_MAX;
    unsigned char rho_res = RHO_RES;
    signed char theta_min = THETA_MIN;
    signed char theta_max = THETA_MAX;
    unsigned char theta_res = THETA_RES;
    unsigned short n_rho = N_RHO;
    unsigned short n_theta = N_THETA;

```

```

    // rho and theta indices and values
    unsigned short rho_ind, theta_ind;
    signed short rho;
    signed char theta;
    signed char theta_lower_limit, theta_upper_limit;

```

```

    // parameters for trig functions look-up tables
    unsigned short LUT_ind;
    signed short atan_max = ATAN_MAX;
    signed short atan_min = ATAN_MIN;
    float atan_res = ATAN_RES;

```

```

    // x and y values in the line equation
    unsigned short x, y;

```

```

    // max y value
    short ymax = rows-1;

```

```

    // Threshold value to be used in sobel filtered image to find edges
    // EDGE_THRESH is defined in system.h

```

```

unsigned char edge_thr = EDGE_THRESH;

// Edge pixels having a gray level below this threshold will be considered only
// GRAY_THRESH is defined in system.h
unsigned char gray_thr = GRAY_THRESH;

// set the pointer to the start of the upper half of the image
// first line is not processed
ptrIn += row_min*cols;

for (i=row_min;i<row_max;i++)
{
    ptrIn += col_min;

    for(j=col_min;j<col_max;j++)
    {
        if (*ptrIn<=gray_thr)                // for each pixel below gray threshold:
        {
            Gy = (signed char) *(ptrIn-1) - *(ptrIn+1); // vertical filter [1 0
-1]
            //Gy = (signed char) *(ptrIn) - *(ptrIn-1);
            //Gy = ((signed char) *(ptrIn-1-cols)-*(ptrIn+1-cols)) + ((signed
char) *(ptrIn-1)-*(ptrIn+1)) + ((signed char) *(ptrIn+1+cols)-*(ptrIn+1+cols));
            Gx = (signed char) *(ptrIn-cols) - *(ptrIn+cols);    // horizontal
filter [1 ; 0 ; -1]
            //Gx = (signed char) *(ptrIn) - *(ptrIn-cols);
            //Gx = ((signed char) *(ptrIn-1-cols)-*(ptrIn-1+cols)) + ((signed
char) *(ptrIn-cols)-*(ptrIn+cols)) + ((signed char) *(ptrIn+1-cols)-*(ptrIn+1+cols));

            if (abs(Gy)>=edge_thr)                // if the current pixel is an edge
pixel:
            {
                if (Gx==0)
                {
                    theta_lower_limit = -6;
                    theta_upper_limit = 6;
                }
                else
                {
                    slope = ((float) Gy)/Gx;
                    int_slope = slope;
                    if (int_slope>=atan_max || int_slope<=atan_min)
                    {
                        theta_lower_limit = -6;
                        theta_upper_limit = 6;
                    }
                    else
                    {
                        LUT_ind = (slope+atan_max)/atan_res;
                        theta = atan_LUT[LUT_ind];
                        theta_lower_limit = theta-6;
                        theta_upper_limit = theta+6;
                    }
                }
            }
        }
    }
}

```

```

// x and y values from the indices
// Origin (0,0) is the lower left corner of the image
x = j;
y = ymax-i;

for (theta=theta_lower_limit; theta<=theta_upper_limit;
theta+=theta_res)

//for (theta=theta_min; theta<=theta_max;
theta+=theta_res)
{
// compute theta index and convert theta to radians.
//theta_ind = (theta+theta_max)/theta_res; // comment out if
theta_res = 1
theta_ind = theta + theta_max; //
comment out if theta_res != 1

// compute the corresponding rho value from the line equation.
LUT_ind = theta+90;
rho = (x*cos_LUT[LUT_ind] + y*sin_LUT[LUT_ind])/32767;
rho_ind = (rho+rho_max)/rho_res; // comment out if rho_res = 1
//rho_ind = rho + rho_max; // comment
out if rho_res != 1

// update the accumulator array
houghArrayInd = rho_ind * n_theta + theta_ind;
ptrHT_Array[houghArrayInd]++;
//ptrHT_Array[houghArrayInd-1]++;
//ptrHT_Array[houghArrayInd+1]++;
ptrHT_Array[houghArrayInd-n_theta]++;
ptrHT_Array[houghArrayInd+n_theta]++;
}

}

ptrIn++;
}
ptrIn += col_min;
}
}

```

```
/******
```

Function :Hough_Transform

Input Argument :(*ptrIn) pointer to the input gray scaled image

Return Arguments :(*ptrHT_Array) pointer to the Hough Accumulator Array

Purpose :Computes the HT of a gray scaled image returning the accumulator array

