KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

MOBİL ROBOTTA SES VE GÖRÜNTÜ İŞLEME

YÜKSEK LİSANS

Bertan DAĞISTANLI

Anabilim Dalı: Mekatronik Mühendisliği
Danışman: Yrd. Doç. Dr. Hasan OCAK

KOCAELİ, 2008

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

MOBİL ROBOTTA SES VE GÖRÜNTÜ İŞLEME

YÜKSEK LİSANS TEZİ

Bertan DAĞISTANLI

Tezin Enstitüye Verildiği Tarih: 26 Mayıs 2008

Tezin Savunulduğu Tarih: 25 Haziran 2008

Tez Danışmanı	Üye	Üye
Yrd. Doç. Dr.	Yrd. Doç. Dr.	Yrd. Doç. Dr.
Hasan Ocak	Metin Ertunç	Şeref Naci Engin
(.....)	(.....)	(.....)

KOCAELİ, 2008

ÖNSÖZ ve TEŞEKKÜR

Teknolojinin hızla geliştiği dünyamızda otomasyon ve robot teknolojileri hayatın hemen hemen her alanında kullanılmaya başlamıştır. Robot uygulamaları, uzay çalışmalarından, savunma sanayine kadar birçok alanda uygulamalarla karşımıza çıkmaktadır ve kullanım alanları her geçen gün büyük bir hızla artmaktadır.

Bu tez çalışmasında ses ve görüntü algılayan bir robot projesi üzerinde çalışılmıştır. Robot tarafından sesin yönü tespit edilip robot sese yönlendirilmiş ve imge işleme uygulamasıylada buna destek verilmiştir.

Bu alanda tez çalışması yapmama ve projemin uygulanması aşamasında yönlendirmeleri ile bana sürekli destek olan değerli hocam Yrd.Doç.Dr. Hasan OCAK'a, bana yaşamım boyu katkıda bulunan aileme ve tüm hocalarıma teşekkür ederim.

İÇİNDEKİLER

ÖNSÖZ i
İÇİNDEKİLER ii
ŞEKİLLER DİZİNİ iv
TABLOLAR DİZİNİ v
SİMGELER vi
ÖZET vii
İNGİLİZCE ÖZET iix
1. GİRİŞ 1
1.1. Robot Algısı 5
1.2. Tezin İçeriği 6
2. TEORİK BİLGİLER 8
2.2. Mikrofon Türleri 8
2.2.1. Kapasitif mikrofonlar 8
2.2.2. Dinamik mikrofonlar 9
2.2.3. Piezoelektrik kristalli mikrofonlar 9
2.2.4. Elektret mikrofonlar 10
2.2.5. Şeritli mikrofonlar 11
2.2.6. Karbon tozlu mikrofonlar 11
2.3. Çoklu Mikrofon Uygulamaları 12
2.3.1. Ses kaynağının yerinin kestirimi 12
2.3.2. Kamera yönlendirilmesiyle konuşmacının izlenmesi 13
3.SES KAYNAĞININ YERİNİN KESTİRİLMESİ 14
3.1.Zaman Farkı ve Çapraz Korelasyon Kullanılarak Kaynağın Konumunun Saptanması 17
3.2.Ses Mimarisi 20
4.HAREKET KESTİRİMİ 21
4.1. Hareket Dengeleme 22
4.2. Hareket Algılamasını Güçleştiren Durumlar 22
4.2.1. Işık değişimleri 23
4.2.2. Gölge hareketleri 23
4.2.3. Örtülen ve örtülmeyen bölge 23
4.2.4. Gürültü 23
4.2.5. Açıklık Problemi 24
4.2.6. Piksel değerlerinin düzgün dağılıma sahip olması 24
4.3. Basit Fark Alma Yöntemi 24
4.4. İstatistiksel Basit Fark Alma Yöntemi 26
5.DONANIM 31
5.1. Kontrol Karsı 31
5.1.1. ADSP-BF561 Blackfin simetrik çoklu işlemci 31
5.1.2. ADSP BF561 EZ-KIT LITE 33
5.2. Elektrik Motorları 34
5.2.1. DC motor 36
5.3. Redüktör 37

5.4. Optik Kodlayıcılar....38
5.5. Motor Sürücü Devreleri....39
5.6. Mikrofon Ön Yükselteç Devresi....42
5.7. Sistem Şeması....43
6. SONUÇLAR VE ÖNERİLER....44
KAYNAKLAR....46
EKLER....47

ŞEKİLLER DİZİNİ

Şekil 1.1. Deprem kurtarma robotu Hibiscus 2
Şekil 1.2. NASA'nın Mars robotları Spirit, Opportunity 3
Şekil 1.3. ASELSAN tarafından gerçekleştirilen insansız araç projesi İZCİ 3
Şekil 1.4. Lynx – 5 Robot kolu (sabit robot) 4
Şekil 1.5. Mobil (hareketli) robot 4
Şekil 1.6. Sensör örnekleri 5
Şekil 2.1. Kapasitif mikrofonun yapısı 9
Şekil 2.2. Dinamik mikrofonun yapısı 9
Şekil 2.3. Piezoelektrik kristalli mikrofonun yapısı 10
Şekil 2.4. Elektret mikrofonun yapısı 10
Şekil 2.5. Şeritli mikrofonun yapısı 11
Şekil 2.6. Karbon tozlu mikrofonun yapısı 12
Şekil 3.1. Ses kaynağından mikrofonlara gelen sinyaller arasındaki gecikme 14
Şekil 3.2. Sağ mikrofondan alınan ses işaretinin vektör grafiği 15
Şekil 3.3. Sol mikrofondan alınan ses işaretinin vektör grafiği 15
Şekil 3.4. Korelasyon vektörünün grafiği 16
Şekil 3.5. Ses kaynağının mikrofonların ortasında olma durumu 17
Şekil 3.6. Ses kaynağının açı değerinin hesaplanması 18
Şekil 3.7. Sin(x) ve Cos(x) 'in geometrik diyagramı 19
Şekil 3.8. Ses Mimarisi 20
Şekil 5.1. Blackfin işlemci 31
Şekil 5.2. Blackfin işlemcinin iç yapısı 32
Şekil 5.3. ADSP BF561 EZ-KIT LITE 33
Şekil 5.4. Motor çeşitleri 35
Şekil 5.5. Robotta kullanılan motor 36
Şekil 5.6. DC motorun yapısı 37
Şekil 5.7. Redüktörlerin iç yapısı 37
Şekil 5.8. Optik kodlayıcının yapısı 38
Şekil 5.9. Optik kodlayıcı için tasarlanan sinüs – kare dalga dönüştürücü devresi 39
Şekil 5.10. L293D entegresinin bağlantı şeması 40
Şekil 5.11. Motor sürücü devre şeması 41
Şekil 5.12. Ön yükselteç devre şeması 42
Şekil 5.13. Ön yükselteç devresinin frekans karakteristiği 42
Şekil 5.14. Sistemin ön, arka ve alttan görünümleri 43
Şekil 5.15. Sistem şeması 44

TABLOLAR DİZİNİ

Tablo 5.1. Motoru farklı yönlere döndürmek için L293D nin data portuna gönderilen Sinyaller 40

SİMGELER

t	: zaman
V_{ses}	: ses hızı
T_h	: eşik değeri
D	: fark görüntüsü
A	: ortalama
S	: standart sapma
L	: benzerlik oranı
f_s	: örnekleme frekansı
mV	: milivolt
kHz	: kilohertz
ms	: milisaniye
KByte	: kilobayt

σ	: korelasyon sonucu bulunan gecikme katsayısı
Δ	: örnek değeri arasındaki zaman farkı
θ	: ses kaynağının mikrofonlara göre bulunduğu açı
μ	: ortalama değer
α	: adaptasyon katsayısı

MOBİL ROBOTTA SES VE GÖRÜNTÜ İŞLEME UYGULAMASI

Bertan DAĞISTANLI

Anahtar Kelimeler: Ses Kaynağının Yerinin Kestirimi, Mobil Robot, Ses İşleme, Görüntü İşleme.

Özet: Bu tez çalışmasında ses ve görüntü sinyallerini işleyerek insan gibi görme ve işitme algılarına sahip, dış dünya ile iletişim kuran bir mobil robot uygulaması gerçekleştirilmiştir. Proje, yazılım ve donanım kısımlarından oluşmaktadır. Tasarlanan robot, üzerinde bulunan mikrofonlar yardımıyla ses kaynağının yerini tespit edip kaynağa yönelmektedir. Proje donanımsal olarak, çift çekirdekli bir çokluortam kontrol kartı, redüktörlü ve optik kodlayıcılı dc motorlar, motor sürücü devreleri, optik algılayıcı sinyal dönüştürücü devreleri, mikrofonlar ve mikrofonların ön yükselteç devrelerinden oluşmaktadır. Kontrol kartının programlanmasında C programlama dili kullanılmıştır. Proje geliştirilerek insansız araç projesi yada kurtarma robotu olarak kullanılabilme özelliğine sahiptir.

SOUND AND IMAGE PROCESSING IN MOBILE ROBOT

Bertan DAĞISTANLI

Keywords: Sound Localization, Mobile Robot, Sound Processing, Image Processing

Abstract: In this study, a mobile robot was developed that can interact with environment like humans do through processing sound signals and images . The project consists of software and hardware components. Using the microphones installed, the designed mobile robot can localize a sound source and turns towards the source. The hardware component of the project consists of a dual core multi-media control card, dc motors with reductors and optical decoders, motor drive circuits, optical sensor signal conditioner circuits, microphones and microphone pre-amplifier circuits. C programming language was used to program the control card. With further improvements, the project can be used in unmanned car and rescue robot applications.

1. GİRİŞ

Teknolojinin son yıllardaki hızlı gelişimiyle, hayatımızın pek çok alanında robotlar kullanılmaya başlanmıştır. Robot kelimesi ilk olarak Karel Capek'in 1920 yılında yazmış olduğu R.U.R.("Rossum' un Evrensel Robotları") adlı oyununda geçmiştir. Bu oyunda robot, duyguları olmayan, otonom ve mekanik bir varlık olarak ifade edilmiştir. Sonraki yıllarda Isaac Asimov'un Robot serisi ile teknolojik bir robot olgusu oluşturmuş ve üç robot yasası ile robotların her zaman insanların amaçlarına hizmet edeceği, hiçbir şartta kendi amaçlarının insanların amaçlarından üstün olamayacağını belirtilmiştir [1,2].

Bu yasalar;

- Robot, hiçbir şekilde insanoğluna zarar veremez veya pasif kalarak zarar görmesine izin veremez.
- Robot, kendisine insanlar tarafından verilen komutlara, 1. kuralla çelişmediği sürece uymak zorundadır.
- Robot, birince ve ikinci kuralla çelişmediği sürece kendi varlığını korumak zorundadır.

olarak belirtilmiştir[1].

Robotlar dış dünya ile etkileşimde bulunabilirler. Algılayabilmek için robotlarda genellikle sensörler ve/veya ses – video sistemleri kullanılır. Bu aygıtlardan alınan veriler, elektronik bir beyinde işlenip, mekanik hareket sistemi kullanılarak tanımlanmış işlemler yerine getirilir. Bununla birlikte, yine mekanik bir sisteme robot denilebilmesi için;

- İşlem yapma becerisi,
- İşlemin sonucunu tahmin etme becerisi,
- Karar verme becerisi

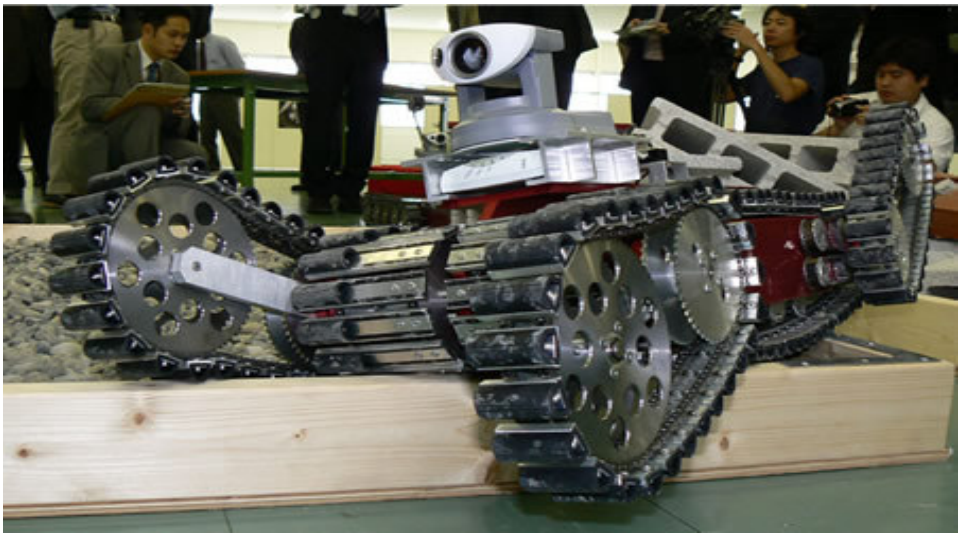
olması gereklidir[2].

Robot kavramı, zamanla insan hayatındaki konfor ve güvenlikle ilgili alanlarda değişime yol açan bir kavramdır. Bu alanlara bakarsak;

- Mekanik üretim
- Kurtarma
- Uzay ve deniz araştırmaları
- Nükleer santraller
- Sağlık hizmetleri
- Madenler
- Tarım
- Askeri ve savunma
- Eğitim

gibi alanlar örnek gösterilebilir.

Kurtarma robotları, deprem, sel, büyük yıkımlar gibi afetlerden sonra kullanılarak, arama – kurtarma görevini üstlenebilirler (Şekil 1.1). Robotlar benzer şekilde uzay araştırmalarında da kullanılabilecek yeni bir sınıftır. NASA'nın Mars gezegeninde görev yapmış robotları, Spirit (Şekil 1.2 a) ve Opportunity (Şekil 1.2 b), robotların uzay çalışmalarında kullanımına örnek gösterilebilir.



Şekil 1.1: Deprem kurtarma robotu Hibiscus [3]



(a) (b)

Şekil 1.2: NASA'nın Mars robotları (a) Spirit, (b) Opportunity [4]

Günümüzde robotların kullanım alanlarına baktığımızda en yoğun olarak sanayi de karşımıza çıktıkları görülmektedir. Özellikle fabrikaların üretim bantlarında robotlar, hassasiyet ve çaba gerektiren işleri, süratle ve hatasız bir şekilde yapabilmektedirler. Otomotiv sanayinin en sık kullanıldıkları alan olduğu söylenilebilir.

Sanayi dışında insanların çalışmasının tehlikeli olduğu ortamlarda da robotlar tercih edilmektedir. Nükleer enerji santralleri, madenler, deniz ve yeryüzü (volkan) araştırmaları bu alanlara örnek gösterilebilir.

Savunma sanayi ve askeriyede ise insansız hava ve kara aracı projelerinde, robotlar tercih edilmektedir. Böylece en az zayiat ile düşmana ağır kayıplar verdirilebilir. Ülkemizde de bazı yerli firmalar bu alanlarda çalışmalar yürütmektedir (Şekil 1.3).

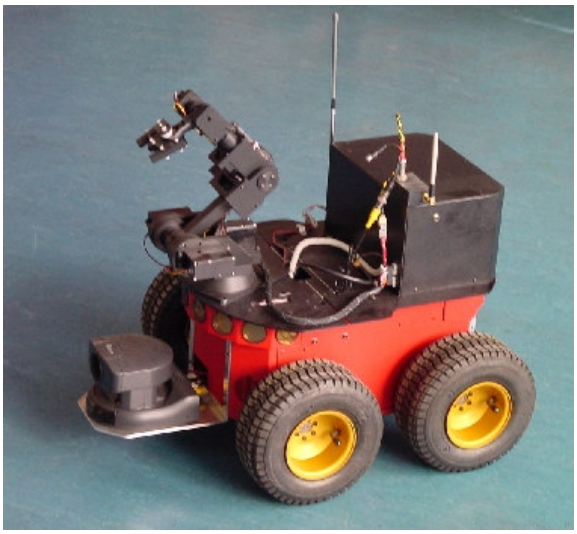


Şekil 1.3: ASELSAN tarafından gerçekleştirilen insansız araç projesi İZCİ

Robotlar özelliklerine göre çeşitli alt gruplara ayrılmaktadırlar. En basit olarak robotlar mobil ve mobil olamayan robotlar olmak üzere iki gruba ayrılabilir. Robotların hareket alanları içerisindeki noktalar kümesinin birleşimine çalışma uzayı denilmektedir. Koordinat sistemine göre bir robotun uzayı yer değiştirmiyorsa bu robot sabit bir robot olarak adlandırılmaktadır (Şekil 1.4). Eğer çalışma uzayı, koordinat sistemine göre yer değiştiriyorsa bu robot, mobil robot olarak tanımlanmaktadır (Şekil1.5).



Şekil 1.4: Lynx – 5 Robot kolu (sabit robot) [5]

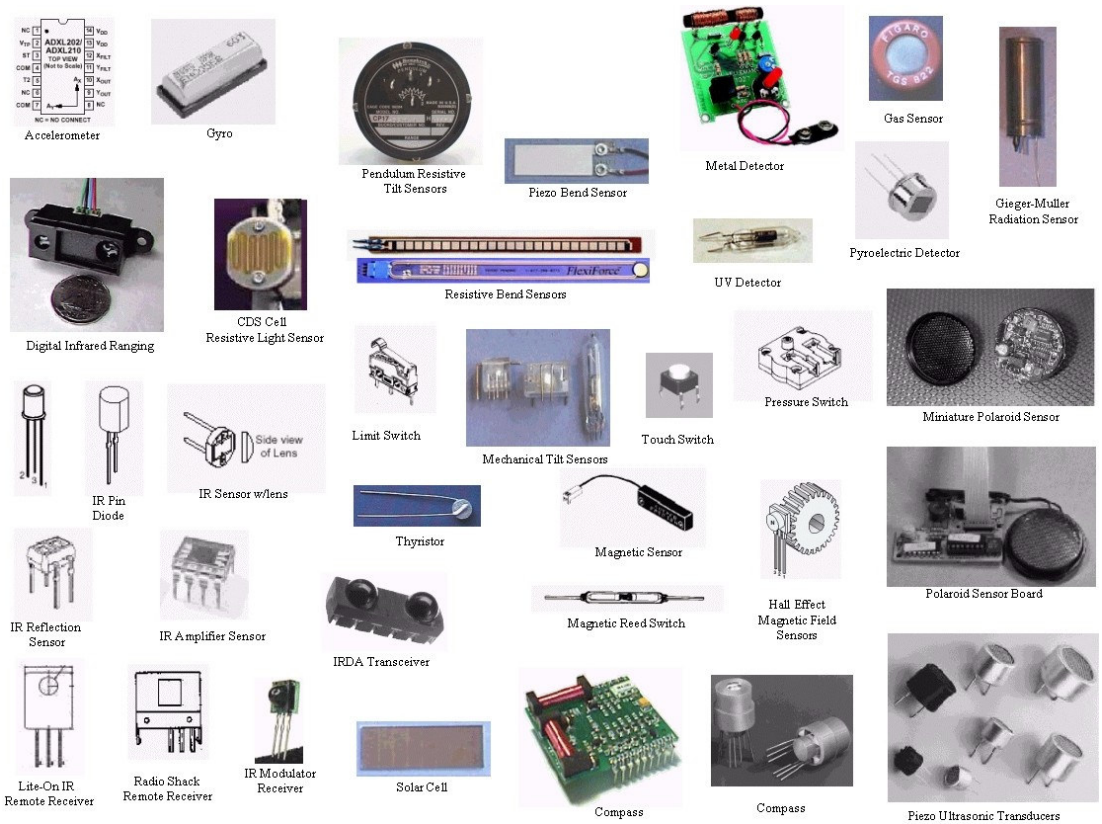


Şekil 1.5: Mobil (hareketli) robot P2AT

1.1. Robot Algısı

Robotlar dış dünya ile çok çeşitli algılayıcılar sayesinde iletişim kurabilirler. Kullanılan sensörler (Şekil 1.6) robotlarda iki gruba ayrılabilir. Bunlardan ilki iç durum sensörleridir. İç sensörler, robotun kontrolünü sağlayan; eklemlerde, motorlarda, pozisyon veya hız gibi değişkenleri saptayacak tipte sensörlerdir. Dış durum sensörleri ise robotun dış dünya ile iletişimini sağlayan; mesafe, ışık, dokunma, basınç gibi değişkenleri saptayan sensörlerdir [6].

