

# BÖLÜM 1

## GİRİŞ

Hızla artan nüfusa paralel olarak araçlarda hızlı artışta görülmektedir. Araçların artması ile sürücülerin yola daha fazla adapte olmaları için farklı sistemler geliştirilmektedir. Trafik işaretlerinin artması, yeni kuralların konulması ve benzeri gelişmelerden ötürü sürücülerin araç kullanırken birden fazla odaklanmaları gereken nokta olduğu için sürücülerin araç kullanımı zorlanmaktadır. Sürücülerin dikkatlerinin dağılmasını önlemek ve sürücüye daha rahat bir sürüş sağlamak için günümüzde otomatik olarak şeritleri tanıyan ve bu şeritler arasında aracın kalmasını sağlayan sistemlerin geliştirilmesine önem verilmektedir.

Görüntü işleme metotlarıyla buna paralel olarak gelişen görsel teknoloji ile birlikte otomotiv firmaları yeni araçlarına yol güvenliğini artırmamak için yeni sistemler eklemeye başladılar. Bunlardan en yaygın olarak kullanılan yol çizgilerini takip etme, öndeki araç mesafesini takip etme gibi görsel teknolojiyi kullanan yeni sistemler geliştirilmektedir. Google, Tesla, ve Nvidia gibi firmalar bu sistemler üzerine çalışan başlıca firmalardır.

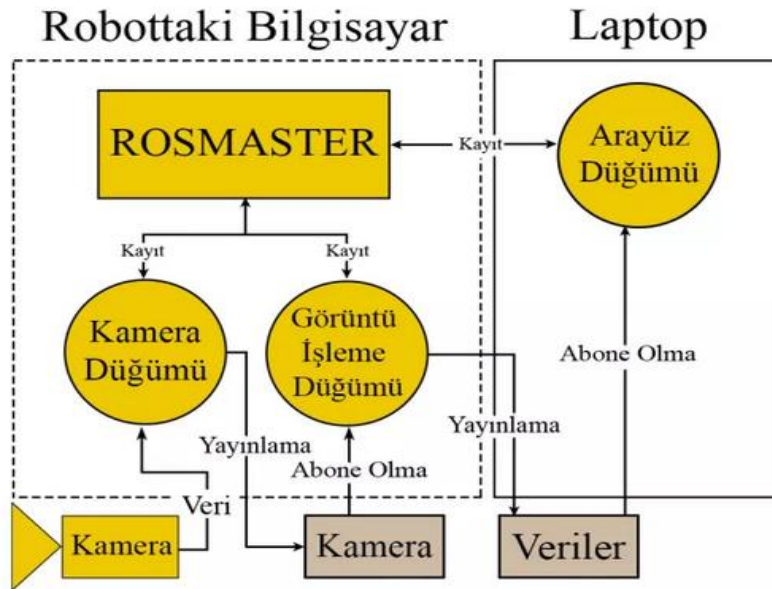
Günümüzde şerit takibi yapan sistemler, sürücünün şeridinde ilerlemesinde, şerit dışına çıkıldığında uyarı veren ve şeridine tekrardan girmesine yardımcı olmaktadır. Bu nedenle şerit tanımlama ve takip sistemleri geliştirilmesine gerek duyulmaktadır.

### 1.1 Robot Operating System(ROS)

ROS, bilgisayarlar üzerinden robotlarımızı ve robot bileşenlerimizi kontrol etmemizi sağlayan BDS lisanslı, açık kaynak kodlu bir yazılım sistemidir. Kurulum için Linux işletim sistemine ihtiyaç duyan ROS'un amacı robot ve programcı arasında standartlaşmış bir yapı oluşturmaktır. Böylece başkasının yazmış olduğu bir kodu rahatlıkla kendi projenizde kullanabilirsiniz. ROS kendi içerisinde bulunduğu standart kütüphaneler ile desteklediği robotlar dışında geliştiriciler tarafından yazılan kütüphaneler ve sürücülerle desteklediği robot sayısı günden güne artmaktadır.