```
*****/
```

```
void Hough_Transform_Non_Optimized(unsigned char *ptrIn, unsigned volatile short *ptrHT_Array)
{
```

```
    unsigned short int i;    //row index
    unsigned short int j;    //column index
```

```
    unsigned short houghArrayInd;
```

```
    unsigned short rows = N_Rows;    // # of rows
    unsigned short cols = N_Cols;    // # of columns
```

```
    // define the region to be processed
    unsigned short row_min = rows/2;
    unsigned short row_max = rows-1;
    unsigned short col_min = 10;
    unsigned short col_max = cols-col_min;
```

```
    signed char Gx, Gy;    // gradient in x and y directions
    float slope;    // Gy/Gx
    signed short int_slope;
```

```
    // Hough parameters defined in system.h
    signed short rho_min = RHO_MIN;
    signed short rho_max = RHO_MAX;
    unsigned char rho_res = RHO_RES;
    signed char theta_min = THETA_MIN;
    signed char theta_max = THETA_MAX;
    unsigned char theta_res = THETA_RES;
    unsigned short n_rho = N_RHO;
    unsigned short n_theta = N_THETA;
```

```
    // rho and theta indices and values
    unsigned short rho_ind, theta_ind;
    signed short rho;
    signed char theta;
    signed char theta_lower_limit, theta_upper_limit;
```

```
    // parameters for trig functions look-up tables
    unsigned short LUT_ind;
    signed short atan_max = ATAN_MAX;
    signed short atan_min = ATAN_MIN;
    float atan_res = ATAN_RES;
```

```
    // x and y values in the line equation
    unsigned short x, y;
```

```
    // max y value
    short ymax = rows-1;
```

```

// Threshold value to be used in sobel filtered image to find edges
// EDGE_THRESH is defined in system.h
unsigned char edge_thr = EDGE_THRESH;

// Edge pixels having a gray level below this threshold will be considered only
// GRAY_THRESH is defined in system.h
unsigned char gray_thr = GRAY_THRESH;

    // set the pointer to the start of the upper half of the image
    // first line is not processed
    ptrIn += row_min*cols;

    for (i=row_min;i<row_max;i++)
    {
        ptrIn += col_min;

        for(j=col_min;j<col_max;j++)
        {
            if (*ptrIn<=gray_thr)                // for each pixel below gray threshold:
            {

                Gy = (signed char) *(ptrIn-1) - *(ptrIn+1);                //
vertical filter [1 0 -1]
                //Gy = (signed char) *(ptrIn) - *(ptrIn-1);
                //Gy = ((signed char) *(ptrIn-1-cols)-*(ptrIn+1-cols)) + ((signed
char) *(ptrIn-1)-*(ptrIn+1)) + ((signed char) *(ptrIn+1+cols)-*(ptrIn+1+cols));
                Gx = (signed char) *(ptrIn-cols) - *(ptrIn+cols);        // horizontal
filter [1 ; 0 ; -1]
                //Gx = (signed char) *(ptrIn) - *(ptrIn-cols);
                //Gx = ((signed char) *(ptrIn-1-cols)-*(ptrIn-1+cols)) + ((signed
char) *(ptrIn-cols)-*(ptrIn+cols)) + ((signed char) *(ptrIn+1-cols)-*(ptrIn+1+cols));

                if (abs(Gy)>=edge_thr)                // if the current pixel is an edge
pixel:
                {

                                                                    //don't determine the theta value limit from
gradient(before optimization)
                if (Gx==0)
                {
                    theta_lower_limit = -6;
                    theta_upper_limit = 6;
                }
                else
                {
                    slope = ((float) Gy)/Gx;
                    int_slope = slope;
                    if (int_slope>=atan_max || int_slope<=atan_min)
                    {
                        theta_lower_limit = -6;
                        theta_upper_limit = 6;
                    }
                    else
                    {
                        LUT_ind = (slope+atan_max)/atan_res;
                        theta = atan_LUT[LUT_ind];
                        theta_lower_limit = theta-6;
                        theta_upper_limit = theta+6;
                    }
                }
            }
        }
    }

```

```

    }
}

// x and y values from the indices
// Origin (0,0) is the lower left corner of the image
x = j;
y = ymax-i;

for (theta=theta_lower_limit; theta<=theta_upper_limit;
theta+=theta_res)
    //for (theta=theta_min; theta<=theta_max;
theta+=theta_res) //for all theta values
    {
        // compute theta index and convert theta to radians.
        //theta_ind = (theta+theta_max)/theta_res; // comment out if
theta_res = 1
        theta_ind = theta + theta_max; //
comment out if theta_res != 1