Basınç, ışık, ısı, mesafe sensörleri robot kontrolünde ve performansında çok önemli roller üstlenseler de, kullanılan duyusal algıların en güçlüleri; görme ve duyma yeteneğidir. Robotların insanlara benzetilmeye çalıştığını kabul edersek, tıpkı insanlarda olduğu gibi robotlarda da görme ve işitme en önemli iki duyudur.



Şekil 1.6: Sensör örnekleri [7]

Görme işlemi robotlarda kameralar aracılığıyla alınan görüntü sinyallerinin imge işleme teknikleri ile bir dizi işlemden geçirilmesiyle gerçekleştirilir. Böylece robot, hareket yönü ve mesafe algılama, nesne tanımlama gibi yeteneklere sahip olabilir. Genellikle, makine görüşü (machine vision) olarak adlandırılan bu sistemlerde çift kanallı görüş (stereo vision) denilen, birbirinden belirli mesafe uzaklıktaki iki kameranın kullanımı ile robot tıpkı insanda olduğu gibi bulunduğu ortamda derinlik, uzaklık gibi tanımlamalar yapabilmektedir. Robot görüşü temel olarak 6 kısımdan oluşmaktadır. Bu kısımlar;

- hissetme,
- önişlem,
- parçalara bölme,
- açıklama,
- tanıma,
- yorumlama,

olarak ayrılmıştır.

Benzer şekilde ses işlemede de insan kulağı model alınmıştır. Robotlarda ses işleme temel olarak iki değişkene göre uygulanır. Bunlardan birisi ses tanıma, değeri ise çoklu mikrofonlar kullanılarak sesin şiddetinin veya mikrofonlar arasında oluşan zaman farkının hesaplanıp işleme sokulmasıdır. Bu sistemler ile robot dışarıdan gelen seslerin kaynağını tespit edebilir ya da söylenen komutu algılayabilir.

1.2. Tezin İçeriği

Bu tez çalışmasında dış dünya ile etkileşimde bulunurken ses ve görüntü sinyallerini işleyen bir robot üzerinde çalısılmıştır. Robot ses kaynağının yerini tespit etmeye çalısmakta, ses kaynağının bulunduğu yeri tespit ettikten sonra sese yönelmektedir. Ses kaynağının yerinin tespit edilmesinde mikrofonlar arasında oluşan zaman farkından faydalanılmaktadır.

Tezin ilk kısmında genel olarak robotlardan bahsedilmektedir. İkinci kısımda teorik bilgiler üzerinde durulmuş, üçüncü ve dördüncü kısımlarda sinyal işleme uygulamalarından, ses ve görüntü işlemeden, yazılımdan bahsedilmiştir. Beşinci kısımda ise tezde kullanılan donanımla ilgili ayrıntılı bilgi verilmiştir.

2. TEORİK BİLGİLER

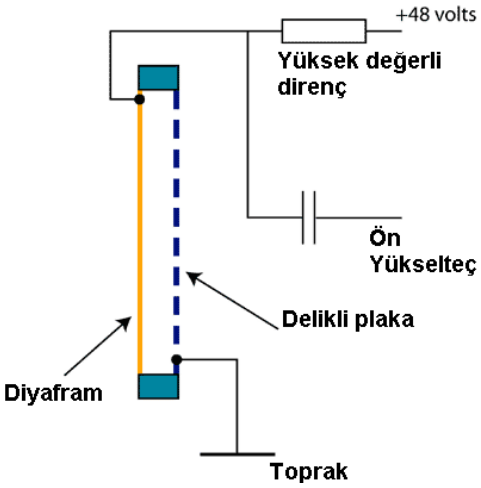
2.1. Mikrofon

Ses işaretlerini algılayıp, elektriksel işaretlere dönüştüren elektromekanik elemanlara mikrofon denir. İki işaret çeşidi arasında dönüşüm yaptıkları için mikrofonlar transduser olarak tanımlanabilirler. Ön kısımlarında bulunan diyafram tüm mikrofon çeşitlerinde ortak kullanılan yapıdır. Havada dağılan ses işaretleri, diyaframa ulaştığında, diyaframa çarparak basınç uygular ve diyaframı titreştirirler. Diyafram arkasında bulunan yapılar sayesinde bu titreşimler elektriksel işarete dönüştürülür. Mikrofonlar, diyaframın arkasında bulunan bu yapılara göre çeşitli sınıflara ayrılmaktadır.

2.2. Mikrofon Türleri

2.2.1. Kapasitif mikrofonlar

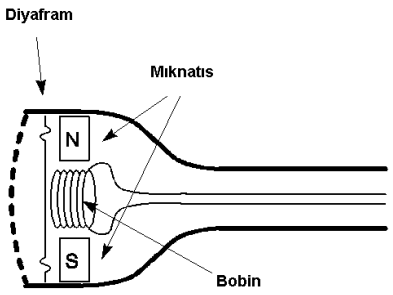
Kapasitif mikrofon, diyaframın titremesi ile birlikte gelen işaretin şiddetine göre içerisindeki kapasitenin değeri değişen mikrofon tipidir. İçerisinde bulunan kapasitenin bir elektrotu diyaframa bağlı olarak hareket ederken diğer elektrotu sabittir (Şekil 2.1). İşaret geldiğinde hareket eden diyafram, sabit yük altında bulunan kapasitenin elektrotları arasındaki mesafenin ve dolayısıyla uçlarındaki gerilimin değişmesine sebep olmaktadır. Bu şekilde ses işareti, elektriksel işarete dönüştürülmektedir. Kapasitif mikrofonlar genellikle, yüksek ses kalitesi gerektiren sistemlerde kullanılmaktadır.



Şekil 2.1: Kapasitif mikrofonun yapısı

2.2.2. Dinamik mikrofonlar

Dinamik mikrofonlarda diyafram bir bobine bağlıdır (Şekil 2.2). Diyafram titrediğinde, üzerindeki bobinler içine yerleştirilmiş oldukları silindirik mıknatıs üzerinde hareketlenerek salınım yaparlar ve elektriksel işaret oluştururlar. Bu tip mikrofonların oluşturdukları elektrik sinyallerinin değeri 1-10mV düzeyinde oldukça düşük işaretlerdir. Dinamik mikrofonlar, bir ön yükselteç ile birlikte kullanılırlar.

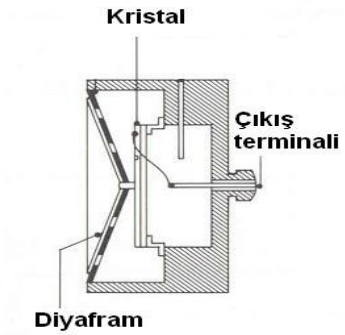


Şekil 2.2: Dinamik mikrofonun yapısı

2.2.3. Piezoelektrik kristalli mikrofonlar

Doğada kristal yapılı bazı maddeler (kuartz, baryum vb.) , basınç uygulandığında üzerlerinde elektrik akımı oluştururlar. Piezoelektrik kristalli mikrofonlar da bundan faydalanılarak geliştirilmiştir. Uygulanan basıncın frekansına ve kuvvetine göre üretilen elektrik akımının şiddeti değişir. Kristalli mikrofonlarda iki metal elektrot

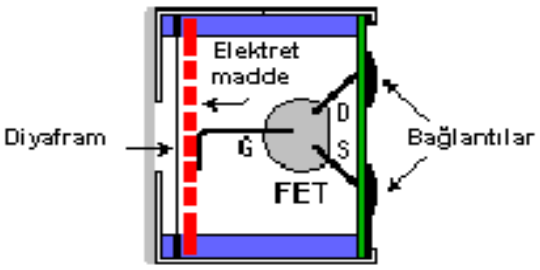
arasına bu kristal madde yerleştirilmiş ve diyaframa tutturulmuştur (Şekil 2.3). Diyafram hareket ettiğinde kristal de hareketlenir ve alternatif (AC) elektrik akımı oluşturur. Üretilen akımın değeri çok küçük olduğundan içersinde bir ön yükselteç bulunur. Piezoelektrik kristalli mikrofonlar içlerinde kristal bulunduğundan, sarsıntı, güneş ışığı, nem gibi etkenlere karşı hassastır.



Şekil 2.3: Piezoelektrik kristalli mikrofonun yapısı

2.2.4. Elektret mikrofonlar

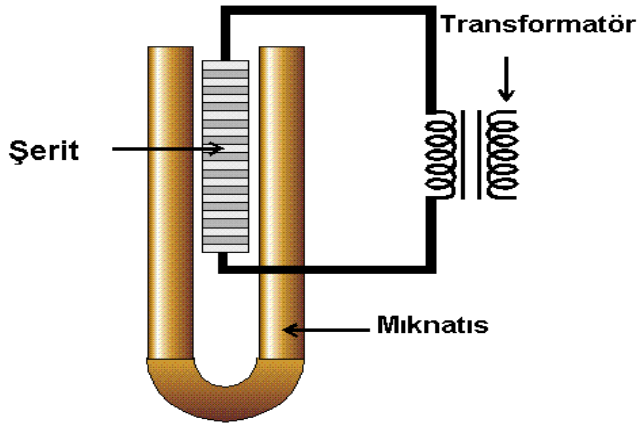
Elektret mikrofonlarda diyafram yarı iletken halka şeklinde bir maddeye bağlanmıştır (Şekil 2.4). Bu maddenin iki yüzü üretimde elektrostatik çeşitli aşamalardan geçirilerek (+) ve (-) yükle elektriklendirilmiştir. Elektriksel yükler bu maddenin özelliğinden dolayı senelerce değerini korur. Elektret madde üzerinde elektrotlar vardır ve kristal mikrofonlardaki kristal gibi diyaframa bağlanmıştır. Diyafram titrediğinde elektret maddenin moleküler yapısı değişerek elektrotlar arasında elektrik akımı oluşmasını sağlar.



Şekil 2.4: Elektret mikrofonun yapısı

2.2.5. Şeritli mikrofonlar

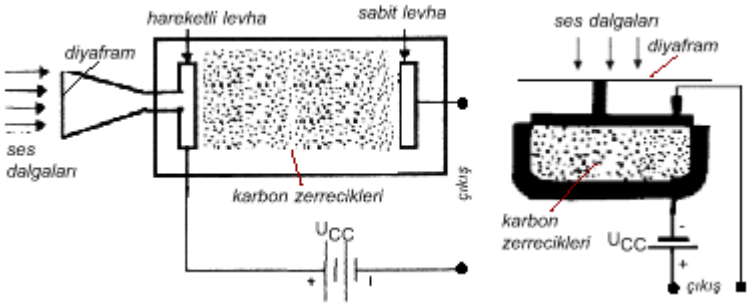
Şeritli mikrofonlarda karşılıklı yerleştirilmiş iki mıknatıs arasında oluşan manyetik alana ince bir kalay veya alüminyum şerit yerleştirilmiştir (Şekil 2.5). Ses işaretleri çarptığında şerit hareketlenir ve manyetik alan içerisinde hareket eder. Bu da şeridin iki ucu arasında elektrik akımı oluşmasına sebep olur.



Şekil 2.5: Şeritli mikrofonun yapısı

2.2.6. Karbon tozlu mikrofonlar

Karbon tozlu mikrofonlarda diyaframın arkasında sıkıştırılmış karbon tozu bulunur (Şekil 2.6). Ses işaretleri diyaframa çarpınca titreştirerek arkasında bulunan karbon tozlarının sıkışıp gevşemesine yol açarlar. Tozlar sıkıştığında iki ucu arasındaki mesafe azaldığından akımın geçiş yolu kısalır ve direnç azalır. Tozlar gevşediğinde ise akımın geçiş yolu uzar ve direnç artar. Bu direnç değişiminden faydalanılarak diyaframa çarpan ses işaretleri elektriksel işarete dönüştürülmüş olur. Karbon tozlu mikrofonlar dışarıdan doğru akım besleme kaynağına ihtiyaç duyarlar. Bu mikrofon tipi kullanım olarak artık pek tercih edilmemektedir. Bunun sebebi ise mikrofon içerisinde bulunan karbon tozlarının zamanla özelliğini kaybetmesidir. Tozların özelliğini kaybetmesi mikrofon hassasiyetinin azalmasına sebep olur.



Şekil 2.6: Karbon tozlu mikrofonun yapısı

2.3. Çoklu Mikrofon Uygulamaları

Çoklu mikrofon uygulamaları, eşzamanlı çalışan, farklı noktalardaki birden çok mikrofonla gerçekleştirilir. Ses işleme uygulamalarında çeşitli amaçlarla kullanılabilirler. Çoklu mikrofon uygulamalarıyla yapılan çalışmalara örnek vermek gerekirse;

- Ses kaynağının yerinin kestirimi,
- Çok konuşmacı bulunan video konferans görüşmelerinde kamera yönlendirmesi,
- İşitme cihazları,
- Gürültü bulunan ortamlarda ses sinyallerinin gürültüden ayrıştırılması,
- Ses tanımlama sistemlerinin geliştirilmesi.

2.3.1. Ses kaynağının yerinin kestirimi

Ses kaynağının yerinin kestirimi birçok canlı tarafından çevreyi tanımlama ve keşfetme aracı olarak kullanılmaktadır. Robot biliminde temel amaç, insanlar gibi duyabilen, görebilen ve bu duyup gördüklerini yorumlayıp ona göre hareket eden robotlar tasarlamaktır. Ses kaynağının yerinin kestirilmesi, ortamda hareketli bir kaynak bulunduğunda kaynağın yerinin gerçek zamanlı olarak bulunması amacıyla kullanılır.

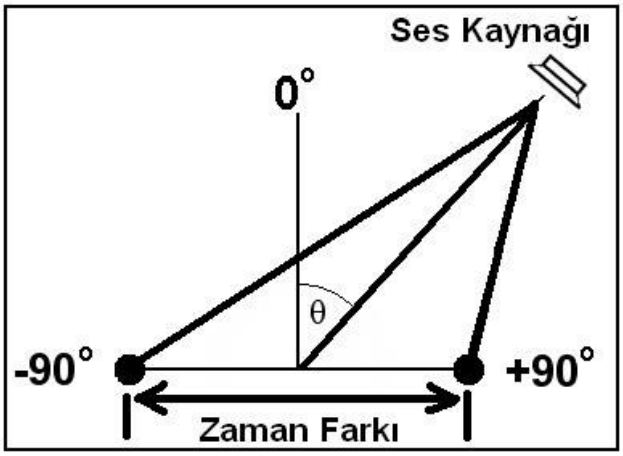
2.3.2. Kamera yönlendirilmesiyle konuşmacının izlenmesi

Konuşmacıların hareketli olduğu ya da birden fazla konuşmacının olduğu ortamlarda, özellikle video konferans sistemlerinde, kamera sisteminin konuşmacıya doğru yönlendirilmesinde yine çoklu mikrofon uygulamaları kullanılmaktadır. Bu sistemlerde kamera, konuşmacı konuşmaya başladığında, oda içerisinde konuşmacının yerini tespit edilip o yöne çevrilmektedir. Çok konuşmacı bulunan ortamlarda, toplantılarda otomatik olarak kameranın konuşmacıları dolaşması bu şekilde sağlanabilir.

3.SES KAYNAĞININ YERİNİN KESTİRİLMESİ

Mikrofon dizisi tabanlı ses kaynağının yerinin saptanmasında, en az iki mikrofon kullanılması gerekir. İki mikrofon ile çeşitli ses işareti işleme teknikleri kullanılarak kaynağın koordinatları, pozisyonu oda yada benzeri akustik ortamlarda belirlenebilir.

Ses kaynağının, $s(t)$ ses sinyalini yaydığı varsayılırsa, ses dalgaları her bir mikrofona farklı zamanlarda ulaşır ve iki mikrofona gelen ses sinyalleri arasında Şekil 3.1'de de görüldüğü gibi ölçülebilir bir zaman farkı oluşur.

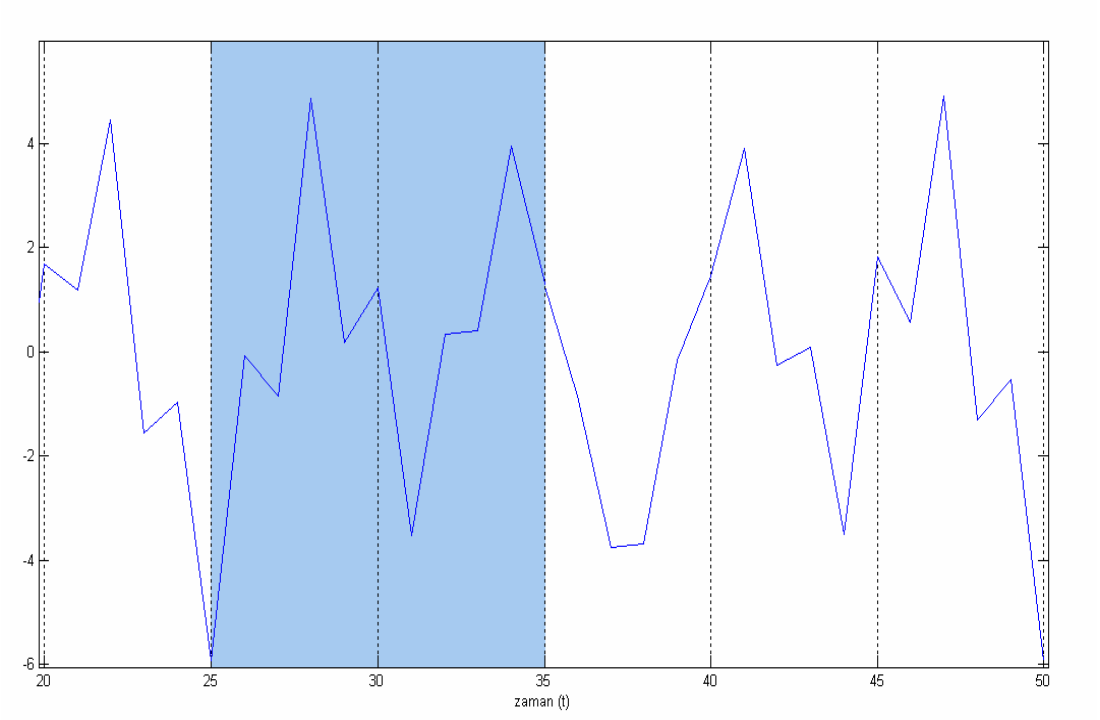


Şekil 3.1: Ses kaynağından mikrofonlara gelen sinyaller arasındaki gecikme

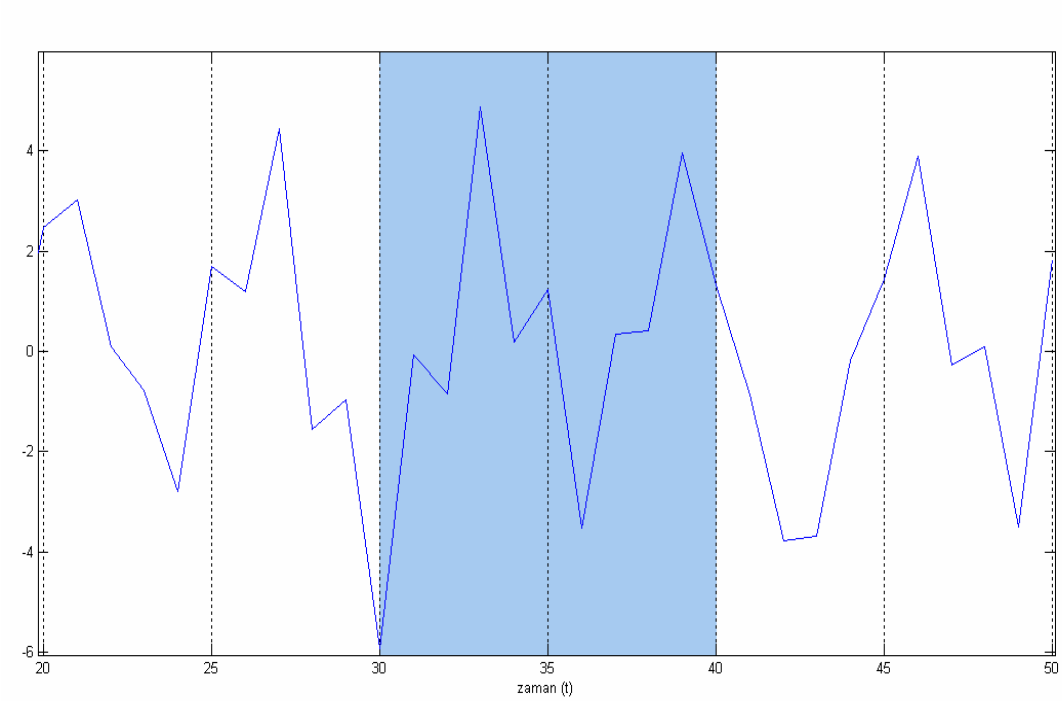
İki mikrofon tarafından alınan ses sinyalleri $g(t)$ ve $h(t)$ olarak adlandırılırsa, ses kaynağının konumunun bulunabilmesi için bu sinyallerin maksimum korelasyon noktası (sinyallerin üst üste bindirildiğinde en uyumlu oldukları nokta) tespit edilmelidir [8]. Bunun için çapraz korelasyon fonksiyonu kullanılabilir. $g(t)$ ve $h(t)$ sinyallerinin çapraz korelasyonu,

$$Corr(g,h)_j(t) \equiv \sum_{k=0}^{N-1} g_{j+k} h_k \tag{3.1}$$

şeklinde tanımlanmaktadır.