## 1.2 ROS Nasıl Çalışır?

Bir ROS sistemi yayın yapma ve abone olma modelini kullanarak aralarında haberleşen düğümlerden oluşur. Düğümler hesaplama işlemi yapabilen ve elde ettikleri verileri başlıklar altında yayınlatabilen veya başka bir düğümün yayınladığı verileri başlıklara abone olarak edinebilen birimlerdir. Düğümlerin haberleşmesini sağlayan “rosmaster” ağ tabanlı XML-RPC bir sunucudur. Bir ROS sisteminde birden çok düğüm ve başlık bulunabilir. Örneğin; Kamera verilerini kamera başlığı altında yayınlayan bir düğümün başlığına abone olup görüntü verilerini işleyerek veriler isimli bir başlıktan yayınlayan bir başka düğüm olabilir. Bir ara yüz düğümü yazar ve veriler başlığına abone olup gelen verileri ekrana çizdirebilirsiniz.



Şekil 1.2.1: ROS Düğüm Örneği

## 1.3 C++ Programlama Dili

C++, 1979 yılında Bjarne Stroustrup tarafından Bell Labs’de geliştirilen nesne yönelimli ve yüksek seviyeli, genel maksatlı programlama dilidir. C++’ın ilk ismi “C With Classes” dir ve C programlama dilinin bir eklentisi olarak işlev görmektedir. C++’ın yaratıcısı Bjarne Stroustrup bu programlama dilini öğrenci iken geliştirmiştir. Kullandığı

programlama dili yeteri kadar hem işlevli hem de yüksek performanslı görmeyen Stroustrup, kendi programlama dilini oluşturarak bilgisayar tarihinin en önemli yazılım dilinden birinin ortaya çıkmasına yardımcı olmuştur. C++ programlama dili C tarzında veya nesne yönelimli tarzda kesin senaryolarla kodlamalar yapılabilir. Bu açıdan C++ en önemli ve işlevsel hibrit programlama dillerinden biri olma niteliğine sahiptir.

C++ geliştirilmesinden sonra tüm dünyada en yaygın kullanılan programlama dillerinden biri olmuştur. Özellikle söz konusu performans olduğunda C++ daima ilk seçim olmaktadır. Zira programlama dilleri ile kıyaslandığında C++ çok daha fazla güncellenen bir yazılım dilidir. C++ yazılım dili kullanılarak geliştiriciler tarafından sistem yazılımları, özel yazılımlar, uygulamalar, sürücü yazılımları, kullanıcı taraflı yazılımlar ve gömülü firmware yazılımları üretilmektedir.

## 1.4 OpenCV

OpenCV (Open Source Computer Vision) açık kaynak kodlu görüntü işleme kütüphanesidir. 1999 yılında Intel tarafından geliştirilmeye başlanmış daha sonra Itseez, Willow, Nvidia, AMD, Google gibi şirket ve toplulukların desteği ile gelişim süreci devam etmektedir. İlk sürüm olan OpenCV alfa 2000 yılında piyasaya çıkmıştır. İlk etapta C programlama dili ile geliştirilmeye başlanmış ve daha sonra birçok algoritması C++ dili ile geliştirilmiştir. Open source yani açık kaynak kodlu bir kütüphanedir ve BSD lisansı ile altında geliştirilmektedir. BSD lisansına sahip olması bu kütüphaneyi istediğiniz projede ücretsiz olarak kullanabileceğiniz anlamına gelmektedir. OpenCV platform bağımsız bir kütüphanedir, bu sayede Windows, Linux, FreeBSD, Android, Mac OS ve iOS platformlarında çalışabilmektedir. C++, C, Python, Java, Matlab, EmguCV kütüphanesi aracılığıyla da Visual Basic.Net, C# ve Visual C++ dilleri ile topluluklar tarafından geliştirilen farklı wrapperlar aracılığıyla Perl ve Ruby programlama dilleri ile kolaylıkla OpenCV uygulamaları geliştirilebilir. 2016-05-27 tarihli güncelleme, OpenCV geliştirici Itseez firması Intel tarafından satın alındı. OpenCV geliştirmesine Intel çatısı altından devam edeceğini duyurdu. OpenCV kütüphanesi içerisinde görüntü işlemeye (image processing) ve makine öğrenmesine (machine learning) yönelik 2500'den fazla algoritma bulunmaktadır. Bu algoritmalar ile yüz tanıma, nesneleri ayırt etme, insan hareketlerini tespit edebilme, nesne sınıflandırma, plaka tanıma, üç boyutlu görüntü