        // compute the corresponding rho value from the line equation.
        // LUT_ind = theta+90;
        // rho = (x*cos_LUT[LUT_ind] + y*sin_LUT[LUT_ind])/32767; //comment
out if don't use LUT
        rho=(x*cos(theta* 0.0175) + y*sin(theta*0.0175));
        rho_ind = (rho+rho_max)/rho_res; // comment out if rho_res = 1
        //rho_ind = rho + rho_max; // comment
out if rho_res != 1

        // update the accumulator array
        houghArrayInd = rho_ind * n_theta + theta_ind;
        ptrHT_Array[houghArrayInd]++;
        //ptrHT_Array[houghArrayInd-1]++;
        //ptrHT_Array[houghArrayInd+1]++;
        ptrHT_Array[houghArrayInd-n_theta]++;
        ptrHT_Array[houghArrayInd+n_theta]++;
    }
}
ptrIn++;
}
ptrIn += col_min;
}
}

```

```
/******
```

Function :Hough_Peaks

Input Argument :(*ptrHT_Array) pointer to the Hough Accumulator Array

Return Arguments :None

Purpose :Locates the highest N_PEAKS_MAX peaks (local maxima)
in the Hough space. Rho, theta and HT values for the
peaks are written in global arrays theta_peaks,
rho_peaks and peak_scores

theta_peaks : theta values for the peaks
rho_peaks : rho values for the peaks
peak_scores : HT values for the peaks

```
*****/
```

```
void Hough_Peaks(unsigned volatile short *ptrHT_Array)
{
```

```
    // rho and theta indices
    unsigned short rho_ind, theta_ind;
```

```
    // rho and theta values
    signed short rho;
    signed char theta;
```

```
    // Hough parameters defined in system.h
    signed short rho_min = RHO_MIN;
    signed short rho_max = RHO_MAX;
    unsigned char rho_res = RHO_RES;
    signed char theta_min = THETA_MIN;
    signed char theta_max = THETA_MAX;
    unsigned char theta_res = THETA_RES;
    unsigned short n_rho = N_RHO;
    unsigned short n_theta = N_THETA;
```

```
    // HT scores below the threshold will not be considered as peaks
    unsigned short hough_thr=20;
```

```
    // peaks must be the local maximum around the specified neighborhood
    unsigned short nhoodsize_theta=5/THETA_RES; // 5 degrees
    unsigned short nhoodsize_rho=10/RHO_RES; // 10 pixels
```

```
    // region to be processed
    unsigned short rho_ind_min = nhoodsize_rho;
    unsigned short rho_ind_max = n_rho-nhoodsize_rho;
    unsigned short theta_ind_min = nhoodsize_theta;
    unsigned short theta_ind_max = n_theta-nhoodsize_theta;
```

```
    // Hough array index
    unsigned short houghArrayInd = rho_ind_min*n_theta;
```

```
    unsigned short current_score;
    unsigned char bln_peak=0;
    unsigned short i, j, imax, jmax;
    unsigned short tmp_ind;
    unsigned short tmp_score;
```

```
    // max # of peaks (local maxima) that will be located
    unsigned short n_peaks_max=N_PEAKS_MAX;
```

```

// initialization of global arrays
for (i=0;i<n_peaks_max;i++) {
    peak_scores[i] = 0;
    theta_peaks[i] = 0;
    rho_peaks[i] = 0; }
N_peaks = 0;

for (rho_ind=rho_ind_min; rho_ind<rho_ind_max; rho_ind++)           // iterate through all
points
{
    houghArrayInd += nhoothesize_theta;

    for (theta_ind=theta_ind_min; theta_ind<theta_ind_max; theta_ind++)
    {

        //houghArrayInd = rho_ind*n_theta+theta_ind;

        current_score = ptrHT_Array[houghArrayInd];                // current HT score

        if (current_score>hough_thr) {                             // if the current score is above the
threshold

            // check the neighborhood whether the point is a local maximum or not
            // set bln_peak to 1 if so
            bln_peak = 1;
            imax = rho_ind+nhoothesize_rho;
            jmax = theta_ind+nhoothesize_theta;
            for (i=rho_ind-nhoothesize_rho; i<=imax; i++)
            {
                for (j=theta_ind-nhoothesize_theta; j<=jmax; j++)
                {
                    tmp_ind = i*n_theta+j;
                    tmp_score = ptrHT_Array[tmp_ind];
                    if (current_score<tmp_score)
                    {
                        bln_peak = 0;
                        break;
                    }
                }
            }
            if (bln_peak==0)
                break;
        }

        // If the point is a local maximum:
        if (bln_peak==1)
        {
            // compute theta
            //theta = theta_min + theta_ind*theta_res;                // comment
out if theta_res = 1
            theta = theta_min + theta_ind;
            // comment out if theta_res != 1

            // compute rho
            rho = rho_min + rho_ind*rho_res;                        // comment
out if rho_res = 1

```