Şekil 3.2: Sağ mikrofondan alınan ses işaretinin vektör grafiği



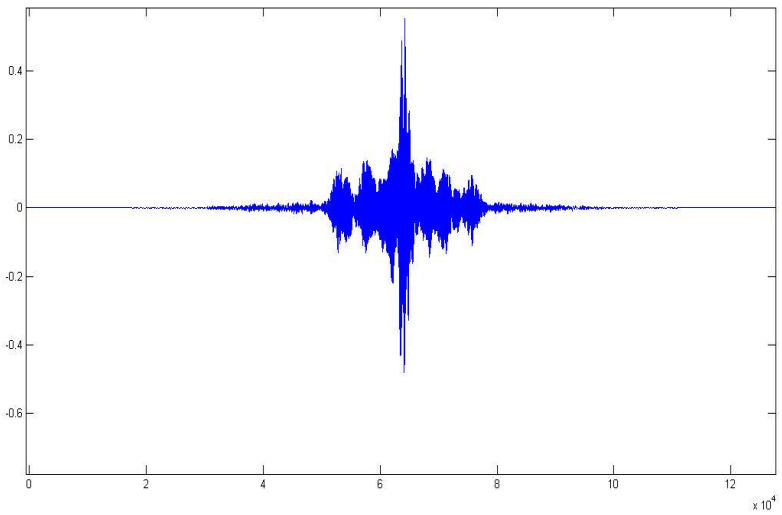
Şekil 3.3: Sol mikrofondan alınan ses işaretinin vektör grafiği

Çapraz korelasyon, $g(t)$ sinyalinin $h(t)$ sinyali üzerinde kaydırılarak, iki sinyalin karşılıklı örnek değerlerinin çarpılıp toplanması şeklinde hesaplanmaktadır. Bu şekilde çapraz korelasyon iki vektörü karşılaştırmak için kullanılabilmektedir. Bu fonksiyona giren vektörlerin A ve B olduğunu kabul edersek, bu vektörler çapraz korelasyon fonksiyonuna sokulduğunda elde edilecek C vektörünün uzunluğu,

$$Uzunluk(C) = (Uzunluk(A) + Uzunluk(B)) - 1 \tag{3.2}$$

olarak hesaplanabilir [8].

olarak hesaplanabilmektedir. Elde edilen C vektöründeki en yüksek değer, girişteki iki sinyal arasındaki maksimum korelasyonu ve zaman gecikmesini (σ) vermektedir. Şekil 3.2 ve 3.3'te birbirine 30cm uzaklıkta olan iki mikrofon tarafından kaydedilen ses işaretleri (A ve B) gösterilmektedir. Şekil 3.4'de ise bu iki işaretin çapraz korelasyon vektörü, C, verilmektedir. C vektörünün grafiğinde x ve y olmak üzere iki eksen bulunmaktadır. x ekseni zaman gecikmesini (σ) gösterirken, y ekseni ise verilen zaman gecikmesine karşılık gelen çapraz korelasyon değerini göstermektedir. İki giriş sinyali $g(t)$ ve $h(t)$ arasında zaman gecikmesi olmaması durumunda (ses kaynağı 0° de), çapraz korelasyon vektöründeki en yüksek değer vektörün tam orta noktasında olacaktır.



Şekil 3.4: Korelasyon vektörünün grafiği

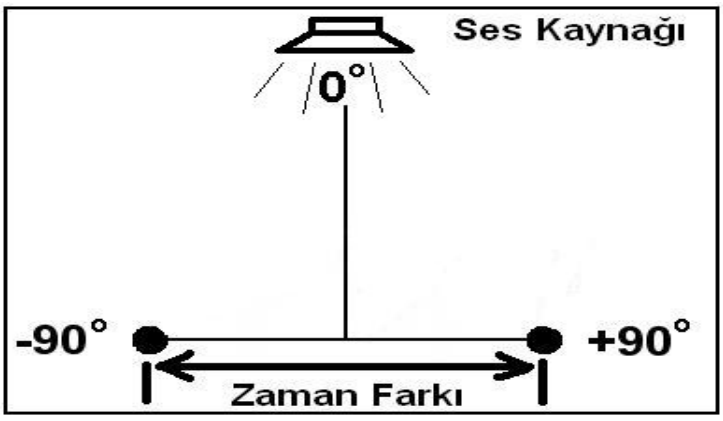
Bu şekilde iki kanaldan aldığımız A ve B vektörleri gibi ses işaretleri arasındaki faz kayması yada zaman gecikmesini hesaplarken görüldüğü gibi çapraz korelasyon yöntemini kullanılabilir.

3.1.Zaman Farkı ve Çapraz Korelasyon Kullanılarak Kaynağın Konumunun Saptanması

Ses kaynağının yerinin tespit edilmesinde açı değeri, çapraz korelasyon metodu ile bulunan C vektöründeki en yüksek değere karşılık gelen gecikme değerinin (σ) tespit edilmesi ile saptanabilir. Zaman gecikmesi, C vektöründeki en yüksek değerin vektörün orta noktasına olan uzaklığı ile hesaplanabilir. Eğer ses kaynağı Şekil 3.5'te görüldüğü gibi iki mikrofonun tam ortasında ise bulunan en yüksek değerin vektör üzerindeki yeri C vektörünün tam ortasında bulunacaktır. C vektörünün en yüksek değeri vektörün orta noktasından geride yada ileride olabilir. En yüksek değer orta noktanın gerisinde olması negatif, ilerisinde olması ise pozitif zaman gecikmesine karşılık gelmektedir. Bulunan bu gecikme değeri kullanılarak ses kaynağının konumu tespit edilebilir. Ancak bunun için de yine bazı değişkenlerin hesaplanması gerekmektedir. Belirlenmesi gereken ilk değişken iki örnek değeri arasındaki zaman farkıdır (Δ) [8].

$$\Delta = 1/f_s = 1/44.1*10^3 = 2.2676*10^{-5} \text{ s} \quad (3.3)$$

Burada f_s örnekleme frekansı olup, değerinin 44,1 kHz olduğu varsayılmıştır.



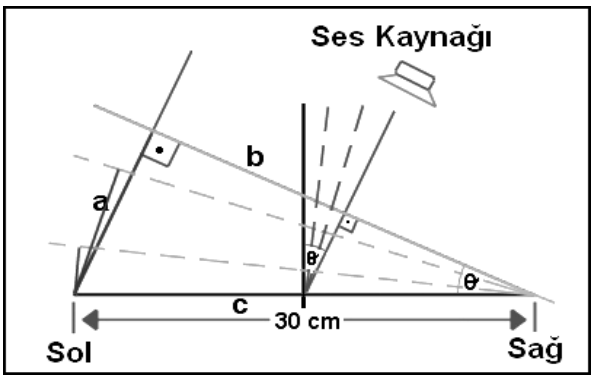
Şekil 3.5: Ses kaynağının mikrofonların ortasında olma durumu

Ses dalgalarının kaynaktan çıktıktan sonra Şekil 3.6 'da görüldüğü gibi kaynağa paralel şekilde *b* çizgisi yönünde ilerlediğini farz edersek, sesin ses kaynağından geliş açısını hesaplamak için şekildeki üçgen içerisinde bulunan bilinmeyen değişkenleri bulmamız gerekmektedir. Temel trigonometrik fonksiyonlar kullanılarak bu değişkenler hesaplanabilir.

$$Sin\theta = \frac{a}{c} \tag{3.4}$$

$$Cos\theta = \frac{b}{c} \tag{3.5}$$

$$Tan\theta = \frac{c}{b} \tag{3.6}$$



Şekil 3.6: Ses kaynağının açı değerinin hesaplanması

Şekil 3.6' da θ ile gösterilen açının bulunması için öncelikle üçgenin kenar uzunlukları olan *a*, *b* ve *c*' den ikisinin uzunluğunun bilinmesi gereklidir. Buradaki *c* uzunluğu, robot üzerinde bulunan mikrofonlar arası uzaklıktır ve değerinin 30cm olduğu varsayılmıştır. Ses hızından ve zaman gecikmesinden faydalanılarak üçgenin *a* kenarı da hesaplanabilir. *a* uzunluğunun hesaplanmasında "ses hızı x zaman" formülünden faydalanılmıştır. Bu formülde zamanı hesaplamak için aşağıdaki formül kullanılabilir.

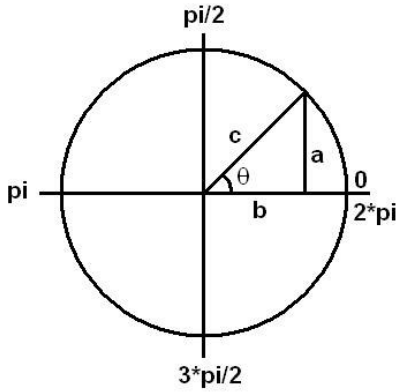
$$t=\Delta*\sigma \tag{3.7}$$

Burada Δ değeri, formül 3.3'te verilen iki örnek arasındaki zaman farkıdır. σ çapraz korelasyon fonksiyonundan saptanan gecikme örneklerinin sayısıdır. Bu değişkenler bulunduktan sonra, *t* bulunur ve formül 3.8'den *a* 'nın uzunluğu hesaplanabilir.

$$\text{Uzunluk} = t*V_{ses} = (\Delta*\sigma)*V_{ses} \tag{3.8}$$

$$a=\sin(x) \tag{3.9}$$

$$b=\cos(x) \tag{3.10}$$



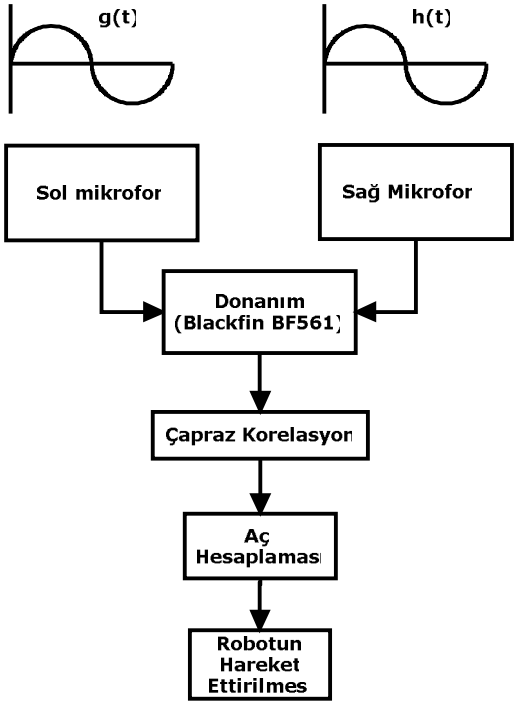
Şekil 3.7: Sin(x) ve Cos(x) 'in geometrik diyagramı

Formülde ses hızı normal şartlar altında (24°C oda sıcaklığı ve deniz seviyesinde) *V*=384m/s olarak alınmıştır. Şekil 3.6 'da *a* ve *c* uzunlukları bulunduktan sonra yukarıda verilen trigonometrik denklemler kullanılarak, ses kaynağının iki mikrofonun merkezine göre konumunu veren θ açısı hesaplanabilir (Şekil 3.7) [8].

$$Sin\theta = \frac{a}{c} \Rightarrow \theta = Sin^{-1}\frac{a}{c} \tag{3.11}$$

$$\theta = Sin^{-1}\frac{(\Delta*\sigma)*V_{ses}}{c} \tag{3.12}$$

3.2.Ses Mimarisi



Şekil 3.8: Temel Mimari

Şekil 3.8 de ses kaynağının yerinin tespit edilmesindeki aşamalar gösterilmektedir. Görüldüğü gibi önce aralarındaki uzaklık 30cm olarak ayarlanmış, iki mikrofondan 250ms uzunluğunda ses işaretleri ($g(t)$ ve $h(t)$), 48kHz örnekleme frekansı ile alındıktan sonra donanım tarafından kaydedilmektedir. Kaydedilen işaretler bu işlemin ardından çapraz korelasyona sokulmaktadır. Çapraz korelasyon sonucunda hesaplanan açı değeri baz alınarak, mobil robot ses kaynağına doğru yönlendirilmektedir.

4.HAREKET KESTİRİMİ

Hareket, bir nesnenin zaman içerisinde, şeklinde ya da konumunda olan değişiklik olarak tanımlanabilir. Hareket kestirimi ise, nesnenin zaman içindeki hareketlerinin tespit edilip bundan sonuç çıkarılması olarak tanımlanabilir. Hareket kestirimini ele alırsak, kameradan alınan görüntü çerçeveleri arasındaki farkların, değişimlerin belirlenmesidir. Videoda çerçeveler arası hareket kestirimi, değişimin yön ve miktarını gösteren vektörler dizisi olarak tanımlanabilir.

Hareket kestiriminde, kestirim işlemi ilk önce ardışık çerçeveler arasındaki hareket eden ve etmeyen kısımların belirlenmesi ile başlar ve böylece çerçevenin arkaplanı ve hareket eden nesne birbirinden ayrılmış olur.

Kestirim algoritmasında aşağıdaki bilgiler kullanılır;

- Değişkenlerin zamanla farklılıkları,
- Rastgele faktörlerin karakteristik dağılımları,
- Öncelik bilgisi.

İmge işleme tekniklerini verimli bir şekilde kullanılabilmesi için, görüntüde hangi alanların hareket ettiğinin ve onların hareket vektörlerinin zaman ekseninde bulunması gerekmektedir. Geliştirilmiş çok sayıda hareket kestirim metodu bulunmaktadır. Bunların birbirine üstünlüklerini belirleyen kıstaslara bakarsak;

- Doğruluk,
- Çözünürlük (hassasiyet),
- Hata Oranı,
- Hız sınırı,
- Vektör sayısı,

olarak tanımlayabiliriz.

Genelde kullanılan hareket kestirimi algoritmaları ise;

- arka plan farkı yöntemleri
  - basit fark alma yöntemi
  - bedyan filtreleme kullanarak arka plan modelleme yöntemi
  - ağırlıklı toplam yöntemi
  - çift arka plan yöntemi
- istatistiksel Yöntemler
  - benzerlik oranı (Likelihood Ratio) yöntemi
  - maksimum fark yöntemi
  - ağırlık merkezi analizi yöntemi
  - hough dönüşümü yöntemi
  - istatistiksel basit fark alma yöntemi
- görsel akış yöntemi

olarak gösterilebilir.

4.1. Hareket Dengeleme

Hareket dengeleme işleminde ardı ardına gelen iki görüntü çerçevesi içerisinde bulunan hareket vektörleri, ikinci çerçeveye uygulanarak üçüncü çerçeve bulunmaya çalışılır.

Genellikle görüntü kodlama sistemlerinde, bant genişliği azaltma sistemlerinde hareket dengeleme kullanılır. Hareket dengeleme kullanılan sistemlere bakıldığında kestirim hatasının hareket dengeleme kullanmayan sistemlere göre düştüğü ve imge kalitesinin arttığı görülebilir. Hareket dengeleme aynı zamanda hareket sebebi ile oluşan görüntüdeki olumsuz etkileri giderir.

4.2. Hareket Algılamasını Güçleştiren Durumlar

Hareketin çerçeveler arasında algılanmasını güçleştiren durumlara bakarsak, bunlar;

- Işık değişimleri,
- Gölge hareketleri,
- Örtülen ve örtülmeyen bölge,

- Gürültü,
- Açıklık problemi,
- Piksel değerlerinin düzgün dağılıma sahip olması,

olarak sıralanabilir.

4.2.1. Işık değişimleri

Ortamdaki ışık değişimleri, hareket kestirimi sırasında çerçeveler arası gerçekleşen, istenmeyen, algoritmayı yanıltan durumlardan birisidir. İmge üzerindeki piksel gurupları üzerinde yaratacakları değer değişimleri sebebiyle algoritma tarafından hareket olarak algılanabilmektedirler.

4.2.2. Gölge hareketleri

Yine imge içerisinde bulunan bir nesnenin, ortamdaki ışık kaynağının yerinin değişimi yada nesnenin kendi çevresinde hareketi sonucu değişebilmekte ve buda zaman zaman algoritmayı yanıltabilmektedir.

4.2.3. Örtülen ve örtülmeyen bölge

Çerçeveler arasında hareket sırasında hareket eden nesne, hareketi sırasında durağan nesnelerin üzerini kapatabilir, bu durumda çerçeveler arasında durağan nesneler, gizlenip tekrar görünür olacakları için hareketli gibi alınabilir. Bu hareket kestrimi algoritmalarınca istenmeyen bir durumdur.

4.2.4. Gürültü

Çerçeveler arası imgelerde oluşabilecek gürültüler, hareketli nesnenin hareket vektöründe sapmalara yol açabilir. Buda yine hareket kestirim algoritması tarafından istenmeyen durumlara hatalara sebebiyet verebilir.

4.2.5. Açıklık problemi

İmge içerisindeki hareketli nesneler için hareket vektörlerinin değişimlerinin birbirinden bağımsız hareket etmesi sonucu bilinmeyen sayısının artması durumudur. Bu durumda algoritmanın sonucunda çözüm tek olmaz.

4.2.6. Piksel değerlerinin düzgün dağılıma sahip olması

Hareketli nesnenin çerçeve içerisinde hareketinin algılanamama durumudur. İmge içerisinde bir nesnenin hareketinin algılanabilmesi için renk değişimleri olması gerekmektedir. Eğer hareketlilik renk değişimlerine yol açmıyorsa nesnenin hareketi algılanamaz.