üzerinde işlem yapabilme, görüntü karşılaştırma, optik karakter tanımlama OCR (Optical Character Recognition) gibi işlemler rahatlıkla yapılabilmektedir.

## 1.5 OpenCV Bileşenleri

- **Core:** OpenCV'nin temel fonksiyonları ve matris, point, size gibi veri yapılarını bulundurur. Ayrıca görüntü üzerine çizim yapabilmek için kullanılabilecek metotları ve XML işlemleri için gerekli bileşenleri barındırır.
- **HighGui:** Resim görüntüleme, pencereleri yönetme ve grafiksel kullanıcı arabirimleri için gerekli olabilecek metotları barındırır. 3.0 öncesi sürümlerde dosya sistemi üzerinden resim dosyası okuma ve yazma işlemlerini yerine getiren metotları barındırmaktaydı.
- **Imgproc:** Filtreleme operatörleri, kenar bulma, nesne belirleme, renk uzayı yönetimi, renk yönetimi ve eşikleme gibi neredeyse tüm fonksiyonları içine alan bir pakettir. 3 ve sonra sürümlerde bazı fonksiyonlar değişmiş olsada 2 ve 3 sürümünde de bir çok fonksiyon aynıdır.
- **Imgcodecs:** Dosya sistemi üzerinden resim ve video okuma/yazma işlemlerini yerine getiren metotları barındırmaktadır.
- **Videoio:** Kameralara ve video cihazlarına erişmek ve görüntü almak ve görüntü yazmak için gerekli metotları barındırır. OpenCV 3 sürümü öncesinde bu paketteki birçok metot video paketi içerisindeydi.

## BÖLÜM 2

### Tasarım Ve Geliştirilen Uygulamalar

#### 2.1 Araba Modeli Belirleme

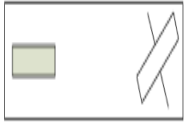
Bu projede otonom insansız kara araç tasarlanarak, aracın kendi kendine hareket ederek şerit içerisinde kalması hedeflenmiştir.

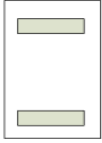
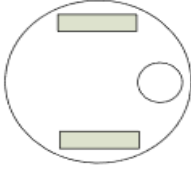
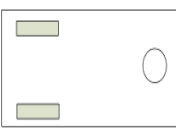
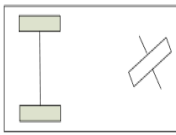
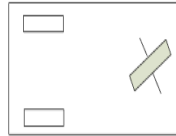
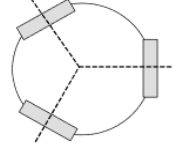
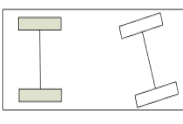
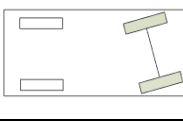
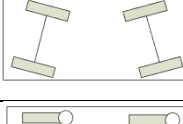
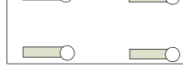
Son yıllarda ortaya çıkan otonom insansız kara araçları, aracın kendi kendisini kontrol ettiği insandan kontrolünden bağımsız çalışan insansız kara araçlarıdır. Bu tip araçlar avantajlı olmakla beraber uygulaması oldukça pahalı ve zordur. Kendisine verilen belirli bir görevi (yükü bir yerden bir yere taşıma, görme engelli bir hastayı evden işe götürme gibi) yolunu etraftaki engellerden sakınarak belirleyen ve etraftaki yapıları, duvarları, araçları tespit ederek uygulamaya çalışmaktadır. Bu tip araçların kullanılmasındaki en önemli sebep insan kontrolünün olmadığı bölgeler veya zamanlarda robotun verilen işlevi yerine getirebilmesidir.