```

//rho = rho_min + rho_ind;
// comment out if rho_res != 1

// compare the peak with the ones in the output array
for (i=0;i<n_peaks_max;i++)
{
    // note that peaks are sorted from highest to
    // if the peak is greater than the i-th peak,
    // starting from the i-th peak slide each peak
    // to the right in the array throwing out the
    // lowest peak (last peak in the array).
    // Finally, add the new peak as the i-th peak
    // to the array
    if (current_score>peak_scores[i])
    {
        // slide elements to the right
        for (j=n_peaks_max-1;j>i;j--)
        {
            peak_scores[j] = peak_scores[j-
1];
            theta_peaks[j] = theta_peaks[j-
1];
            rho_peaks[j] = rho_peaks[j-1];
        }

        // add the new peak as the i-th peak
        peak_scores[i] = current_score;
        theta_peaks[i] = theta;
        rho_peaks[i] = rho;

        // set the neighborhood around the peak
        for (i=rho_ind-nhoodsize_rho; i<=imax;
i++)
        {
            for (j=theta_ind-nhoodsize_theta;
j<=jmax; j++)
            {
                tmp_ind = i*n_theta+j;
                ptrHT_Array[tmp_ind] = 0;
            }
        }

        N_peaks++; // update peak count
        break; // continue with the next point
    }
}

// update Hough array index so it points to next point in the Hough space

```

```

        houghArrayInd++;
    }

    houghArrayInd += nhoudsize_theta;    // update Hough array so it points to next line
}

if (N_peaks > n_peaks_max)
    N_peaks = n_peaks_max;
}

```

```
/******
```

```
Function      :Find_Lane_Marks
```

```
Input Arguments :None
```

```
Return Arguments :None
```

```
Purpose        :Among the peaks located in the Hough space, finds the  
               ones that correspond to the lane marks.
```

```
*****/
```

```
void Find_Lane_Marks(void)  
{
```

```
    unsigned short rows = N_Rows;          // # of rows  
    unsigned short cols = N_Cols;          // # of columns
```

```
    // # of peaks detected in the Hough space by Hough_Peaks function  
    // N_peaks is a global variable  
    unsigned short n_peaks = N_peaks;  
    unsigned short i, j, LUT_ind;
```

```
    // rho and theta values  
    signed short rho;  
    signed char theta1, theta2;
```

```
    unsigned short candidate_pair_score=0;  
    unsigned short current_score;  
    unsigned short lane_mark1_index, lane_mark2_index;
```

```
    signed short bottom_intercept_1, bottom_intercept_2;  
    signed short top_intercept_1, top_intercept_2;  
    unsigned short lane_width_bottom_image, lane_width_top_image;  
    unsigned short lane_width_bottom_world, lane_width_top_world;  
    float ratio;  
    signed short left_lane, right_lane;  
    signed int road_angle;
```

```
    for (i=0; i<n_peaks; i++)  
    {
```

```
        rho = rho_peaks[i];  
        theta1 = theta_peaks[i];  
        LUT_ind = theta1+90;  
        bottom_intercept_1 = (rho*32767)/cos_LUT[LUT_ind];  
        top_intercept_1 = bottom_intercept_1-(rows*sin_LUT[LUT_ind])/cos_LUT[LUT_ind];
```

```
    for (j=i+1; j<n_peaks; j++)  
    {
```

```
        rho = rho_peaks[j];  
        theta2 = theta_peaks[j];  
        LUT_ind = theta2+90;  
        bottom_intercept_2 = (rho*32767)/cos_LUT[LUT_ind];  
        top_intercept_2 = bottom_intercept_2-(rows*sin_LUT[LUT_ind])/cos_LUT[LUT_ind];  
        lane_width_bottom_image = abs(bottom_intercept_1-bottom_intercept_2);  
        lane_width_top_image = abs(top_intercept_1-top_intercept_2);
```

```
    // left and right lane intercepts at the bottom of the image  
    if (bottom_intercept_1 > bottom_intercept_2){  
        left_lane = bottom_intercept_2;
```

```

        right_lane = bottom_intercept_1;
    }
    else {
        left_lane = bottom_intercept_1;
        right_lane = bottom_intercept_2;
    }

    // lane (road) angle for the candidate pair
    road_angle = theta1/2 + theta2/2;

    // lane widths at the bottom and the top of the image
    LUT_ind = road_angle+90;
    lane_width_bottom_world =
lane_width_bottom_image*cos_LUT[LUT_ind]/32767;
    lane_width_bottom_world =
lane_width_bottom_world*(Xmax+Xmin)/(cols-1);
    lane_width_top_world =
lane_width_top_image*cos_LUT[LUT_ind]/32767;
    lane_width_top_world = lane_width_top_world*(Xmax-Xmin)/(cols-1);

    if (lane_width_bottom_image>lane_width_top_image && lane_width_bottom_world >
0.8*lane_width_world && lane_width_bottom_world < 1.2*lane_width_world && (float)
lane_width_top_world/lane_width_bottom_world > 0.8 &&
lane_width_top_world/lane_width_bottom_world < 1.2)