4.3. Basit Fark Alma Yöntemi

Basit fark alma yöntemi, görüntüde çerçeveler arasındaki farkı bulmak için kullanılan en temel yöntemlerden birisidir. Fark, t_0 ve t_k zamanlarında alınmış $f(x,y,t_0)$ ve $f(x,y,t_1)$ çerçevelerinin matematiksel farklarının alınmasıyla tespit edilmektedir. Bu çerçevelerden bir tanesi referans (arka plan) görüntüsü, diğer çerçeve ise hareketin tespit edildiği ilk çerçeveye göre aynı hareketsiz nesnelerle beraber hareketi de içeren imgedir. Bu iki çerçevenin farkı alınarak elde edilen imge, hareketsiz nesneleri eleyerek, hareketli bileşenleri ortaya çıkarabilmektedir. t_0 ve t_1 zamanlarında alınmış iki çerçeve için değişiklikleri içeren fark imgesi;

$$D(x,y)=\begin{cases}1, & |f(x,y,t_k)-f(x,y,t_0)|>Th\\ 0, & \text{Diger } Durumlarda\end{cases} \qquad (4.1)$$

şeklinde hesaplanabilmektedir. Bu eşitlikte kullanılan T_h değeri, iki piksel arasında bulunan fark değerinin hareket kaynaklı olup olmadığını tespit etmede kullanılan, önceden tanımlanmış bir eşik değeridir. Algoritma iki çerçeve arasındaki piksel değerlerini karşılaştırılarak eşik değerini aşan değerlerde fark resmine o koordinatta "1" yerleştirilir. Elde edilen fark resmi "0" ve "1" lerden oluşan ikili (siyah – beyaz) bir imgedir [9].

Oluşturulan bu fark görüntüsünün analizinde, $D(x,y)$'deki "1" değerleri hareket olarak tanımlanmaktadır. Bu yaklaşım ışık yoğunluğu sabit olduğunda geçerlidir. Işık yoğunluğunun ortamda değişken olması $D(x,y)$'de gürültü oluşmasına sebep olabilmektedir. Burada gürültüden kasıt, gerçekte hareketli olmayan piksel gruplarının ışık yoğunluğunun değişken olması nedeni ile hareketli gibi algılanmasıdır. Bu durumun aşılabilmesi için genellikle birbirine -4- yada -8- komşuluk ile bağlı piksel değeri "1" olan kümeler sayılarak, küçük alanların elenmesi yöntemi kullanılmaktadır. Bu eleme sonucu ikili fark imgesinde oluşturulan kümelerin gerçek resme ait olma olasılıkları büyük ölçüde artmaktadır [9].

Bu işlemler için oluşturulan algoritmaya göz atarsak;

- t_0 ve t_1 zamanında alınan görüntülerden 4.1'de verilen formül kullanılarak fark resmi oluşturulur.
  - Piksel (x,y) için eğer;
    - Mutlak fark > T_h ise $D(x,y) = 1$
    - Değilse $D(x,y) = 0$
- Oluşan fark imgesinde T_h eşik değerine bağlı olarak -4- yada -8- komşuluk taraması ile hesapla ve temizle olmak üzere iki fonksiyon tanımlanabilir.
  - Piksel (x,y) eğer;
    - $D(x,y) = 1$ ise fonksiyon (Hesapla)
    - Hesaplanan alan > T_h ise Fonksiyon (Temizle)
  - Fonksiyon (Hesapla)
    - Her pikselin -8- komşuluğu için eğer;
    - Komşu piksel = 1 ise fonksiyon (hesapla)
    - Alan bilgisini gönder
  - Fonksiyon (Temizle)
    - Her pikselin -8- komşuluğu için eğer;
    - Komşu piksel = 1 ise fonksiyon (Temizle)
- Bu işlemlerin ardından gürültü temizlendikten sonra harekete ait pikselleri içeren bir fark imgesi elde edilmiş olur.

4.4. İstatistiksel Basit Fark Alma Yöntemi

İstatistiksel basit fark alma yöntemi, basit fark alma yönteminin istatistiksel yaklaşımıdır. Basit fark alma yönteminde yapılan işlem önceden belirlenmiş bir eşik değeri kullanılarak, görüntüler arasındaki farklı pikselleri belirlemektir. İstatistiksel basit fark alma yönteminde ise bu eşik değeri istatistiksel olarak oluşturulmaktadır. Eşik değeri, t_0 - t_{k-1} zaman aralığında alınmış olan $f(x,y,t_0)$, $f(x,y,t_1)$…….. $f(x,y,t_{k-1})$ görüntülerinden alınan fark resimleri aşağıdaki eşitliğe göre belirlenmiştir [10].

$$D(x,y)=\begin{cases}1 & \left|f(x,y,t_k)-\mu_{xy}\right|>N\sigma_{xy}\\ 0 & Diger\ Durumlarda\end{cases} \tag{4.2}$$

Bu eşitlikte ortalama değer (μ_{xy}) aşağıdaki eşitlikten;

$$\mu_{xy}=\frac{1}{K}\sum_{k=0}^{K-1}f(x,y,t_k) \tag{4.3}$$

Ve aşağıdaki eşitlikten standart sapmasına (σ_{xy}) göre belirlenir.

$$\sigma_{xy}=\sqrt{\frac{1}{K}\sum_{k=0}^{K-1}\left[f(x,y,t_k)-\mu_{xy}\right]^2} \tag{4.4}$$

Bu yöntemde hareket görüldüğü gibi, ardışık iki çerçeve arası pikseller arası farktan değil, incelenen çerçevedeki piksel ile hesaplanan μ değeri arasındaki fark hesaplanarak belirlenmektedir.

Ortalama değerleri (μ) ve standart sapmaları (σ) hesaplanan piksellerin normal bir dağılıma sahip oldukları varsayılırsa; eşitlik 4.2'de eşik değeri olarak ifade edilen $N\sigma_{xy}$'deki N katsayısının değerinin 2 olarak alınması, hesaplamaya katılan değerlerin %95'ini, 3 olarak alınması ise %99'unu kapsayacaktır. Bu şekilde eşik değeri artık istatistiksel olarak hesaplanabilir [10].

Bir imge dizisinden, ortalama değeri ve standart sapma değeri piksellerden hesaplanabileceği gibi aynı zamanda MxM boyutunda piksellerden oluşan bloklar kullanılarakta hesaplanabilmektedir. Blok blok imge karşılaştırılması ile piksel karşılaştırma yönteminde karşılaşılan ortamdaki ışık değişikliklerine karşı olan hassasiyet azaltılabilir. (*x,y*) koordinatında bulunan MxM pikselden oluşan bloklara bölünmüş imgedeki her bir blok, piksellerin ortalama değeri aşağıdaki eşitlikle ifade edilebilir [10].

$$\mu_{xy} = \frac{1}{MxM} \sum_{i=\frac{M-1}{2}}^{\frac{M-1}{2}} \sum_{j=\frac{M-1}{2}}^{\frac{M-1}{2}} f(x+i, y+j) \tag{4.5}$$

Bu şekilde her bir blok için ortalama hesaplandıktan sonra değerlendirilmek üzere üç boyutlu bir matrise aktarılır. İmge içerisinde yer alan bütün bloklar incelenip bu inceleme bittikten sonra bir sonraki imgeye geçilir ve aynı işlemler tekrarlanır. $t_0 - t_k$ aralığında yer alan tüm imgelerin ortalama değerleri hesaplandıktan sonra oluşturulan üç boyutlu matrise, blokların K adet ortalama verisi atılmış olur. Her blok için $D(x,y)$ ve μ_{xy} değerlerinin bulunmasıyla blokların karakteristiklerine ulaşılmış olur. Bu işlemlerin ardından $f(x,y,t_k)$ imgesinden, her blok için σ_{xy} kullanılarak fark imgesi tanımlanabilir.

Bu algoritmada yapılan işlemlerin uzun zaman alacağı göz önünde bulundurulursa blok karakteristiklerinin zaman içerisinde güncellenme zorunluluğu ortaya çıkacaktır. Bu aşamada karakteristiklerin güncellenmesi için ortalama çalışma "Running Average" algoritması kullanılmaktadır. Bu algoritmada, ilk elde edilen ortalamalara her yeni imgenin belirli bir orandaki ağırlığı aşağıdaki eşitliğe göre eklenerek ortalamalar güncellenmektedir.

$$\mu_{xyt_{k+1}} = \alpha f(x, y, t_k) + (1-\beta)\mu_{xyt_k} \tag{4.6}$$

Bu eşitlikte α adaptasyon katsayısıdır ve 0,05 gibi küçük bir değerdir. Genellikle α belirlenirken 1/(MxM) olarak alınır. Otalamalar güncellendikten sonraki aşama, standart sapma değerlerinin hesaplanması için üç boyutlu matriste bulunan değerlerin

güncelleştirilmesi işlemidir. Matristeki eleman sayısı her zaman sabit olacaktır. Yeni gelen değerler matrise son eleman olarak eklenir. Eklenen elemanla beraber, elemanlar geriye doğru kaydırılır ve matristeki ilk değer matristen silinir. Bu yöntemlerde kullanılan algoritmalar sonuç olarak şu şekildedir;

- $t_0 - t_{k-1}$ aralığında bulunan K sayıdaki imge için her piksel değerinin ortalaması ve standart sapması hesaplanır.
  - 0'dan k-1'e kadar her imge için;
    - Her imge elemanı için;
      - Ortalamayı bul
      - Standart sapmayı bul
- Videodan alınan $f(x,y,t_k)$ imgesinin pikselleri ortalama ve standart sapma değerleri bulduktan sonra yukarıdaki eşitliğe göre fark imgesini oluşturur.
  - Her blok için
    - Her resim elemanı için eğer;
      - Mutlak fark > $N\sigma$ ise $D(x,y) = 1$
      - Değil ise $D(x,y) = 0$
- İkinci yöntemde ise $t_0 - t_{k-1}$ aralığındaki K adet imge içerisinde bulunan MxM piksele sahip her blok için ortalama hesaplanıp üç boyutlu matrise aktarılır.
  - 0'dan k-1'e kadar her imge için;
    - Her blok için;
      - İmge elemanı için;
        - Ortalamayı hesapla
    - Geçici$_{xyk}$ = ortalama
- $f(x,y,t_k)$ imgesinin tüm blokları, üç boyutlu matris içerisinde bulunan verilerden hesaplanmış olan ortalama ve standart sapma değerleriyle yukarıdaki eşitlikten fark resmi oluşturulur ve güncelleme işlemi yapılır.
  - 0'dan k-1'e kadar her geçici$_{xyk}$ için;
    - Ortalamayı hesapla
    - Standart sapmayı hesapla
  - Her blok için eğer;
    - Mutlak fark > $N\sigma$ ise
      - Her imge için $D(x,y) = 1$

- Değil ise;
- Her imge için $D(x,y) = 0$
  - Ortalamayı eşitlikten güncelle

Yatayda i, düşeyde j sayıda bloğa bölünmüş $f_1(x,y)$ ve $f_2(x,y)$ imgeleri arasındaki farkın belirlenmesi için, ixj sayıda bloğa bölünmüş iki imgedeki her bloğun gri seviye karakteristiği çıkarılmalıdır. Eşitliklerden görüldüğü gibi MxM piksele sahip blok ortalamaları ($\mu_{1_{ij}}, \mu_{2_{ij}}$) ve standart sapmalarının ($\sigma_{1_{ij}}, \sigma_{2_{ij}}$) hesaplanır. Bu şekilde imgedeki bloklar arası benzerlik oranı aşağıdaki eşitlikten bulunabilir.

$$L_{ij} = \frac{\left[\frac{\sigma_{1_{ij}} + \sigma_{2_{ij}}}{2} + \left(\frac{\mu_{1_{ij}} - \mu_{2_{ij}}}{2} \right)^2 \right]^2}{\sigma_{1_{ij}} + \sigma_{2_{ij}}} \tag{4.7}$$

Bu eşitliğin tanımsız olmaması için, kesrin paydasında bulunan $\sigma_{1_{ij}}\ \sigma_{2_{ij}}$ çarpımının "0" değerini almamasına dikkat edilmesi gerekmektedir. Bloklar arası benzerlik oranı bulunduktan sonra, bu oranın eşik değerine göre eşitlikten kıyaslanması ile birlikte D_{ij} fark imgesi elde edilebilir. Fark imgesi elde edilirken benzerlik oranına sahip piksel değerleri 0 olarak kalırken aktif blokların değeri ise 1 olarak gösterilir. Bakıldığında fark imgesinde ortaya iki bitlik siyah – beyaz bir imge çıkmış olduğu görülür [10].

$$D_{ij}(x,y) = \begin{Bmatrix} 1 & Lij > Th \\ 0 & Diger\ Durumlarda \end{Bmatrix} \tag{4.8}$$

Aralarında benzerlik bulunan $f_1(x,y)$ ve $f_2(x,y)$ imgeleri, ardışık olarak alınmış iki imge olabileceği gibi, birinin arka plan değeri hareketli nesne bulunduran iki imge de olabilir. Eğer fark imgesi ardışık imgelerden oluşmuş ise, objenin hızı ve çerçeveler arası zaman farkına bağlı olarak nesne üzerinde bir kuyruk oluşabilir. Eğer ardışık imgelerden oluşan bir görüntü değilse ani ışık yoğunluğu değişimlerinden kaynaklı olarak fark imgesinde gürültü görülebilir. Bu tip bir durumla karşılaşıldığında,

gürültü analizi veya belirli bir değerden küçük kümelerin fark imgesinden temizlenmesi algoritmaları kullanılabilir.

Benzerlik oranı algoritmasına bakarsak;

- Her iki imge için;
  - Her blokij için;
    - Ortalamayı hesapla,
    - Standart sapmayı hesapla;
  - $A1_{ij}$ = Ortalama, $S1_{ij}$ = standart sapma
  - $A2_{ij}$ = Ortalama, $S2_{ij}$ = standart sapma
- İki imgeye ait bütün blokların karakteristiğini barındıran matrisler kullanılarak, her iki imgede aynı koordinata denk gelen matris değerlerinden eşitlikte olduğu gibi faydalanılarak, benzerlik oranı bulunup iki boyutlu matrise aktarılır. Bu benzerlik oranı da eşitlik kullanılarak fark imgesi D_{ij} oluşturulur.

  - Her $A1_{ij}$, $S1_{ij}$, $A2_{ij}$, $S2_{ij}$ elemanı için;
    - Benzerlik oranını hesapla
    - L_{ij} = benzerlik oranı

  - Her $blok_{ij}$ için eğer;
    - $L_{ij} > T_h$ ise;
      - Her eleman (x,y) için;
        - $D_{ij}(x,y) = 1$
    - Değil ise;
      - Her eleman (x,y) için;
        - $D_{ij}(x,y) = 0$

5. DONANIM

5.1. Kontrol Kartı

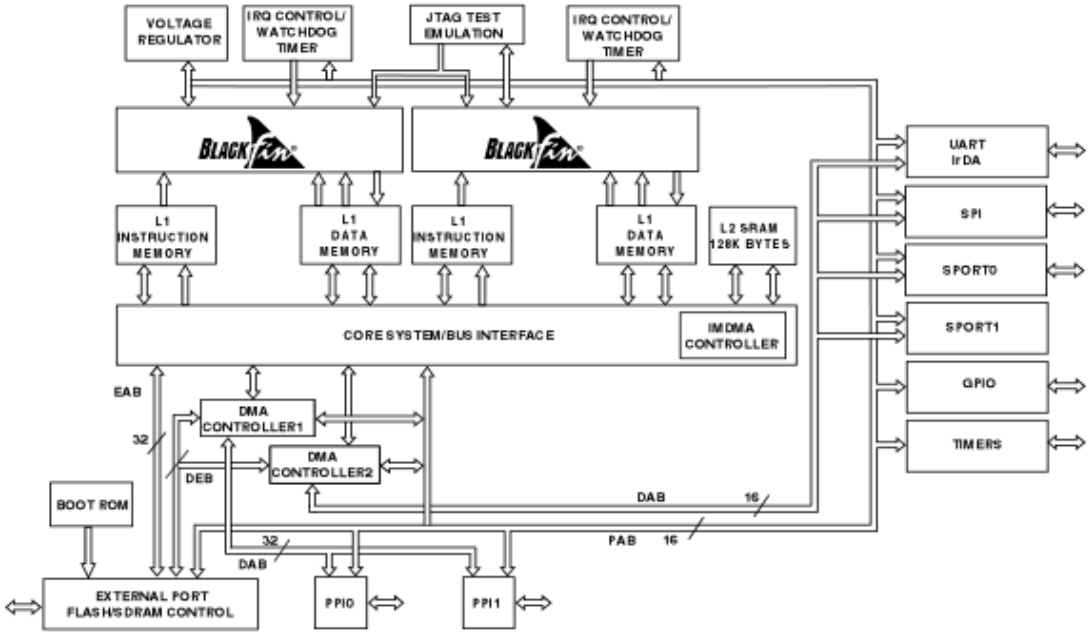
5.1.1. ADSP-BF561 Blackfin simetrik çoklu işlemci

ADSP-BF561 işlemcisi Analog Devices firmasının ürettiği Blackfin işlemci ailesinin performansı arttırılmış bir üyesidir (Şekil 5.1). Çift yüksek performans işlemci çekirdeği ile daha hızlı çalışabilmektedir. İşlemci bu sayede eşzamanlı olarak birkaç işlemi işlemci kolaylıkla yürütebilmektedir. ADSP-BF561 işlemci çekirdekleri, esnek önbellek mimarisi, geliştirilmiş DMA (Direkt Hafıza Erişimi) alt sistemi ve dinamik güç yönetim sistemi (DPM) ile karmaşık kontrolleri ve işaret işleme görevlerini hızlı bir şekilde gerçekleştirebilmektedir.



Şekil 5.1: Blackfin işlemci

İç yapısı Şekil 5.2'de görülen Blackfin işlemciler çok yaygın bir kullanım alanına sahiptirler. Bu işlemciler, endüstri, enstrümantasyon, medikal gibi alanlarda ve tüketici cihazlarında (fotoğraf makinesi, kamera, otomobil vs.) sıkça kullanılmaktadırlar.



Şekil 5.2: Blackfin işlemcinin iç yapısı

İşlemcinin özelliklerine bakarsak;

- 328 KByte çip üstünde hafıza;
  - 32 KByte L1 komut hafızası SRAM/her bir çekirdek için önbellek
  - 64 KByte L1 veri hafızası SRAM/her bir çekirdek için önbellek
  - 4 KByte L1 her bir çekirdek için karalama belleği
  - 128 KByte düşük gecikmeli, paylaştırılmış L2 belleği
- 32 bit bellek kontrol bağlantısı (SDRAM, SRAM, Flash veya Rom)
- İki adet çift kanal, full-dublex, senkronize seri port (8 stereo I2S kanal destekler)
- Çiftli 16 kanal DMA (direkt hafıza erişimi) kontrolörü (tek yada iki boyutlu transferleri destekler).
- SPI Portu
- IrDA destekli UART
- 12 zamanlayıcı (timer) / sayıcı (counter) PWM destekli, Pals genişliği ve sayıcı modları
- 48 programlanabilir bayrak yapısı / genel amaçlı G/Ç
- Denetim birimi
- Çiftli Watchdog zamanlayıcı
- 1x den 63x e kadar frekans çarpanlı PLL

5.1.2. ADSP BF561 EZ-KIT LITE

Analog Devices firmasının ürettiği, ADSP BF561 EZ-KIT LITE geliştirme kartı (Şekil 5.3), ürün geliştiricilere başlangıçta uygun maliyetle Blackfin BF561 işlemci kullanarak, bilgisayarın USB portundan karta bağlanıp ses ve video işleme uygulamalarını gerçekleştirmelerini sağlayan bir donanımdır. Analog ses işleme uygulamaları için kart üzerinde çok kanallı 96 kHz AD1836 kodlayıcı ve kod çözücü (codec) bulunur. Ayrıca video uygulamaları için yine kart üzerinde ADV7183A 10 bit kod çözücü (decoder) ve ADV7179 NTSC/PAL video kodlayıcı (coder) entegreleri bulunmaktadır. Kullanıcı kart üzerinde yer alan ses ve görüntü giriş – çıkış soketlerini kullanarak BF561 çift çekirdek işlemci yardımıyla ses ve görüntü işleme uygulamalarını gerçekleştirebilir.