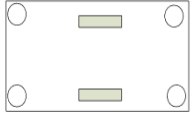






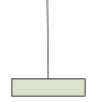
##### 2.1.1 Tahrik Sistemleri

İnsansız kara araçlarının çevrede dolaşabilmesi için çeşitli tahrik mekanizmalarına ihtiyaçları vardır. Bu mekanizmalar tekerlekli olabileceği gibi, paletli, ayaklı da olabilmektedirler. Bunun için robotun hangi amaçla kullanılacağı tahrik sistemini belirlemek için önemlidir. En çok kullanılan tahrik sistemi tekerlektir. Aracın hareket kabiliyeti açısından tekerlek yapıları çok önemlidir. Tekerlekli insansız kara araçlarında kullanılan standart, nakliye, isveç ve küresel olmak üzere dört temel tekerlek sınıfı mevcuttur. Tablo 2.1.1’de çeşitli teker konfigürasyon yapıları görülmektedir.

Tablo 2.1.1: Tahrik Sistemleri

Teker Sayısı	Teker Düzeni	Açıklama	Örnek
2		Yön belirleyen bir ön teker, gövdeyi ön tekerin çektiği yere sürükleyen bir arka teker vardır.	Bisiklet Motersiklet

		İki teker bir mifle baęlıdır, aęırlık merkezi milin orta noktasıdır.	
3		Önde bir teker, arkada farklı sürüş merkezleri olan iki teker bulunmaktadır.	
		Önde çok yönlü bir teker, arkada iki bağımsız teker vardır.	Çoęu kapalı ortam robotları (P3malion ve Alice)
		Önde yön veren bir teker, arkada bir mifle birbirine baęlı iki ayrı teker vardır.	
		Önde yön veren bir teker, arkada iki teker vardır.	Neptune (Carnegie Mellon Üniversitesi)
		Bir üçgen düzenine oturtulmuş üç adet çok yönlü İsveç tekeri veya küresel teker.	Tribolo EPFL, Palm Pilot Robot Kit (Carnegie Mellon Üniversitesi)
4		Önde yön veren iki teker, arkada motorlu iki teker bulunmaktadır.	Arkadan sürüşlü arabalar
		Önde iki motorlu ve yön veren teker, arkada iki serbest teker vardır.	Önden sürüşlü arabalar
		Dört adet yön belirleyen ve motorlu teker vardır.	Dört tekerli sürüşe sahip Hyperion (Carnegie Mellon Üniversitesi)
		Dört adet motorlu ve yön belirleyen taşıyıcı tekerler vardır.	Nomad XR4000

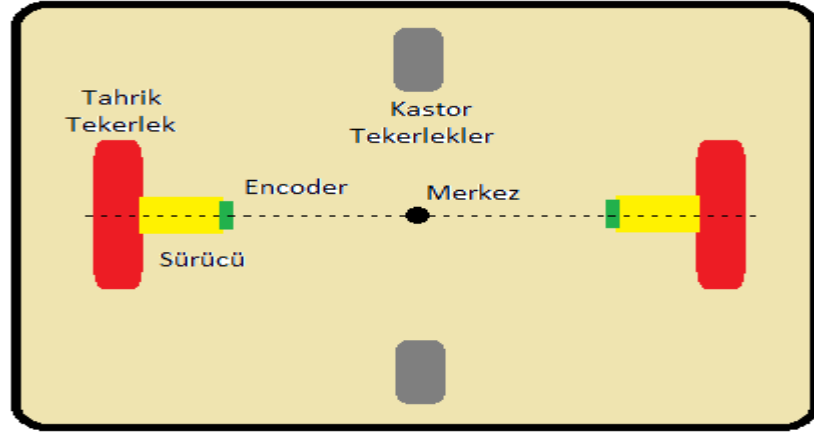
6		Merkezde iki adet çekici (sürükleyici) teker, her köşede birer tane çok yönlü teker vardır.	Terragator (Carnegie Mellon Üniversitesi)
Yukarıda gösterilen teker tiplerindeki ikonların anlamı			
	Güç verilmemiş çok yönlü teker (küresel, nakliye tekeri, İsveç tekeri)		
	Motorlu İsveç tekeri (Stanford tekeri)		
	Güç verilmemiş standart teker		
	Motorlu ve yön belirleyen nakliye tekeri		
	Yön belirleyen standart teker		
	Motorlu standart teker		
	Bağlı tekerler		