    {
        current_score = peak_scores[i]+peak_scores[j];
        if (current_score>candidate_pair_score)
        {
            candidate_pair_score = current_score;
            lane_mark1_index = i;
            lane_mark2_index = j;
        }
    }

}

}

if (candidate_pair_score==0) // two lane marks not found
{

    N_peaks = 0;

    for (i=0; i<n_peaks; i++)
    {

        rho = rho_peaks[i];
        theta1 = theta_peaks[i];
        LUT_ind = theta1+90;
        bottom_intercept_1 = (rho*32767)/cos_LUT[LUT_ind];
        top_intercept_1 = bottom_intercept_1-(rows*sin_LUT[LUT_ind])/cos_LUT[LUT_ind];

        if (bottom_intercept_1<cols/3 || bottom_intercept_1>2*cols/3)
        {
            current_score = peak_scores[i];
            if (current_score>candidate_pair_score)

```

```

        {
            candidate_pair_score = current_score;
            lane_mark1_index = i;
            N_peaks = 1;
            N_frames_lanes_detected++;
        }
    }

    if (N_peaks==1) {
        rho_peaks[0] = rho_peaks[lane_mark1_index];
        theta_peaks[0] = theta_peaks[lane_mark1_index];
    }

}
else
{
    N_peaks = 2;
    rho_peaks[0] = rho_peaks[lane_mark1_index];
    theta_peaks[0] = theta_peaks[lane_mark1_index];
    rho_peaks[1] = rho_peaks[lane_mark2_index];
    theta_peaks[1] = theta_peaks[lane_mark2_index];
    N_frames_lanes_detected++;
}
}

```

```

/*****

```

Function :Draw_Hough_Lines

Input Arguments :(*ptrOut) pointer to the output buffer

(*theta) pointer to the theta values for the lines

(*rho) pointer to the rho values for the lines

(nlines) # of lines to be drawn

(color) color of the line (0:yellow 1:red 2: blue)

Return Arguments :None

Purpose :Draw Hough lines over the active field 1 of NTSC/PAL

frame. Theta and rho parameters for the lines are given in theta and rho arrays.

```

*****/

```

```

void Draw_Hough_Lines(volatile short *ptrOut, signed volatile char *theta, signed volatile short
*rho, unsigned volatile short nlines, unsigned char color)
{

```

```

    // Define line color

```

```

    // Y, Cb and Cr values for yellow

```

```

    unsigned char Y = 225;

```

```

    unsigned char Cb = 0;

```

```

    unsigned char Cr = 150;

```

```

    // set line color to red if color=1

```

```

    if (color==1) {

```

```

        Y = 225;

```

```

        Cb = 0;

```

```

        Cr = 250; }

```

```

    else if (color==2) {

```

```

        Y = 225;

```

```

        Cb = 250;

```

```

        Cr = 0;

```

```

    }

```

```

    int lineIndex;    // line index

```

```

    int i;            // row index

```

```

    int j;            // column index

```

```

    unsigned short LUT_ind;    // LUT index

```

```

    int imgIndex;

```

```

    unsigned short rows = N_Rows;    // # of rows

```

```

    unsigned short cols = N_Cols;    // # of columns

```

```

    // x and y values for the lines

```

```

    short x, y;

```

```

    // max x and y values

```

```

    short ymax = rows-1;

```

```

    short xmax = cols-1;

```

```

    // theta and rho values for the current line

```

```

    float current_theta;

```

```

    signed short current_rho;

```

```

    // pointer to the output buffer

```

```

    // since the pixel values will be written byte-by-byte,

```

```

// we need to define a pointer with unsigned char type.
unsigned char *ptrOutBuffer;

// # line width (in bytes) of a NTSC/PAL frame
unsigned short line_width = ADI_ITU656_LINE_WIDTH;

// skip vertical blanking
ptrOutBuffer = ((unsigned char *) ptrOut) + V_BLANK;

for (lineIndex=0; lineIndex<nlines; lineIndex++) // for each line:
{
    // theta and rho values for the line
    current_theta = (*theta); // theta in degrees
    current_rho = *rho; // rho
    LUT_ind = current_theta+90;

    if (abs(current_theta)>=45)
    {
        for (x=0; x<cols; x++) // for all x:
        {
            // compute y from the line equation
            y = (current_rho -
(x*cos_LUT[LUT_ind])/32767)*32767/sin_LUT[LUT_ind];

            // if the point is outside the image coordinates,
            // continue with the next one.
            if (y<0 || y>ymax) {
                continue;
            }