Şekil 5.3: ADSP BF561 EZ-KIT LITE

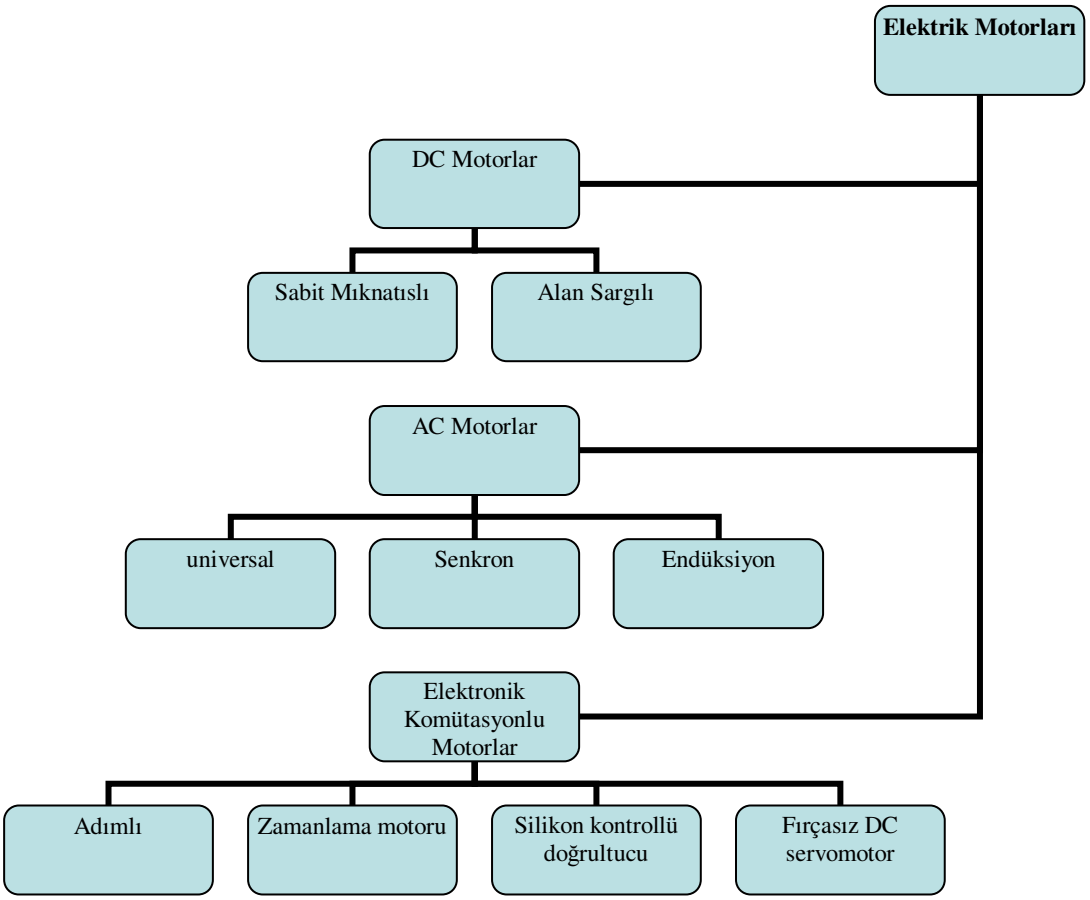
Kartın özelliklerine göz atarsak;

- Analog Devices ADSP-BF561 işlemci
  - 256-pin Mini-BGA (Ball Grid Array) paketi
  - 30 MHz CLKIN osilatör
- Eşzamanlı dinamik rastgele erişimli bellek (SDRAM)
  - 64 MB (16M x 16 bit x 2 çip)
- Kalıcı bellek (Flash Memory)
  - 8 MB (4M x 16 bit)

- Analog Ses Arayüzü
  - AD1836 A - Analog Devices 96 kHz ses kodlayıcı- kod çözücü (codec)
  - 4 giriş RCA soketi (2 Stereo kanal)
  - 6 çıkış RCA soketi (3 Stereo kanal)
- Analog Video Arayüzü
  - ADV7183A video kod çözücü w/ 3 giriş RCA soketi
  - ADV7179 video kodlayıcı w/ 3 çıkış RCA soketi
- Evrensel asenkron veri alışverişi arayüzü (UART)
  - ADM3202 RS-232
  - DB9 erkek soket
- LEDler
  - 20 LED: 1 güç (yeşil), 1 kart reset (kırmızı), 1 USB (kırmızı), 16 genel amaçlı (turuncu), ve 1 USB veri alışverişi (turuncu)
- Butonlar
  - 5 buton: 1 reset, 4 programlanabilir.
- Genişleme arayüzü
  - PPI0, PPI1, SPI, EBIU, Timers11-0, UART, programlanabilir bayraklar, SPORT0, SPORT1
- Diğer özellikler
  - JTAG ICE 14-pin

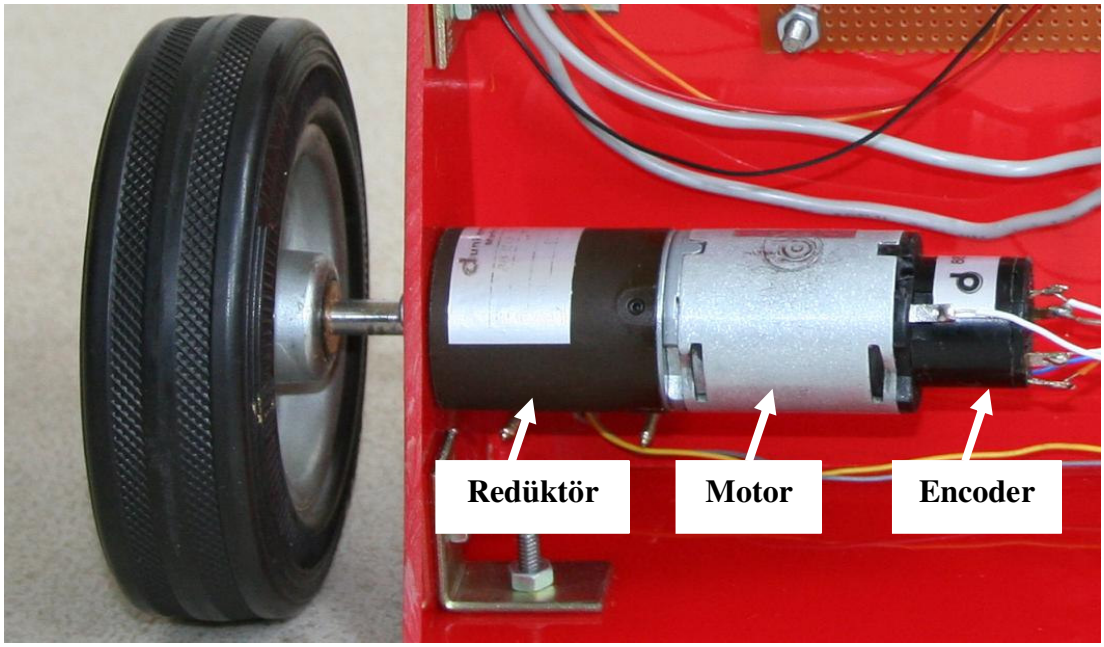
5.2. Elektrik Motorları

Motorlar bilindiği gibi günlük hayatımızda birçok elektromekanik cihazda kullanılmaktadırlar. Elektriksel enerjiyi, mekanik enerjiye dönüştüren elemanlardır. Motorlar temel olarak üç başlık altında incelenebilir. Bunlar DC motor, AC motor ve elektronik komütasyonlu motorlardır (Şekil 5.4). Aşağıdaki şekilde motor çeşitleri ve alt sınıfları görülmektedir.



Şekil 5.4: Motor çeşitleri

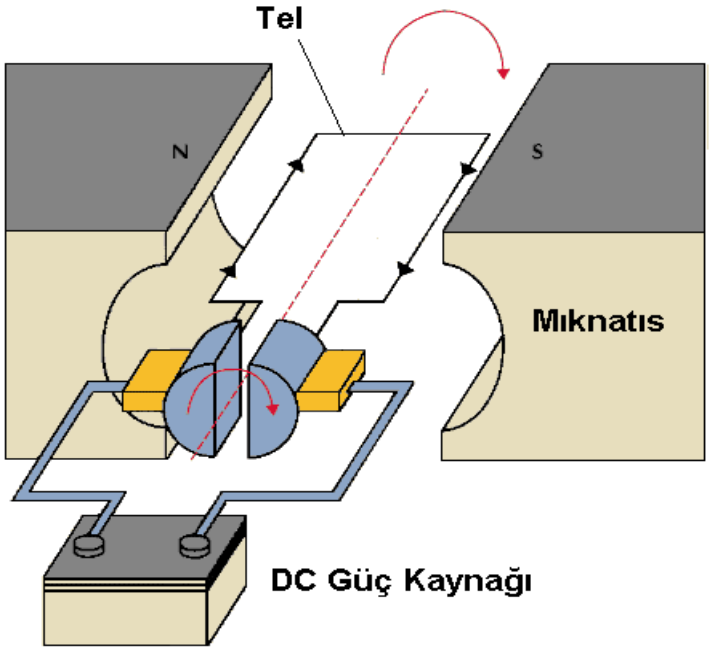
Bu projede Dunkenmotoren firmasından alınan DC motorlar tercih edilmiştir (Şekil 5.5). Bunun sebebi robotun mobil olması, dc motorların redüktör takılarak torklarının arttırılabilmesi, kodlayıcı (encoder) ile beraber kontrol edilebilmeleri ve bu sayede kontrolleri sırasındaki hassasiyetlerinin diğer motorlarla karşılaştırıldığında, kullanılan kodlayıcıya (encoder'a) da bağlı olarak yüksek olmasıdır.



Şekil 5.5: Robotta kullanılan motor

5.2.1. DC motor

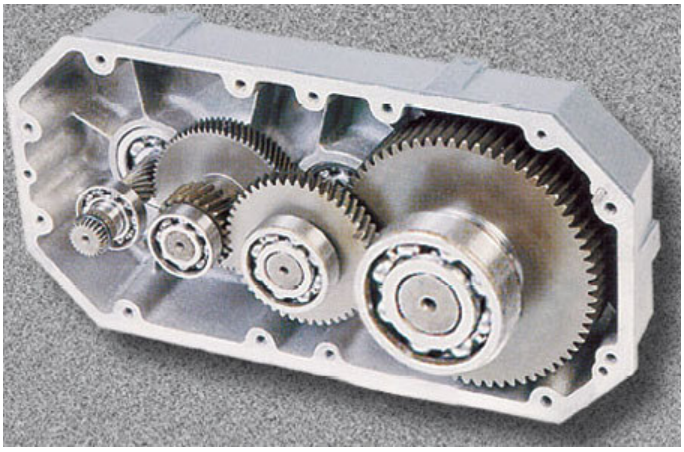
Basit bir DC motor düzeneği gözlemlenirse, sistemin mıknatıslar, akım taşıyan tel sargıları ve elektriksel bir güç kaynağından oluştuğu görülür. İçerisinde bulunan tel, nüve denilen makara benzeri yalıtkan madde üzerine sarılarak bobin oluşturulmuştur (Şekil 5.6). Etrafına elektrik akımının geçtiği tel sarılmış olan bu yapı elektromıknatıs gibi hareket ederek motor içerisindeki sabit mıknatıs kutupları arasına yerleştirildiğinde dönme hareketi yapar. Belirli zamanlarda aşağıdaki şekilde görüldüğü bobin uçlarının kutupları değiştirilerek dönme hareketinin sürekli olması sağlanır. Temel olarak elektrik motorlarında bundan faydalanılır ve bobinin mıknatıs içerisindeki dönüş yönü ve/veya hızı kontrol edilmeye çalışılır. Eğer motor istenildiği gibi kontrol edilebilirse elektriksel enerji, mekanik enerjiye dönüştürülüp istenilen sistem kontrol edilebilir.



Şekil 5.6: DC Motorun Yapısı

5.3. Redüktör

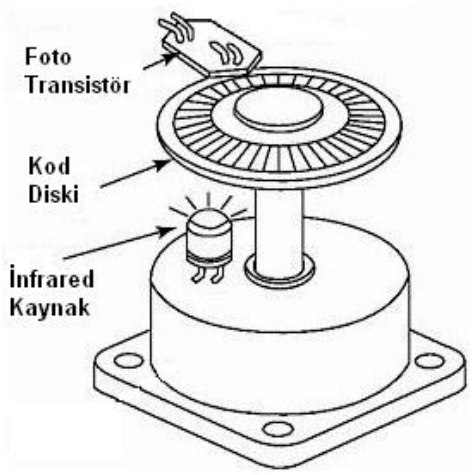
Redüktörler dişli yapılarından oluşurlar (Şekil 5.7). Motorlarda kullanılan bu kapalı dişli sistemleri genellikle motorun hızını düşürmek ve bununla birlikte tork değerini arttırmak için kullanılır. Bu projede de redüktörlü motor seçilmesindeki amaç robotun hareketinin daha kontrollü olması için hareketi yavaşlatmak, motorların torkunu ve motorun arkasında bulunan kodlayıcının çözünürlük değerini arttırmaktır. Projede kullanılan motorlarda bulunan redüktörler 1:36 oranındadır.



Şekil 5.7: Redüktörlerin iç yapısı

5.4. Optik Kodlayıcılar

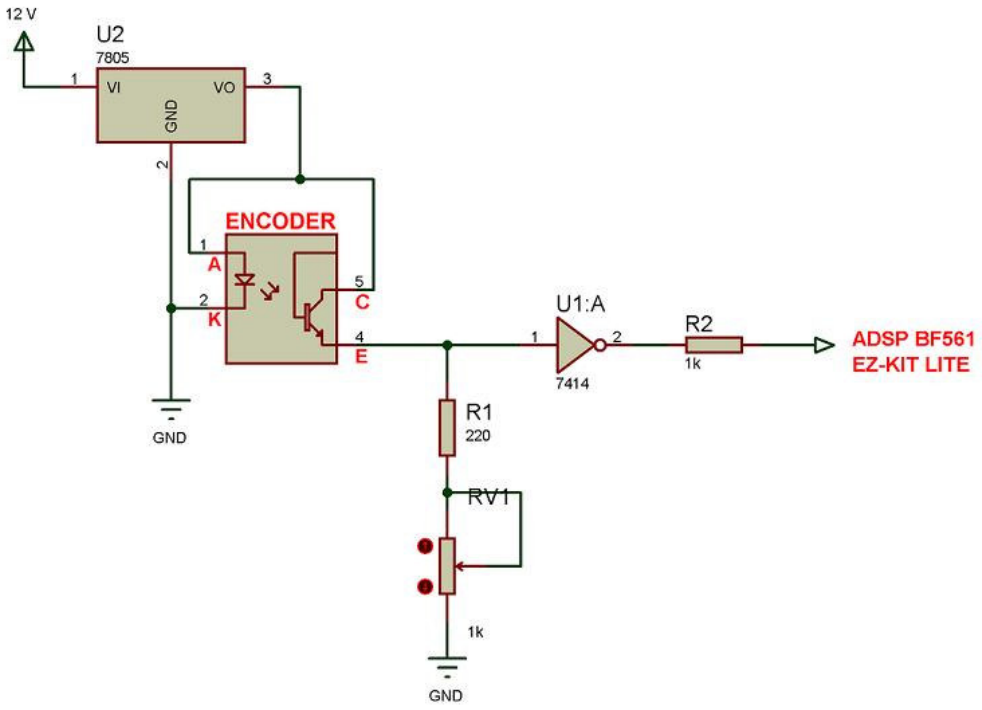
Kodlayıcılar genellikle motorların hız veya pozisyon bilgilerini almak için kullanılan elemanlardır. Kapasitif, endüktüf, optik ve potansiyometre gibi birçok çeşitleri vardır. Kodlayıcıların bir turda verdikleri pals sayısından faydalanılarak motorun hızı yada pozisyonu sürücü devreler ile kontrol edilebilir.



Şekil 5.8: Optik kodlayıcının yapısı

Projede kullanılan motorların arkasında optik kodlayıcı bulunmaktadır. Optik kodlayıcının çalışma mantığı, üzerinde belirli sayıda delik bulunan bir disk ve bu diskin iki tarafında karşılıklı olarak yer alan ışık kaynağı ile optik algılayıcıya dayanır (Şekil 5.8). Disk motora bağlanır, motorla beraber disk döndükçe ışık kaynağı ile optik algılayıcının arasından geçen diskin üzerindeki delikler optik algılayıcının pals oluşturmasına sebep olmaktadır. Bu palslerin frekansından faydalanılarak motorun hızı, ya da sayısından faydalanılarak motorun tur sayısı hesaplanabilmektedir. Projede kullanılan motorlarda bulunan optik kodlayıcının yapısı bir kızıl ötesi (infrared) led (ışık kaynağı) ve foto transistörden (optik algılayıcı) oluşmaktadır.

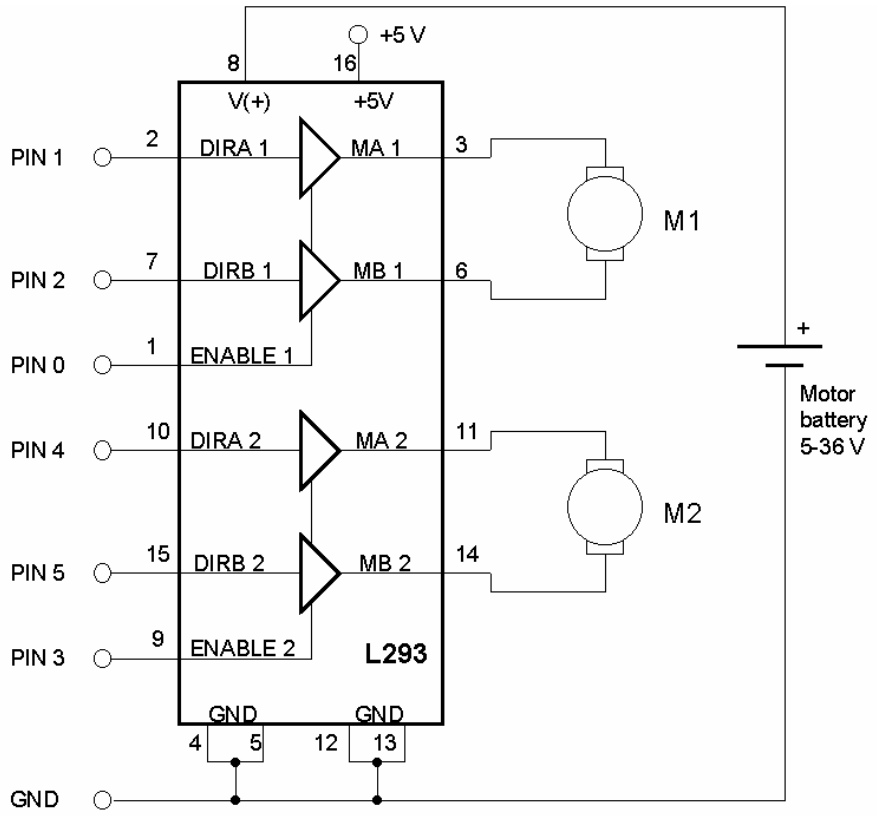
Bu projede kullanılan motorların arkasında bulunan optik kodlayıcı sinüs sinyali vermekte ve sinyalin minimum ve maksimum değerleri ADSP-BF561 kartının lojik 0 ve lojik 1 değer aralığı ile uyumluluk göstermemektedir. İşlemcinin palsleri algılamasını sağlamak için sinüs sinyali kare dalgaya dönüştürülüp minimum ve maksimum değerleri de kart ile uyumlu hale getirilmiştir. Bunun için tasarlanan devre Şekil 5.9'da görülmektedir.



Şekil 5.9: Optik kodlayıcı için tasarlanan sinüs – kare dalga dönüştürücü devresi

5.5. Motor Sürücü Devreleri

Motorların işlemciden gelen sinyallere göre istenilen yöne hareket ettirilmesi veya durdurulması için işlemciden gelen sinyalleri algılayıp bu sinyallere göre motorun uçlarına elektrik akımı uygulayan bir sürücü devresi kullanılmıştır. Sürücü devresinde çok kullanılan bir entegre olan L293D [11] entegresi tercih edilmiştir. L293D içerisinde 2 motoru birlikte sürebilecek bir yapıya sahiptir. Aynı zamanda bu entegrede motordan geri dönebilecek ters EMK' ya karşı korumak amacı ile diyodlar motora giden uçlara entegre içerisinde eklenmiştir.