            // image coordinate of the point corresponding to x and y values
            // It is assumed that the pixels per line in the processed image
            // is half the pixels per line in the original frame. That is why
            // the equation has the term 4*x instead of 2*x.
            imgIndex = (ymax-y)*line_width+H_BLANK+4*x+1;

            // set color of the point (pixel) and its neighbor
            ptrOutBuffer[imgIndex-1] = Cb;
            ptrOutBuffer[imgIndex] = Y;
            ptrOutBuffer[imgIndex+1] = Cr;
            ptrOutBuffer[imgIndex+2] = Y;

        }
    }
    else
    {
        for (y=0; y<rows; y++) // for all y:
        {
            // compute x from the line equation
            x = (current_rho -
(y*sin_LUT[LUT_ind])/32767)*32767/cos_LUT[LUT_ind];

            // if the point is outside the image coordinates,
            // continue with the next one.
            if (x<0 || x>xmax) {
                continue;
            }

```

```

// image coordinate of the point corresponding to x and y values
imgIndex = (ymax-y)*line_width+H_BLANK+4*x+1;

// set color of the point (pixel) and its neighbor
ptrOutBuffer[imgIndex-1] = Cb;
ptrOutBuffer[imgIndex] = Y;
ptrOutBuffer[imgIndex+1] = Cr;
ptrOutBuffer[imgIndex+2] = Y;

    }
}

theta++;
rho++;

}

}

```

```

/*****

```

Function :copy_ProcessedImage2Buffer

Input Argument :(*ptrIn) pointer to the input (processed) image

(*ptrOut) pointer to the output buffer (NTSC/PAL) frame

Return Argument :None.

Purpose :Copies the processed image to the output buffer to be displayed (PAL/NTSC)

It is assumed that the processed image is active field 1 only and
the # of pixels per line is reduced by 2 compared to the original.

```

*****/

```

```

void copy_ProcessedImage2Buffer(unsigned char *ptrIn, volatile short *ptrOut)
{

```

```

    unsigned char *ptrBuffer;

```

```

    unsigned int i, j;

```

```

    int rows=N_Rows;

```

```

    int cols=N_Cols;

```

```

    // Skip vertical blanking.

```

```

    ptrBuffer=((unsigned char *) ptrOut) + V_BLANK;

```

```

    // Copy the gray values as luminosity (Y) values in the output buffer

```

```

    for (i=0; i<rows; i++) {

```

```

        ptrBuffer += H_BLANK; // skip horizontal blanking

```

```

        for (j=0; j<cols; j++) {

```

```

            *(ptrBuffer++) = 128; // chroma (Cb,Cr) values are 128 (no

```

```

color-just gray image)

```

```

            *(ptrBuffer++) = *ptrIn;

```

```

            *(ptrBuffer++) = 128;

```

```

            *(ptrBuffer++) = *(ptrIn++);

```

```

        }

```

```

    }

```

```

}

```

/******

Function: Deinterlace Frame

Description: This functions copies field 1 to the field 2.

This is necessary to reduce the flicker caused
when the odd and even fields contain different frames

*****/

```
void deinterlaceFrame(volatile short *ptrIn) {  
    // Deinterlace frame by copying field1 to field2.  
    // 4 bytes is transferred at one time  
  
    unsigned long int *ipF, *opF;           // pointer to start of active fields  
    unsigned int i, j;  
    unsigned k, m;  
    unsigned int h_blank;  
    unsigned int v_blank;  
  
    // Horizontal blankink in 4-bytes.  
    h_blank = H_BLANK/4;  
  
    // Vertical blankink in 4-bytes.  
    v_blank = V_BLANK/4;  
  
    // initialize pointers  
    // ipF will point to active field 1  
    // opF will point to active field 2  
    ipF = ((u32 *) ptrIn) + v_blank;  
    opF = ((u32 *) ptrIn) + (ADI_ITU656_ILF2_START)*(ADI_ITU656_LINE_WIDTH/4);  
  
    // Copy field1 to field2  
    for (i=0; i<ADI_ITU656_ACTIVE_FLINES; i++) {  
        ipF+=h_blank;  
        opF+=h_blank;  
        for (j=0; j<ADI_ITU656_WIDTH/2; ipF+=1, opF+=1, j++)  
            *opF = *ipF;  
    }  
}
```

```

/*****

Function      :Imag_Thr
Input Argument :(*ptr) pointer to the input gray scaled image.
                (thr) threshold value
Return Argument :The thresholded image
Purpose       :Applies a threshold to a gray scaled image.

*****/
void Imag_Thr(unsigned char *ptr, unsigned char thr)
{
    int i;    //row index
    int j;    //column index
    int rows = N_Rows;    // # of rows
    int cols = N_Cols;    // # of columns

    for (i=0;i<rows*cols;i++)
    {
        if (*ptr>=thr)
            *(ptr++) = 255;
        else
            *(ptr++) = 0;
    }
}

```

```

/*****

```

```

Function      :Navigate_Robot
Input Arguments :None
Return Arguments :None
Purpose       :Navigation algorithm.

```

```

*****/

```

```

void Navigate_Robot(void)
{

```

```

    int duty_cycle_right_motors, duty_cycle_left_motors;

```

```

    if (N_peaks>0)                // at least one lane is detected
    {

```

```

        no_lane_detected=false;

```

```

        if (robot_heading_angle>0)                // positive angle corresponds to a left curve

```

```

                                                    // increase the speed of

```

```

the right motors

```

```

                                                    // decrease the speed

```

```

of the left motors

```

```

    {

```

```

        duty_cycle_left_motors = base_duty_cycle_LF_motor - 3*robot_heading_angle;

```

```

        duty_cycle_right_motors = base_duty_cycle_RF_motor + 3*robot_heading_angle;

```

```

    }

```

```

    else

```

```

                                                    // negative angle corresponds to

```

```

a right curve

```

```

                                                    // increase the speed of

```

```

the left motors

```

```

                                                    // decrease the speed

```

```

of the right motors

```

```

    {

```

```

        duty_cycle_left_motors = base_duty_cycle_LF_motor - 3*robot_heading_angle;

```

```

        duty_cycle_right_motors = base_duty_cycle_RF_motor + 3*robot_heading_angle;

```

```

    }

```

```

}

```

```

else                                // no lanes detected

```

```

    // if no lanes detected first turn the left still ;no lane turn the right

```

```

{

```

```

    no_lane_detected=true;

```

```

    duty_cycle_left_motors = base_duty_cycle_RF_motor;

```

```

    duty_cycle_right_motors = base_duty_cycle_RF_motor;

```

```

}

```

```

if (N_peaks==1)                // one lane detected

```

```

{

```

```

    if (left_lane_detected==true && theta_peaks[0]<0)                // left lane detected only

```

```

                                                    // zero radius turn to the right

```

```

    {

```

```

        duty_cycle_left_motors = base_duty_cycle_LF_motor;

```

```

        duty_cycle_right_motors = 1;

```

```

    }

```

```

        else if (theta_peaks[0]>0) //
right lane detected only

        // zero radius turn to the left
        {
            duty_cycle_left_motors = 1;
            duty_cycle_right_motors = base_duty_cycle_RF_motor;
        }
    }

    // duty cycle limits
    if (duty_cycle_left_motors < min_duty_cycle_motors)
        duty_cycle_left_motors = min_duty_cycle_motors;
    if (duty_cycle_left_motors > max_duty_cycle_motors)
        duty_cycle_left_motors = max_duty_cycle_motors;
    if (duty_cycle_right_motors < min_duty_cycle_motors)
        duty_cycle_right_motors = min_duty_cycle_motors;
    if (duty_cycle_right_motors > max_duty_cycle_motors)
        duty_cycle_right_motors = max_duty_cycle_motors;

    // Update the duty cycles
    if (duty_cycle_LF_motor < duty_cycle_left_motors)
    {
        duty_cycle_LF_motor+=5;
        duty_cycle_LR_motor+=5;
    }
    else if (duty_cycle_LF_motor > duty_cycle_left_motors)
    {
        duty_cycle_LF_motor-=5;
        duty_cycle_LR_motor-=5;
    }
    if (duty_cycle_RF_motor < duty_cycle_right_motors)
    {
        duty_cycle_RF_motor+=5;
        duty_cycle_RR_motor+=5;
    }
    else if (duty_cycle_RF_motor > duty_cycle_right_motors)
    {
        duty_cycle_RF_motor-=5;
        duty_cycle_RR_motor-=5;
    }

    *pTIMER0_WIDTH = *pTIMER0_PERIOD*(duty_cycle_LF_motor)/100.;
    *pTIMER1_WIDTH = *pTIMER1_PERIOD*(duty_cycle_RF_motor)/100.;
    *pTIMER5_WIDTH = *pTIMER5_PERIOD*(duty_cycle_LR_motor)/100.;
    *pTIMER6_WIDTH = *pTIMER6_PERIOD*(duty_cycle_RR_motor)/100.;
}

```

```

/*****

```

```

Function      :Perspective_Correction

```

```

Input Arguments :None

```

```

Return Arguments :None

```

```

Purpose       :Perspective correction for the lane(s) detected

```

```

The function also sets the heading angle for the robot

```

```

*****/

```

```

void Perspective_Correction(void)

```

```

{

```

```

    unsigned short rows = N_Rows;          // # of rows

```

```

    unsigned short cols = N_Cols;          // # of columns

```

```

    float u, v;                            // image coordinates

```

```

    float x1, x2, y; // world coordinates

```

```

    float w;                               // normalization factor

```

```

    // lane angle

```

```

    signed char lane_angle;

```

```

    unsigned short LUT_ind;

```

```

    // bottom and the top intercepts of the lanes

```

```

    signed short bottom_intercept_image, top_intercept_image;

```

```

    signed short bottom_intercept_world, top_intercept_world;

```

```

    // x array will hold the x coordinates of the points on the lanes at y=y_dest_world

```