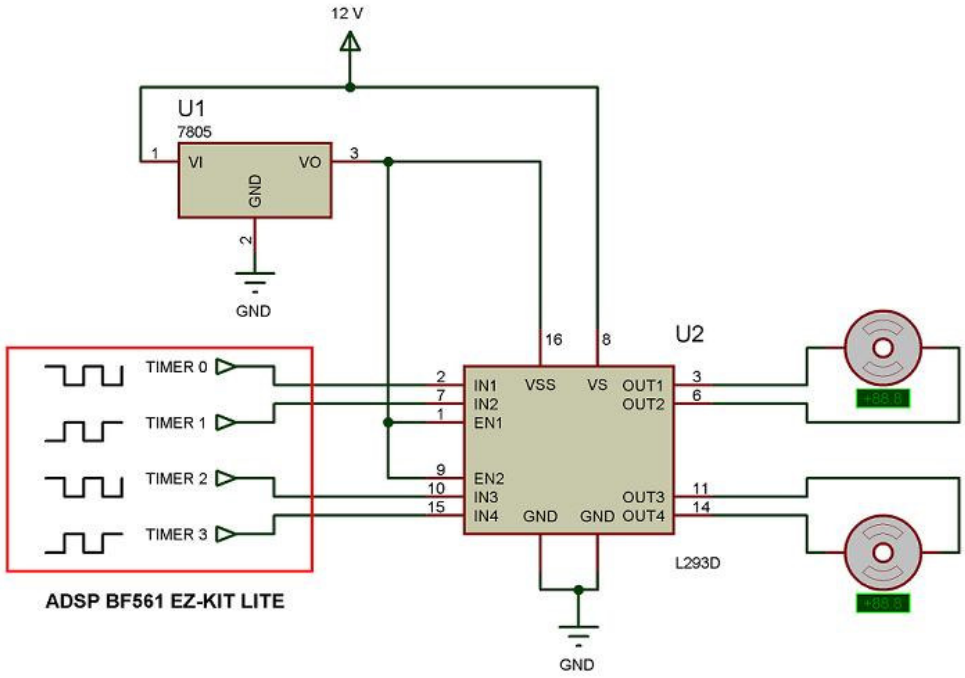
Şekil 5.10: L293D entegresinin bağlantı şeması

Şekil 5.10'da L293D entegresinin bağlantı şeması görülmektedir. Şemadan da görüldüğü gibi 8 nolu bacak motorların çalışması için gerekli kaynağın bağlandığı bacaktır. Bu bacağa 5-36 V arası gerilim verilebilmektedir. Entegre bu gerilimi motor bacaklarına anahtarlama yaparak motorları kontrol eder. 16 nolu bacak ise entegrenin kendi beslemesidir. 3 ve 6 nolu bacaklara motorlardan birisi, 11 ve 14 nolu bacaklara ise diğeri bağlanmaktadır. Motorların kontrolü ise, 1, 2, 7, 9, 10 ve 15 nolu bacaklardan yapılmaktadır.

L293D entegresinin kontrol yapısı Tablo 5.1'de verilmektedir. Tabloda da görüldüğü gibi gerekli pinlere PWM gönderilerek motorların dönüş yönlerinin ayarlanması veya durdurulması mümkün olmaktadır. Bu projede motorların sürülmesinde PWM frekansı 1 kHz olarak ayarlanmıştır.

Tablo 5.1 Motoru farklı yönlere döndürmek için L293D nin data portuna gönderilen Sinyaller

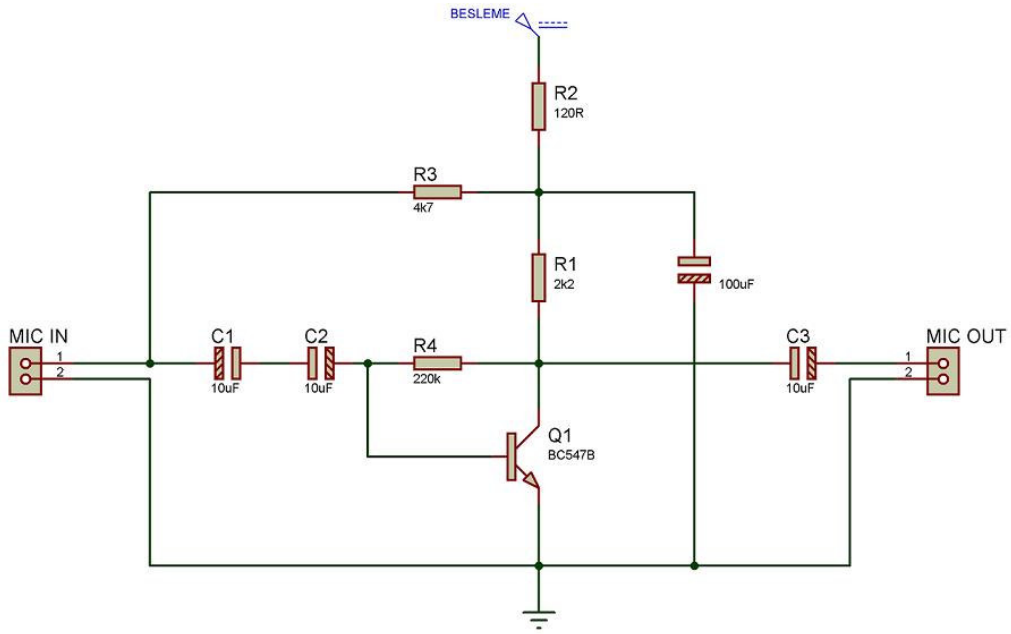
ENABLE	DIRA	DIRB	Fonksiyon
1	1	0	Sağa Dön
1	0	1	Sola Dön
1	1	1	Hızlı Frenleme
1	0	0	Hızlı Frenleme
0	Herhangi biri	Herhangi biri	Yavaş Frenleme



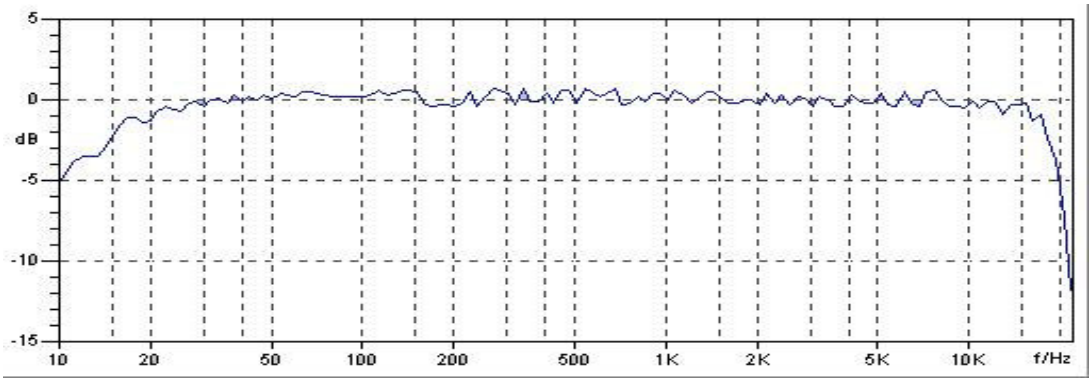
Şekil 5.11: Motor sürücü devre şeması

5.6. Mikrofon Ön yükselteç Devresi

Projede elektret mikrofonlar kullanılmıştır. Elektret mikrofonların çıkışları çok düşük (mV seviyesinde) olduğundan, mikrofonlardan gelen sinyaller ADSP-BF561 kartına gönderilmeden önce, ön yükselteç kullanılarak belirli bir seviyede yükseltilmektedir. Şekil 5.12'de görülen transistörlü devre ile mikrofon çıkışındaki işaret seviyesi istenilen düzeye getirilmiştir. Devrenin çıkışındaki frekans karakteristiği Şekil 5.13'te görülmektedir.

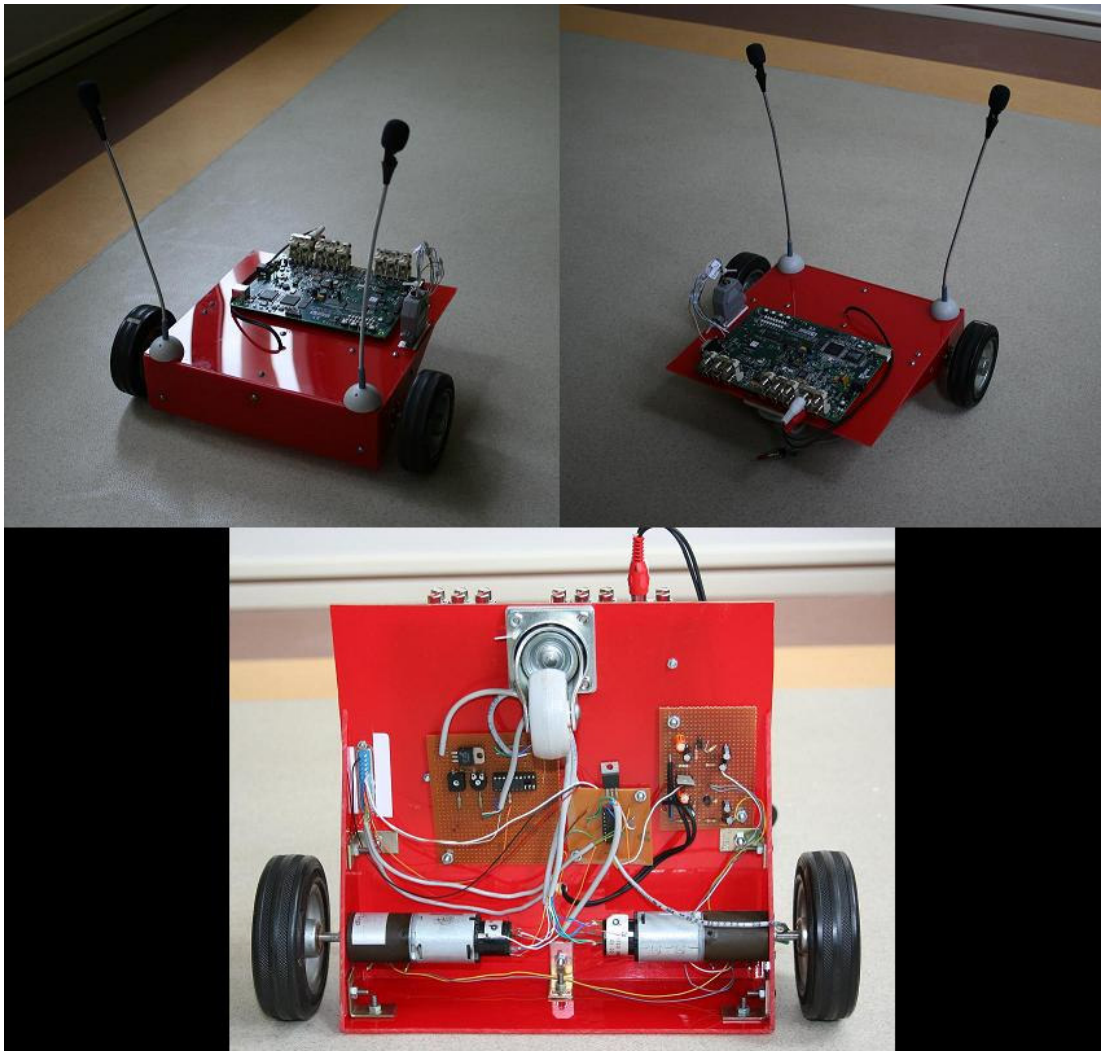


Şekil 5.12: Ön yükselteç devre şeması



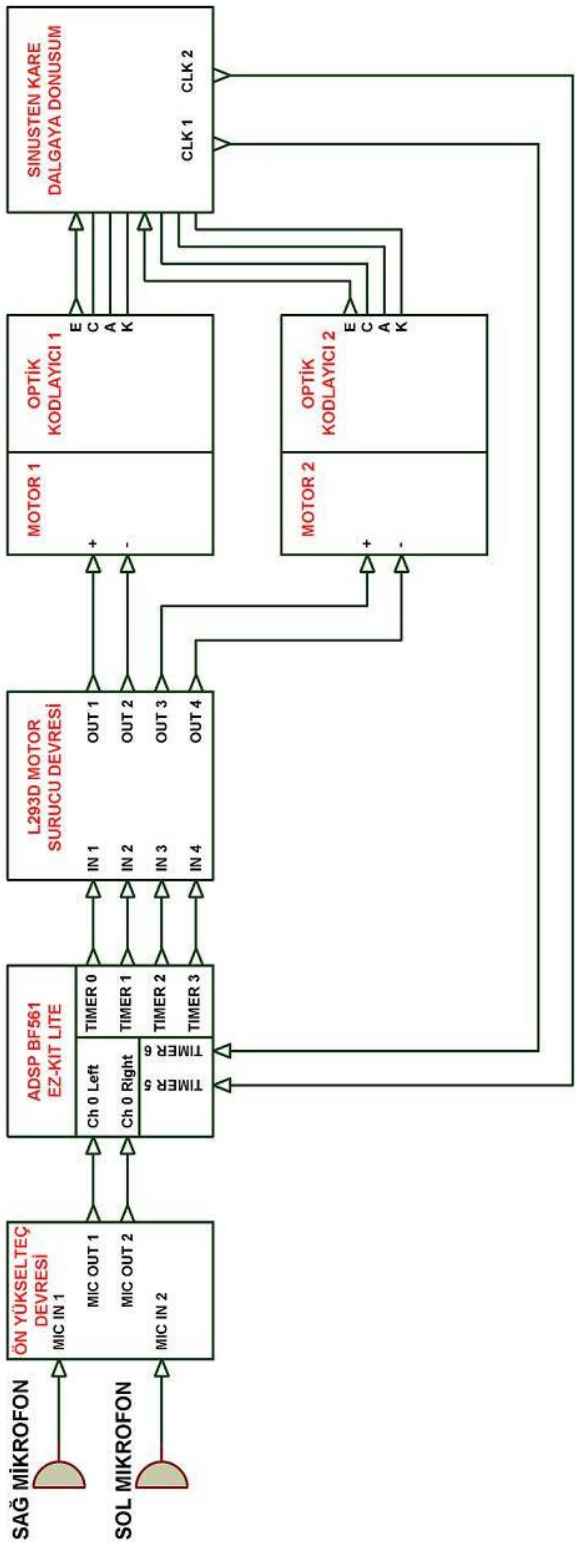
Şekil 5.13: Ön yükselteç devresinin frekans karakteristiği

5.7. Sistem Şeması



Şekil 5.14: Sistemin ön, arka ve alttan görünümleri

Şekil 5.14' te geliştirilen mobil robotun resmi görülmektedir. L293D motor sürücü devresi, optik kodlayıcı sinüs – kare dalga dönüştürücü devresi, ön yükselteç devreleri ve motorlar robotun kasasının alt tarafına, ASDP-BF561 kontrol kartı ve mikrofonlar ise kasanın üstüne yerleştirilmiştir. Şekil 5.15'te ise sistem şeması verilmektedir. Mobil robotu ADSP-BF561 kartı üzerinden programlamak için geliştirilen yazılımlar EKLER kısmında sunulmuştur.



Şekil 5.15: Sistem şeması

6.SONUÇLAR VE ÖNERİLER

Bu tez çalışmasında, bir mobil robot tasarımı ve bu robotun mikrofonlar ve kamera aracılığı ile dış dünya ile veri alışverişinde bulunması amaçlanmıştır. Bilindiği gibi görme ve işitme insanın çevreyi algılamasında büyük rol oynamaktadır. Bu projede de insan görme ve işitme mantığına benzer şekilde robot üzerinde uygulama yapılmıştır.

Robot donanımı tasarlanmış, kontrol sistemi oluşturulmuş, mikrofonlar aracılığı ile ses kaynağının yönü tespit edilmiş ve robot üzerinde bulunan optik kodlayıcılı motorlar vasıtası ile robot ses kaynağına doğru yönlendirilmiştir.

Bu uygulamada kullanılan mikrofon sayısı ikiden üçe çıkartılarak sesin robotun önünden mi yoksa arkasından mı geldiği tespit edilebilir. Benzer şekilde kamera sayısı da ikiye çıkartılarak stereo vision olarak adlandırılan çift kanallı görüş sistemi ile nesnelerin uzaklıkları tespit edilebilir. Bu proje, pratikte üzerinde bazı modifikasyonlar yapılarak kurtarma robotu yada insansız araç projesine dönüştürülebilir.

KAYNAKLAR

1. http://tr.wikipedia.org/wiki/Robot **(Ziyaret Tarihi :15 Ocak 2008)**

2. http://dvice.com/archives/2006/06/bloodhound_robot_sniffs_out_ea.php **(Ziyaret Tarihi: 24 Ocak 2008)**

3. http://robot.metu.edu.tr/dosya/robot_nedir.pdf **(Ziyaret Tarihi: 24 Ocak 2008)**

4. http://robotics.nasa.gov/index.php **(Ziyaret Tarihi: 24 Ocak 2008)**

5. http://www.lynxmotion.com/Category.aspx?CategoryID=2 **(Ziyaret tarihi: 24 Ocak 2008)**

6. K. S. Fu, R. C. Gonzalez, C. S. G. Lee, "ROBOTICS : Control, Sensing, Vision, and Intelligence." ***McGraw-Hill Book Company***. 1987.