```

    // x_dest_world is the x coordinate of the middle of the lane at y=y_dest_world

```

```

    float x[2], x_dest_world;

```

```

    int i; // dummy counter

```

```

    for (i=0;i<N_peaks;i++)                // for each lane detected:

```

```

    {

```

```

        // find the bottom and the top intercept of the lane in the image plane

```

```

        LUT_ind = theta_peaks[i]+90;

```

```

        bottom_intercept_image = (rho_peaks[i]*32767)/cos_LUT[LUT_ind];

```

```

        top_intercept_image=bottom_intercept_image*(rows*sin_LUT[LUT_ind])/cos_LUT[LUT_
ind];

```

```

        // convert the bottom intercept point to world coordinates

```

```

        u = bottom_intercept_image;          // v = 0

```

```

        //x1 = H[0][0]*u+H[0][2];           // y = 0   w = 1

```

```

        x1 = u*(Xmax+Xmin)/(cols-1);

```

```

        // convert the top intercept point to world coordinates

```

```

        u = top_intercept_image;

```

```

        //v = rows-1;

```

```

        //x2 = H[0][0]*u+H[0][1]*v+H[0][2];    // y = Ymax

```

```

        //w = H[2][0]*u+H[2][1]*v+H[2][2];

```

```

        //x2 = x2 / w;

```

```

        x2 = Xmin+u*(Xmax-Xmin)/(cols-1);

```

```

        // compute lane angle in the world coordinates

```

```

        lane_angle = -atanf((x2-x1)/Ymax)*57.2958;           // 180/pi = 57.2958

```

```

        // perspective correction of the lane in the image

```

```

        theta_peaks[i] = lane_angle;

```

```

    LUT_ind = theta_peaks[i]+90;
    rho_peaks[i] = bottom_intercept_image * cos_LUT[LUT_ind]/32767;

    // x coordinate of the point on the lane at y=y_dest_world
    x[i] = x1+(x2-x1)*y_dest_world/Ymax;
}

// compute x_dest_world (see definition above)
if (N_peaks==2) // two lanes detected
{
    x_dest_world = (x[0]+x[1])/2;
}

else if (N_peaks==1) // one lane detected...
{
    left_lane_detected = false;

    // Determine which lane it is
    LUT_ind = theta_peaks[0]+90;
    bottom_intercept_world = (rho_peaks[0]*32767)/cos_LUT[LUT_ind];
    if (bottom_intercept_world < min_field_of_view_width/2) // left lane
    {
        x_dest_world = x[0]+(lane_width_world*32767)/cos_LUT[LUT_ind]/2;
        left_lane_detected = true;
    }
    else // right lane
    {
        x_dest_world = x[0]-(lane_width_world*32767)/cos_LUT[LUT_ind]/2;
    }
}

if (N_peaks>0)
{
    if (x_dest_world==(Xmax+Xmin)/2)
        robot_heading_angle = 0;
    else
        robot_heading_angle = -atanf((x_dest_world-(Xmax+Xmin)/2)/
        y_dest_world)*57.2958;
}
else
{
    robot_heading_angle = 0;
}

// write the heading info to theta_peaks and rho_peaks so a line will be drawn over the image
theta_peaks[2] = robot_heading_angle;
LUT_ind = theta_peaks[2]+90;
rho_peaks[2] = (cols/2) * cos_LUT[LUT_ind]/32767;
}

```

ÖZGEÇMİŞ

Gürkan KÜÇÜKYILDIZ, Eylül 1986, Trabzon doğumludur. İlk, orta ve lise öğrenimini Uşak’ da tamamlamıştır. 2004 yılında kazandığı Kocaeli Üniversitesi Mühendislik Fakültesi Elektronik ve Haberleşme Mühendisliği Bölümü’ nden 2009 yılında Elektronik ve Haberleşme Mühendisi olarak mezun oldu. 2009 yılında, Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, Mekatronik Mühendisliği Anabilim Dalı’ nda Yüksek Lisans eğitime başlamıştır. 2011 yılında, Kocaeli Üniversitesi Mühendislik Fakültesi, Mekatronik Mühendisliği Bölümü’ nde Araştırma Görevlisi olarak çalışmaya başlamıştır ve halen görevine devam etmektedir. Gömülü sistem tasarımı, elektronik devre tasarımı, gömülü yazılım uygulamaları, PIC ve DSP konularında uzmandır.