7. http://www.societyofrobots.com/sensors.shtml (**Ziyaret tarihi: 7 Şubat 2008)**

8. Murray J. C., Erwin H.and Wermter S. "Robotic Sound-Source Localization and Tracking Using Interaural Time Difference and Cross-Correlation" ***EPSRC & OCF Plc***

9. S. Vassiliadis, E.A. Hakkennes, J.S.S.M. Wong, G.G. Pechanek, "The Sum-Absolute-Difference Motion Estimation Accelerator," euromicro, p. 20559, 24 th. ***EUROMICRO Conference Volume 2 (EUROMICRO'98)***, (1998)

10. Sun K. "Adaptive motion estimation based on statistical sum of absolute difference" ***Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on 601-604 vol.3*** (1998)

11. http://www.st.com/stonline/books/pdf/docs/1330.pdf **(Ziyaret Tarihi: 19 Mart 2008)**

EKLER

ISR.c

```
#include "main.h"

//--------------------------------------------------------------------------//
// Function:    Sport0_RX_ISR
//
// Description: This ISR is executed after a complete frame of input data
//has been received. The new samples are stored in
//iChannel0LeftIn, iChannel0RightIn, iChannel1LeftIn and
//iChannel1RightIn, respectively. The function Process_Data()
//is called after NUM_SAMPLES data points are acquired.
//--------------------------------------------------------------------------//
EX_INTERRUPT_HANDLER(Sport0_RX_ISR)
{
        // Confirm interrupt handling.
        *pDMA2_0_IRQ_STATUS = 0x0001;

        // Copy input data from dma input buffer into variables.
        iChannel0LeftIn = iRxBuffer1[INTERNAL_ADC_L0];
        iChannel0RightIn = iRxBuffer1[INTERNAL_ADC_R0];
        iChannel1LeftIn = iRxBuffer1[INTERNAL_ADC_L1];
        iChannel1RightIn = iRxBuffer1[INTERNAL_ADC_R1];

        if (program_mode == SAMPLE_MODE)
        {
                if (data_count < NUM_SAMPLES)  // Continue sampling.
                {
                        x_ch0[data_count] = iChannel0LeftIn;
                        x_ch0[data_count] /= ADC_SCALE_FACTOR;
                        x_ch2[data_count] = iChannel0RightIn;
                        x_ch2[data_count] /= ADC_SCALE_FACTOR;
                        data_count++;
                }
                else                            // Process data.
                {
                        program_mode = PROCESS_MODE;
                        Process_Data();
                        data_count = 0;         // Reset counter.
                        program_mode = SAMPLE_MODE;
                }
        }
}
```

main.h

```
#ifndef  __MAIN_DEFINED
#define __MAIN_DEFINED


//-------------------------------------------------------------------------//
// Header files                                                            //
//-------------------------------------------------------------------------//
#include "system.h"
#include <sys\exception.h>
#include <cdefBF561.h>
#include <ccblkfn.h>
#include <sysreg.h>


//-------------------------------------------------------------------------//
// Single vs dual core operation                                           //
//-------------------------------------------------------------------------//
#define RUN_ON_SINGLE_CORE   // Comment out if dual core operation is
desired.


//-------------------------------------------------------------------------//
// Symbolic constants                                                      //
//-------------------------------------------------------------------------//
// AD1836 reset PF15
#define AD1836_RESET_bit 15

// Names for codec registers, used for iCodec1836TxRegs[]
#define DAC_CONTROL_1          0x0000
#define DAC_CONTROL_2          0x1000
#define DAC_VOLUME_0           0x2000
#define DAC_VOLUME_1           0x3000
#define DAC_VOLUME_2           0x4000
#define DAC_VOLUME_3           0x5000
#define DAC_VOLUME_4           0x6000
#define DAC_VOLUME_5           0x7000
#define ADC_0_PEAK_LEVEL       0x8000
#define ADC_1_PEAK_LEVEL       0x9000
#define ADC_2_PEAK_LEVEL       0xA000
#define ADC_3_PEAK_LEVEL       0xB000
#define ADC_CONTROL_1          0xC000
#define ADC_CONTROL_2          0xD000
#define ADC_CONTROL_3          0xE000
```

```
// Names for slots in ad1836 audio frame
#define INTERNAL_ADC_L0                 0
#define INTERNAL_ADC_L1                 1
#define INTERNAL_ADC_R0                 4
#define INTERNAL_ADC_R1                 5
#define INTERNAL_DAC_L0                 0
#define INTERNAL_DAC_L1                 1
#define INTERNAL_DAC_L2                 2
#define INTERNAL_DAC_R0                 4
#define INTERNAL_DAC_R1                 5
#define INTERNAL_DAC_R2                 6

// Size of array iCodec1836TxRegs and iCodec1836RxRegs
#define CODEC_1836_REGS_LENGTH    11

// SPI transfer mode
#define TIMOD_DMA_TX 0x0003

// SPORT0 word length
#define SLEN_32      0x001f

// DMA flow mode
#define FLOW_1       0x1000

#define PI 3.14159265

// The program will be in one of the two modes.
#define SAMPLE_MODE 0
#define PROCESS_MODE 1

// Samples acquired from the ADC should be scaled by ADC_SCALE_FACTOR
//  to convert the sample values to volts.
#define ADC_SCALE_FACTOR 1.3e9

#define SAMPLE_FREQ 48000                   // Sampling freq. (default) in Hz.
#define SAMPLE_PERIOD 1./SAMPLE_FREQ    // Sampling period in seconds
#define T_WINDOW 0.25                       // Window length in seconds
#define NUM_SAMPLES (long) (SAMPLE_FREQ*T_WINDOW)// # of samples
#define DEC_FACTOR 1                        // Decimating factor
#define NUM_POINTS NUM_SAMPLES/DEC_FACTOR
                                    // # of points in a window after decimation

// FFT size to be used in FFT based cross-correlation computation.
// Must be set to the closest power of two >= 2*NUM_POINTS-1.
#define N_FFT 8192
```

```
#define CLOCK_FREQ 120e6        // in Hz (do not change)
#define PWM_FREQ 1000           // in Hz (change as desired)

#define MIC_DIST 0.3        // Distance between two microphones (in meters)
#define SOUND_SPEED 384         // The speed of sound (in meters per second)

// Cross-correlation between two audio channels will be computed over the range of
//  -MAXLAG:MAXLAG. The difference in the distances between the source and
the
//   microphones is at its maximum when the source is at either 0 or 180 degrees
//   away from the microphones. In such case, the sound will reach the furthest
//   microphone MAXLAG samples after it reaches the closest one.
#define MAXLAG (int)
(1+(MIC_DIST/SOUND_SPEED)*(SAMPLE_FREQ/DEC_FACTOR))

#define NUM_PULSES_PER_TOUR 2048          // # of pulses encoder gives in a
complete tour.

#define WHEEL_CIRCUMFERENCE 31.0          // Circumference of front wheels
in centimeters.
#define AXLE_DISTANCE 31.75                       // Distance between two
wheels in centimeters.
#define ROTATE_CIRCLE_CIRCUMFERENCE 2*PI*AXLE_DISTANCE      //
Only when one motor is running.

//------------------------------------------------------------------------//
// Global variables
//------------------------------------------------------------------------//
extern int iChannel0LeftIn;
extern int iChannel0RightIn;
extern int iChannel0LeftOut;
extern int iChannel0RightOut;
extern int iChannel1LeftIn;
extern int iChannel1RightIn;
extern int iChannel1LeftOut;
extern int iChannel1RightOut;
extern volatile short sCodec1836TxRegs[];
extern volatile int iRxBuffer1[];
extern volatile int iTxBuffer1[];

extern int program_mode;     // SAMPLE_MODE or PROCESS_MODE.
extern int data_count;
extern float x_ch0[NUM_SAMPLES];
//extern float x_ch1[NUM_SAMPLES];
extern float x_ch2[NUM_SAMPLES];
//extern float x_ch3[NUM_SAMPLES];
extern int period_count_motorRight, period_count_motorLeft;
extern int tmp_cnt;
```

```
//-------------------------------------------------------------------------//
// Prototypes
//-------------------------------------------------------------------------//
// in file Initialisation.c
void Init_EBIU(void);
void Init_Flash(void);
void Init1836(void);
void Init_Sport0(void);
void Init_DMA(void);
void Init_Sport_Interrupts(void);
void Enable_DMA_Sport(void);
void Enable_DMA_Sport0(void);
void InitSDRAM(void);
void Set_PLL(short CoreCLOCK_multiplier, short SCLK_divider);

// in file Process_data.c
void Process_Data(void);


//-------------------------------------------------------------------------//
// Prototype interrupts
//-------------------------------------------------------------------------//
EX_EXCEPTION_HANDLER(ex_handler_coreA);
// in file ISRs.c
EX_INTERRUPT_HANDLER(Sport0_RX_ISR);

EX_INTERRUPT_HANDLER(Timer4_ISR);
EX_INTERRUPT_HANDLER(Timer5_ISR);



#endif
```

main_core_A.c

```
#include "cdefBF561.h"
#include "ccblkfn.h"
#include "stdio.h"
#include "sys\exception.h"
#include "main.h"

//-------------------------------------------------------------------------//
// Variables                                                               //
//                                                                         //
// Description: The variables iChannelxLeftIn and iChannelxRightIn contain //
//              the data coming from the codec AD1836.  The                //
//              playback data are written into the variables               //
//              iChannelxLeftOut and iChannelxRightOut respectively, which //
//              are then sent back to the codec in the SPORT0 ISR.         //
//              The values in the array iCodec1836TxRegs can be modified to //
//              set up the codec in different configurations according to  //
//              the AD1836 data sheet.                                     //
//-------------------------------------------------------------------------//

// Left input data from AD1836
int iChannel0LeftIn, iChannel1LeftIn;
// Right input data from AD1836
int iChannel0RightIn, iChannel1RightIn;
// Left ouput data for AD1836
int iChannel0LeftOut, iChannel1LeftOut;
// Right ouput data for AD1836
int iChannel0RightOut, iChannel1RightOut;

// Array for registers to configure the AD1836
// Names are defined in "main.h".
volatile short sCodec1836TxRegs[CODEC_1836_REGS_LENGTH] =
{
                                        DAC_CONTROL_1  | 0x000,
                                        DAC_CONTROL_2  | 0x000,
                                        DAC_VOLUME_0   | 0x3ff,
                                        DAC_VOLUME_1   | 0x3ff,
                                        DAC_VOLUME_2   | 0x3ff,
                                        DAC_VOLUME_3   | 0x3ff,
```

```
                                        DAC_VOLUME_4    | 0x3ff,
                                        DAC_VOLUME_5    | 0x3ff,
                                        ADC_CONTROL_1   | 0x000,
//                                      ADC_CONTROL_1   | 0x024,        // Adjusted
for 12dB gain on both channels.
                                        ADC_CONTROL_2   | 0x180,
                                        ADC_CONTROL_3   | 0x000

};

// SPORT0 DMA transmit buffer
volatile int iTxBuffer1[8];
// SPORT0 DMA receive buffer
volatile int iRxBuffer1[8];

int program_mode=SAMPLE_MODE;       // Start with sample mode.
int data_count=0;                   // Reset data count.

// Data arrays.
float x_ch0[NUM_SAMPLES];
float x_ch2[NUM_SAMPLES];

int period_count_motorRight=0, period_count_motorLeft=0;
int tmp_cnt=0;

//-------------------------------------------------------------------------//
// Function:    main                                                       //
//                                                                         //
// Description: After calling a few initalization routines, main() just    //
//              waits in a loop forever.  The code to process the incoming //
//              data is placed in the function Process_Data() in the file  //
//              "Process_Data.c".                                          //
//-------------------------------------------------------------------------//

void main(void)
{

    // Unblock Core B if dual core operation is desired.
    #ifndef RUN_ON_SINGLE_CORE
        *pSICA_SYSCR &= 0xFFDF;     // Clear bit 5 to unlock.
    #endif

    // Sets Core clock to 600MHz (= 18 x 33MHz) and SCLK to 120MHz
(=600Mhz/5)
```

```
        Set_PLL( (short)(CORECLK/CLKIN), (short)(CORECLK/SYSCLK));

        // Initialize SDRAM.
        InitSDRAM();

        // Initialize Timers 0-3 for PWM Output mode.
        // Set to count periods.
        // Refer to hardware manual 15-13
        *pTIMER0_CONFIG= 0xD;
        asm("ssync;");
        *pTIMER1_CONFIG= 0xD;
        asm("ssync;");
        *pTIMER2_CONFIG= 0xD;
        asm("ssync;");
        *pTIMER3_CONFIG= 0xD;
        asm("ssync;");

        // Initialize Timers 5, 6 and 7 for External Clock mode.
        *pTIMER5_CONFIG = 0x07;
        asm("ssync;");
        *pTIMER6_CONFIG = 0x07;
        asm("ssync;");
        *pTIMER7_CONFIG = 0x07;
        asm("ssync;");

        // Set Timers0-3 periods for PWM.
        *pTIMER0_PERIOD = CLOCK_FREQ/PWM_FREQ;
        *pTIMER1_PERIOD = CLOCK_FREQ/PWM_FREQ;
        *pTIMER2_PERIOD = CLOCK_FREQ/PWM_FREQ;
        *pTIMER3_PERIOD = CLOCK_FREQ/PWM_FREQ;

        Init1836();
        Init_Sport0();
        Init_DMA();
        Init_Sport_Interrupts();
        Enable_DMA_Sport0();

        while(1);

}
```

Process_data.c

```
//-------------------------------------------------------------------------//
// Function:     Process_Data()
// Description: This function is called from inside the SPORT0 ISR every
//time NUM_SAMPLES data points (from each channel) have been
//acquired. The samples are stored in arrays x_ch0 (Channel 0
// left in), x_ch1 (Channel 1 left in), x_ch2 (Channel 0 right
//in) and x_ch3 (Channel 1 right in) arrays.
//-------------------------------------------------------------------------//
#include "main.h"
#include "math.h"
#include "vector.h"
#include "complex.h"
#include "filter.h"
#include "stats.h"
#include "stdio.h"
#include "fract2float_conv.h"
#include "cdefBF561.h"
#include "ccblkfn.h"
#include "sys\exception.h"

#define RMS_SILENCE_THR 0.05
#define CORR_MAX_OVER_MEAN_THR 1.1

// Function prototypes.
void decimate(float x[], float y[], int dec_factor);
void reverse(float x[], int N_points);
int max_index(float x[], int N_points);
void xcorr(float x[], float y[], float correlation[]);

// Motor control functions.
void motorRight_forward(int duty_cycle);
void motorRight_backward(int duty_cycle);
void motorRight_forward_Ntours_And_Stop(int duty_cycle, float N_tours);
void motorRight_backward_Ntours_And_Stop(int duty_cycle, float N_tours);
void motorRight_ForwardSpeedUp(int new_duty_cycle, float N_tours);
void motorRight_BackwardSpeedUp(int new_duty_cycle, float N_tours);
void motorRight_stop(void);
void motorLeft_forward(int duty_cycle);
void motorLeft_backward(int duty_cycle);
void motorLeft_forward_Ntours_And_Stop(int duty_cycle, float N_tours);
void motorLeft_backward_Ntours_And_Stop(int duty_cycle, float N_tours);
void motorLeft_stop(void);
void robot_forward(int duty_cycle);
void robot_backward(int duty_cycle);
void robot_stop(void);
float angle_to_tours(float angle);
```

```
float corr_sig[2*MAXLAG-1];                 // Array to hold corr. signal.
float source_angle;

void Process_Data(void)
{
   int i, max_ind, sigma;
   float d;
   float theta;
   float rms_ch0, rms_ch2;
   float max_corr, mean_corr;

   // Decimate the data.
   if (DEC_FACTOR > 1) {
         decimate(x_ch0, x_ch0, DEC_FACTOR);
         decimate(x_ch2, x_ch2, DEC_FACTOR);
   }

         // Do nothing and return from the function if there is no sound.
         rms_ch0 = rmsf(x_ch0, NUM_POINTS);
   rms_ch2 = rmsf(x_ch2, NUM_POINTS);
   if ((rms_ch0+rms_ch2)/2<RMS_SILENCE_THR) return;

   // Cross-correlation between two channels.
   // Direct method is used which is very slow compared to FFT based method.
   crosscorrf(x_ch0, x_ch2, NUM_POINTS, MAXLAG, &corr_sig[0]);
   reverse(corr_sig, MAXLAG);
   crosscorrf(x_ch2, x_ch0, NUM_POINTS, MAXLAG, &corr_sig[MAXLAG-1]);

   // FFT based cross-correlation method.
   // Has problems with precision since the built-in FFT routine is fixed point.
   // Can be used when a floating point FFT routine is available.
   //xcorr(x_ch0, x_ch2, corr_sig);

   // Compute the angle of the sound source measured from the mid-point of the two
mics.
   max_ind = vecmaxlocf(corr_sig, 2*MAXLAG-1);
   max_corr = corr_sig[max_ind];
   mean_corr = ((2*MAXLAG-1)*meanf(corr_sig, 2*MAXLAG-1)-
max_corr)/(2*MAXLAG-2);
   sigma = max_ind-MAXLAG+1;
   d = sigma*SOUND_SPEED*SAMPLE_PERIOD*DEC_FACTOR;
   if (d<-MIC_DIST)
         d = -MIC_DIST;
   else if (d>MIC_DIST)
         d = MIC_DIST;
   source_angle = asinf(d/MIC_DIST) * 180 / PI;
```

```
    // Do nothing and return from the function if the ratio of the maximum and the
mean
    //  values of the correlation signal is less than a predefined threshold.
    if (max_corr/mean_corr < CORR_MAX_OVER_MEAN_THR) return;

    // Do not move the robot if the RMS values of the sound signals measured from
    //  the two microphones do not agree with the sign of the computed source angle.
    //if ((source_angle > 0 && rms_ch0 > rms_ch2) || (source_angle < 0 && rms_ch0
< rms_ch2)) return;

    // Move the robot to the desired angle.
    if (source_angle < 0) {
          //motorRight_backward(40);
       //motorLeft_forward_Ntours_And_Stop(40, angle_to_tours(-source_angle)/2);
       motorLeft_forward_Ntours_And_Stop(40, angle_to_tours(-source_angle));
          robot_stop(); }
    else if (source_angle > 0) {
       //motorLeft_backward(40);

          //motorRight_forward_Ntours_And_Stop(40,angle_to_tours(source_angle)/2)
;
          motorRight_forward_Ntours_And_Stop(40, angle_to_tours(source_angle));
          robot_stop(); }


    // For test purposes only.
    /*tmp_cnt++;
          if (tmp_cnt == 1) {
                robot_forward(40);
       motorRight_forward_Ntours_And_Stop(40, 1);
                robot_stop(); }

          tmp_cnt++;
    if (tmp_cnt == 1) {
       theta = 45;
                motorLeft_forward_Ntours_And_Stop(40, angle_to_tours(theta));
                robot_stop(); }
          else if (tmp_cnt == 3) {
                theta = 45;
                motorLeft_backward_Ntours_And_Stop(40, angle_to_tours(theta));
                robot_stop();
             }
          */
}
```

```
//-----------------------------------------------------------------//
// Procedure:  void decimate(float x[], float y[], int dec_factor)
// Arguments:  x: input array
// y: output array
// dec_factor: decimating factor
// Returns:    None
// Desc:       Decimates the data by dec_factor.
// In-place calculation is allowed (x can be equal to y)
//-----------------------------------------------------------------//
void decimate(float x[], float y[], int dec_factor)
{
        int i;    // Counter.

        for (i=0; i<NUM_SAMPLES; i++)
           y[i] = x[i*dec_factor];

        return;
}

//-----------------------------------------------------------------//
// Procedure:  void reverse(float x[], int N_points)
// Arguments:  x: input array
// N_points: # of points
// Returns:    None
// Desc:       Reverses the elements of an array.
//-----------------------------------------------------------------//
void reverse(float x[], int N_points)
{
        int i;    // Counter.
        float tmp;

        for (i=0; i<(int)((N_points-1)/2); i++) {
           tmp = x[i];
           x[i] = x[N_points-1-i];
           x[N_points-1-i] = tmp;
        }

        return;
}
```

```
//---------------------------------------------------------------//
// Procedure:  int max_index(float x[], int N_points)
// Arguments:  x: input array
// N_points: # of points
// Returns:    index of the maximum element.
// Desc:       Locates the index of the maximum element in an array
//---------------------------------------------------------------//
int max_index(float x[], int N_points)
{
        int i,max_index;
        float max_element;

        max_element = x[1];
        max_index = 1;
        for (i=2; i<N_points; i++)
           if (x[i] > max_element) {
              max_element = x[i];
              max_index = i; }
        return max_index;
}

//-------------------------------------------------------------------------//
// Procedure:  void xcorr(float x[], float y[], float correlation[])
// Arguments:  x: input array for the first signal
// y: input array for the second signal
// correlation: cross-correlation signal
// Returns:    None
// Desc:       Computes the cross-correlation of two signals. Built-in crosscorr       //
// function uses direct computation. This implementation uses FFT based //
// approach and thus is dramatically fast in comparison to built-in one.//
//-------------------------------------------------------------------------//
void xcorr(float x[], float y[], float correlation[])
{
   int i;                                  // Counter.
   int M=2*NUM_POINTS-1;          // # of elements in the cross-correlation signal.

        // x and y will be scaled before converting them to fract16.
        float x_max, y_max, scale_factor;

        // fract16 versions of x and y will be stored in these arrays.
        // These arrays are of length N_FFT. N_FFT is defined in main.h and
        //  is the closest power of two >= 2*NUM_POINTS-1.
        fract16 frac_x[N_FFT]={0x0}, frac_y[N_FFT]={0x0};

        // To be used for real FFT computation.
        complex_fract16 temp[N_FFT];
        complex_fract16 fft_x[N_FFT];
        complex_fract16 fft_y[N_FFT];
        complex_fract16 twiddle_table[N_FFT/2];
```

```
        // Scale x and y and convert to fract16.
        // Scaling to [-1,1) (=range for fractional #'s) is required to avoid saturation.
        x_max = vecmaxf(x, NUM_POINTS);
        y_max = vecmaxf(y, NUM_POINTS);
        scale_factor = fmaxf(x_max, y_max);
        for (i=0; i<NUM_POINTS; i++) {
           frac_x[i] = float_to_fr16(x[i]/scale_factor);
           frac_y[i] = float_to_fr16(y[i]/scale_factor);
        }

        // FFT of X and Y.
        twidfftrad2_fr16(twiddle_table, N_FFT);
        rfft_fr16(frac_x, temp, fft_x, twiddle_table, 1, N_FFT, 0, 0);
        rfft_fr16(frac_y, temp, fft_y, twiddle_table, 1, N_FFT, 0, 0);

        // Compute fff_x .* conj(fft_y)
        // The result is written to fft_x to save memory space.
        for (i=0; i<N_FFT; i++)
           fft_x[i] = cmlt_fr16(fft_x[i], conj_fr16(fft_y[i]));

        // Compute the inverse FFT of the result.
        // The result is written to fft_y to save memory space.
        ifft_fr16(fft_x, temp, fft_y, twiddle_table, 1, N_FFT, 0, 0);

        // Cross-correlation.
        // Reverse static scaling applied by rfft_fr16() function.
        // Also, reverse the scaling applied before fract16 conversion.
        for(i=0; i<NUM_POINTS-1; i++ ) {
                correlation[i] = fr16_to_float(fft_y[N_FFT-NUM_POINTS+i+1].re);
                correlation[i] *= N_FFT * scale_factor;
        }
        for(i=NUM_POINTS-1; i<M; i++ ) {
                correlation[i] = fr16_to_float(fft_y[i-NUM_POINTS+1].re);
                correlation[i] *= N_FFT * scale_factor;
        }

   return;
}


void motorRight_forward(int duty_cycle)
{
        // Set timer width.
        *pTIMER0_WIDTH = *pTIMER0_PERIOD*(duty_cycle)/100.;

        // Enable Timer 0 and disable Timer 1.
        *pTMRS8_ENABLE = 0x0001;
        *pTMRS8_DISABLE = 0x0002;
```

```
}

void motorRight_backward(int duty_cycle)
{
        // Set timer width.
        *pTIMER1_WIDTH = *pTIMER1_PERIOD*(duty_cycle)/100.;

        // Disable Timer 0 and enable Timer 1.
        *pTMRS8_DISABLE = 0x0001;
        *pTMRS8_ENABLE = 0x0002;
}

void motorRight_forward_Ntours_And_Stop(int duty_cycle, float N_tours)
{
   // Set timer width.
        *pTIMER0_WIDTH = *pTIMER0_PERIOD*(duty_cycle)/100.;

        // Enable Timer 0 and disable Timer 1.
        *pTMRS8_ENABLE = 0x0001;
        *pTMRS8_DISABLE = 0x0002;

        // Wait until the motor makes N_tours.
        *pTMRS8_ENABLE = 0x0020;            // Enable Timer 5.
        *pTIMER5_PERIOD = 0xFFFFFFF;        // Maximum period count.
        period_count_motorRight = 0;
        while (*pTIMER5_COUNTER <= N_tours*NUM_PULSES_PER_TOUR)
                period_count_motorRight = *pTIMER5_COUNTER;

        *pTMRS8_DISABLE = 0x0020;           // Disable Timer 5.

        // Stop the right motor.
        motorRight_stop;
}

void motorRight_backward_Ntours_And_Stop(int duty_cycle, float N_tours)
{
   // Set timer width.
        *pTIMER1_WIDTH = *pTIMER1_PERIOD*(duty_cycle)/100.;

        // Disable Timer 0 and enable Timer 1.
        *pTMRS8_DISABLE = 0x0001;
        *pTMRS8_ENABLE = 0x0002;

        // Wait until the motor makes N_tours.
        *pTMRS8_ENABLE = 0x0020;            // Enable Timer 5.
        *pTIMER5_PERIOD = 0xFFFFFFF;        // Maximum period count.
        period_count_motorRight = 0;
        while (*pTIMER5_COUNTER <= N_tours*NUM_PULSES_PER_TOUR)
                period_count_motorRight = *pTIMER5_COUNTER;
```

```
        *pTMRS8_DISABLE = 0x0020;               // Disable Timer 5.

        // Stop the right motor.
        motorRight_stop;
}

void motorRight_ForwardSpeedUp(int new_duty_cycle, float N_tours)
{
// Right motor makes N_tours in forward direction under
//  duty_cycle1 and returns to its original speed.

        long int timer_width=*pTIMER0_WIDTH;

        // Enable Timer 0 and disable Timer 1.
        *pTMRS8_ENABLE = 0x0001;
        *pTMRS8_DISABLE = 0x0002;

        // Set the timer width to its new value.
        *pTIMER0_WIDTH = *pTIMER0_PERIOD*new_duty_cycle/100.;

        // Wait until the motor makes N_tours.
        *pTMRS8_ENABLE = 0x0020;                // Enable Timer 5.
        *pTIMER5_PERIOD = 0xFFFFFFF;            // Maximum period count.
        period_count_motorRight = 0;
        while (*pTIMER5_COUNTER <= N_tours*NUM_PULSES_PER_TOUR)
                period_count_motorRight = *pTIMER5_COUNTER;

        *pTMRS8_DISABLE = 0x0020;               // Disable Timer 5.

        // Set the timer width to its original value.
        *pTIMER0_WIDTH = timer_width;
}

void motorRight_BackwardSpeedUp(int new_duty_cycle, float N_tours)
{
// Right motor makes N_tours in backward direction under
//  duty_cycle1 and returns to its original speed.

        long int timer_width=*pTIMER0_WIDTH;

        // Disable Timer 0 and enable Timer 1.
        *pTMRS8_DISABLE = 0x0001;
        *pTMRS8_ENABLE = 0x0002;

        // Set the timer width to its new value.
        *pTIMER1_WIDTH = *pTIMER1_PERIOD*new_duty_cycle/100.;

        // Wait until the motor makes N_tours.
```

```
        *pTMRS8_ENABLE = 0x0020;                // Enable Timer 5.
        *pTIMER5_PERIOD = 0xFFFFFFF;            // Maximum period count.
        period_count_motorRight = 0;
        while (*pTIMER5_COUNTER <= N_tours*NUM_PULSES_PER_TOUR)
                period_count_motorRight = *pTIMER5_COUNTER;

        *pTMRS8_DISABLE = 0x0020;               // Disable Timer 5.

        // Set the timer width to its original value.
        *pTIMER1_WIDTH = timer_width;
}

void motorRight_stop(void)
{
        // Disable Timers 0 and 1.
        *pTMRS8_DISABLE = 0x0003;
}

void motorLeft_forward(int duty_cycle)
{
        // Set timer width.
        *pTIMER2_WIDTH = *pTIMER2_PERIOD*(duty_cycle)/100.;

        // Enable Timer 2 and disable Timer 3.
        *pTMRS8_ENABLE = 0x0004;
        *pTMRS8_DISABLE = 0x0008;
}

void motorLeft_backward(int duty_cycle)
{
        // Set timer width.
        *pTIMER3_WIDTH = *pTIMER3_PERIOD*(duty_cycle)/100.;

        // Disable Timer 2 and enable Timer 3.
        *pTMRS8_DISABLE = 0x0004;
        *pTMRS8_ENABLE = 0x0008;
}

void motorLeft_forward_Ntours_And_Stop(int duty_cycle, float N_tours)
{
   // Set timer width.
        *pTIMER2_WIDTH = *pTIMER2_PERIOD*(duty_cycle)/100.;

        // Enable Timer 2 and disable Timer 3.
        *pTMRS8_ENABLE = 0x0004;
        *pTMRS8_DISABLE = 0x0008;

        // Wait until the motor makes N_tours.
        *pTMRS8_ENABLE = 0x0040;                // Enable Timer 6.
```

```
        *pTIMER6_PERIOD = 0xFFFFFFF;          // Maximum period count.
        period_count_motorLeft = 0;
        while (*pTIMER6_COUNTER <= N_tours*NUM_PULSES_PER_TOUR)
                period_count_motorLeft = *pTIMER6_COUNTER;

        *pTMRS8_DISABLE = 0x0040;             // Disable Timer 6.

        // Stop the left motor.
        motorLeft_stop;
}

void motorLeft_backward_Ntours_And_Stop(int duty_cycle, float N_tours)
{
   // Set timer width.
        *pTIMER3_WIDTH = *pTIMER3_PERIOD*(duty_cycle)/100.;

        // Disable Timer 2 and enable Timer 3.
        *pTMRS8_DISABLE = 0x0004;
        *pTMRS8_ENABLE = 0x0008;

        // Wait until the motor makes N_tours.
        *pTMRS8_ENABLE = 0x0040;              // Enable Timer 6.
        *pTIMER6_PERIOD = 0xFFFFFFF;          // Maximum period count.
        period_count_motorLeft = 0;
        while (*pTIMER6_COUNTER <= N_tours*NUM_PULSES_PER_TOUR)
                period_count_motorLeft = *pTIMER6_COUNTER;

        *pTMRS8_DISABLE = 0x0040;             // Disable Timer 6.

        // Stop the left motor.
        motorLeft_stop;
}

void motorLeft_stop(void)
{
        // Disable Timers 2 and 3.
        *pTMRS8_DISABLE = 0x000C;
}

void robot_forward(int duty_cycle)
{
        // Set timer width.
        *pTIMER0_WIDTH = *pTIMER0_PERIOD*(duty_cycle)/100.;
        *pTIMER2_WIDTH = *pTIMER2_PERIOD*(duty_cycle)/100.;

        // Enable Timers 0 and 2.
        // Disable Timers 1 and 3.
        *pTMRS8_ENABLE = 0x0005;
        *pTMRS8_DISABLE = 0x000A;
```

```
}

void robot_backward(int duty_cycle)
{
	// Set timer width.
	*pTIMER1_WIDTH = *pTIMER1_PERIOD*(duty_cycle)/100.;
	*pTIMER3_WIDTH = *pTIMER3_PERIOD*(duty_cycle)/100.;

	// Disable Timers 0 and 2.
	// Enable Timers 1 and 3.
	*pTMRS8_DISABLE = 0x0005;
	*pTMRS8_ENABLE = 0x000A;
}

void robot_stop(void)
{
	// Disable Timers 0,1,2 and 3.
	*pTMRS8_DISABLE = 0x000F;
}

float angle_to_tours(float angle)
{
// angle must be in degrees.
	return
(angle/360)*ROTATE_CIRCLE_CIRCUMFERENCE/WHEEL_CIRCUMFERENC
E;
}
```

Initialize.c

```
#include "main.h"

//-------------------------------------------------------------------------//
// Function:     Init1836()
// Description: This function sets up the SPI port to configure the AD1836.
// The content of the array sCodec1836TxRegs is sent to the
// codec.
//-------------------------------------------------------------------------//
void Init1836(void)
{
        int wait_reset;
        int wait_dma_finish;

        // reset codec
        // set PF15 as output
        *pFIO0_DIR |= (1 << AD1836_RESET_bit);
        ssync();
        // clear bit to enable AD1836
        *pFIO0_FLAG_S |= (1 << AD1836_RESET_bit);
        ssync();

        // wait to recover from reset
        for (wait_reset=0; wait_reset<0xf000; wait_reset++);

        // Enable PF4
        *pSPI_FLG = FLS4;
        // Set baud rate SCK = HCLK/(2*SPIBAUD)
        *pSPI_BAUD = 16;
        // configure spi port
        // SPI DMA write, 16-bit data, MSB first, SPI Master
        *pSPI_CTL = TIMOD_DMA_TX | SIZE | MSTR;

        // Set up DMA2 channel 4 to SPI transmit
        *pDMA2_4_PERIPHERAL_MAP = 0x4000;

        // Configure DMA2 channel4
        // 16-bit transfers
        *pDMA2_4_CONFIG = WDSIZE_16;
        // Start address of data buffer
        *pDMA2_4_START_ADDR = (void *)sCodec1836TxRegs;
        // DMA inner loop count
        *pDMA2_4_X_COUNT = CODEC_1836_REGS_LENGTH;
        // Inner loop address increment
        *pDMA2_4_X_MODIFY = 2;

        // enable DMAs
        *pDMA2_4_CONFIG = (*pDMA2_4_CONFIG | DMAEN);
```

```
	// enable spi
	*pSPI_CTL = (*pSPI_CTL | SPE);
	ssync();

	// wait until dma transfers for spi are finished
	for (wait_dma_finish=0; wait_dma_finish<0xaff; wait_dma_finish++);

	// disable spi
	*pSPI_CTL = 0x0000;
}


//-------------------------------------------------------------------------//
// Function:	Init_Sport0
// Description: Configure Sport0 for TDM mode, to transmit/receive data
// to/from the AD1836. Configure Sport for external clocks and
// frame syncs.
//-------------------------------------------------------------------------//
void Init_Sport0(void)
{
	// Sport0 receive configuration
	// External CLK, External Frame sync, MSB first
	// 32-bit data
	*pSPORT0_RCR1 = RFSR;
	*pSPORT0_RCR2 = SLEN_32;

	// Sport0 transmit configuration
	// External CLK, External Frame sync, MSB first
	// 24-bit data
	*pSPORT0_TCR1 = TFSR;
	*pSPORT0_TCR2 = SLEN_32;

	// Enable MCM 8 transmit & receive channels
	*pSPORT0_MTCS0 = 0x000000FF;
	*pSPORT0_MRCS0 = 0x000000FF;

	// Set MCM configuration register and enable MCM mode
	*pSPORT0_MCMC1 = 0x0000;
	*pSPORT0_MCMC2 = 0x101c;
}
```

```
//-------------------------------------------------------------------------//
// Function:     Init_DMA
// Description: Initialize DMA2_0 in autobuffer mode to receive and DMA2_1
// in autobuffer mode to transmit
//-------------------------------------------------------------------------//
void Init_DMA(void)
{
        // Set up DMA2 channel 0 to Sport receive
        *pDMA2_0_PERIPHERAL_MAP = 0x0000;

        // Configure DMA2 channel0
        // 32-bit transfers, Interrupt on completion, Autobuffer mode
        *pDMA2_0_CONFIG = WNR | WDSIZE_32 | DI_EN | FLOW_1;
        // Start address of data buffer
        *pDMA2_0_START_ADDR = (void *)iRxBuffer1;
        // DMA inner loop count
        *pDMA2_0_X_COUNT = 8;
        // Inner loop address increment
        *pDMA2_0_X_MODIFY     = 4;


        // Set up DMA2 channel 1 to Sport transmit
        *pDMA2_1_PERIPHERAL_MAP = 0x1000;

        // Configure DMA2 channel 1
        // 32-bit transfers, Autobuffer mode
        *pDMA2_1_CONFIG = WDSIZE_32 | FLOW_1;
        // Start address of data buffer
        *pDMA2_1_START_ADDR = (void *)iTxBuffer1;
        // DMA inner loop count
        *pDMA2_1_X_COUNT = 8;
        // Inner loop address increment
        *pDMA2_1_X_MODIFY     = 4;
}


//-------------------------------------------------------------------------//
// Function:     Init_Interrupts
// Description: Initialize Interrupt for Sport0 RX
//-------------------------------------------------------------------------//
void Init_Sport_Interrupts(void)
{
        // assign interrupt channel 23 (DMA2_0) to IVG9
        *pSICA_IAR2 = Peripheral_IVG(23,9);

        // assign ISRs to interrupt vectors
        // Sport0 RX ISR -> IVG 9
        register_handler(ik_ivg9, Sport0_RX_ISR);
```

```
	// clear pending IVG9 interrupts
	*pILAT |= EVT_IVG9;
	ssync();

	// enable Sport0 RX interrupt
	*pSICA_IMASK0 |= SIC_MASK(23);
	ssync();

}


//---------------------------------------------------------------------------//
// Function:	Enable_DMA_Sport
// Description: Enable DMA2_0, DMA2_1, Sport0 TX and Sport0 RX
//---------------------------------------------------------------------------//
void Enable_DMA_Sport0(void)
{
	// enable DMAs
	*pDMA2_1_CONFIG		= (*pDMA2_1_CONFIG | DMAEN);
	*pDMA2_0_CONFIG		= (*pDMA2_0_CONFIG | DMAEN);

	// enable Sport0 TX and RX
	*pSPORT0_TCR1	= (*pSPORT0_TCR1 | TSPEN);
	*pSPORT0_RCR1	= (*pSPORT0_RCR1 | RSPEN);
}
```

set_PLL.c

```
//*********************************************************************
//***** Set_PLL(CoreCLOCK_multiplier, SCLK_divider)
//***** Sets core and system clock
//*********************************************************************
// Only MSEL and SSEL supported at this time
// No checks for invalid or out of range settings
// The routines makes use of Core B's Supplemental Interrupt 0
//  to synchronise entering the idle state
// Link the routine in both cores' L1 memory and call from each core

#include "main.h"

void Set_PLL(short CoreCLOCK_multiplier, short SCLK_divider)
{ short previous_PLL= *pPLL_CTL;
  short previous_SICA_IWR = *pSICA_IWR0;
  short previous_SICB_IWR = *pSICB_IWR0;
  short new_PLL= (previous_PLL & 0x81ff) | ((CoreCLOCK_multiplier & 0x3f)
<<9);

  if (new_PLL != previous_PLL) {              // Skip if multiplier has not changed.

          if ((int)(*pSRAM_BASE_ADDRESS) == 0xFF800000 ) {          // Do
things for Core A.

                  *pSICB_SYSCR |= 0x0080;                           // Raise Core B
Supplemental-Int0
                  ssync();

                  //while((*pSICB_SYSCR & 0x080));                  // Wait: Core B
Acknowledge SUP-B0

                  *pSICA_IWR0 = (previous_SICA_IWR | 0x1);                // Enable
PLL Wakeup Interrupt.
                  *pPLL_CTL = new_PLL;
                  ssync();

                  idle();                                                 // Put in
idle.

                  *pSICA_IWR0 = previous_SICA_IWR;      // Continue here after idle,
restore previous IWR content.
                  ssync();
            }
          else{                                   // Do things for Core B.

                  while(!(*pSICB_SYSCR & 0x0080));        // Wait: For Core A to
raise SUP-B0
```

```
                *pSICB_SYSCR = 0x0800;                          // Acknowledge
Supplemental Interrupt.
                ssync();

                *pSICB_IWR0 = (previous_SICB_IWR | 0x1); // Enable PLL
Wakeup Interrupt.
                ssync();

                idle();                                                 // Put in
idle.

                *pSICB_IWR0 = previous_SICB_IWR;       // Continue here after idle,
restore previous IWR content.
                ssync();

    }      // if (&_CORE == 0)

  }   // if (new_PLL != previous_PLL)

  *pPLL_DIV = (*pPLL_DIV & 0xFFF0) | SCLK_divider;
  ssync();

}
```

system.h

```
#ifndef  _SYSTEM_DEFINED
#define _SYSTEM_DEFINED

#include "cdefBF561.h"
#include "ccblkfn.h"
#include <sys\exception.h>

#define DSPID                0xFFE05000  // DSP Processor ID Register for MP
implementations
#define pDSPID ((volatile unsigned long *)DSPID)

#define TC_PER                          0xFFC00B0C
#define pTC_PER              (volatile unsigned short *)TC_PER


//*********************************************************************
//***** Multipproject global variables and types
//*********************************************************************
extern int _CORE;

//*********************************************************************
//***** Symbolic constants
//*********************************************************************

// system constants

#define CLKIN          (30.0e6)              // clockin frequency in Hz
#define CORECLK  (600.0e6)              // core clock frequency in Hz
#define SYSCLK       (120.0e6)              // system clock frequency in Hz



#endif
```

ÖZGEÇMİŞ

1982 yılında Bursa'da doğdu. İlk, orta ve lise öğrenimini Bursa'da tamamladı. 2001 yılında girdiği Kocaeli Üniversitesi Teknik Eğitim Fakültesi Elektronik Öğretmenliği Bölümü'nden 2005 yılında mezun oldu. 2005 yılından beri Kocaeli Üniversitesi Mühendislik Fakültesi Mekatronik Mühendisliği Bölümü'nde Yüksek Lisans'a devam etmektedir.