Robot uygulamalarında yukarıdaki konfigürasyondan bazıları çok kullanılmamaktadır. En çok kullanılan tekerlek yapılarından ilki Ackermann tekerlek yapısı diye bildiğimiz ve 4 tekerlek yapısının ilk sırasında bulunan konfigürasyona benzemektedir. En çok kullanılan ikinci tekerlek yapısı diferansiyel sürücü tipidir ki 3 tekerlekli konfigürasyonun ilk sırasındaki gibi olabileceği gibi 6 tekerlekli terragator yapısında da olabilir. 4 tekerlekli önde ve arkada kastor tekerlekler ve her iki yan da sürükleyici iki teker yapısında da olabilir. Biz uygulamamızda diferansiyel tekerlek yapısındaki konfigürasyonları kullanacağız.

### 2.1.2 Diferansiyel Tekerlek Modeli

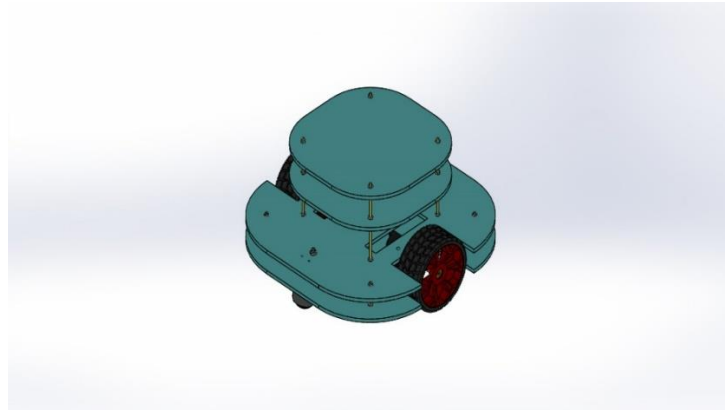
Diferansiyel tekerlek yapısı Ackermann ve birçok tekerlek yapılarına göre kontrolü karmaşık ve zor olsa da birçok uygulama da hareket yapısının esnek olması, dönme, geri geri gelme, engellerden hızlı bir şekilde yön değiştirebilme gibi istenilen hareketleri rahatlıkla yapabilme kabiliyetlerinden dolayı tercih edilmektedir. Diferansiyel tekerlek yapısı gerek 3 tekerlekli, gerek 4 tekerlekli gerekse de 6 tekerlekli yapıda

olabilmektedir. 4 tekerlekli yapıda önde ve arkada iki kaster orta da ise 2 adet tahrik tekerleği; Şekil 2.1.2.1’da görüldüğü gibidir. Bizde projemizde modeli kullanacağız.



Şekil 2.1.2.1: 4 Tekerlekli Diferansiyel Tekel Modeli

Model seçimi yapıldıktan sonra araç SolidWorks ortamında çizilip montaj edilerek incelendi.



Şekil 2.1.2.2: SolidWorks ile Tasarlanan Model

Çizim işlemlerinde sonra “Pleksiglas” isimli dayanıklı bir plastikten çizilen parçalar kesilip, araç gerçek ortamda toplanmıştır.

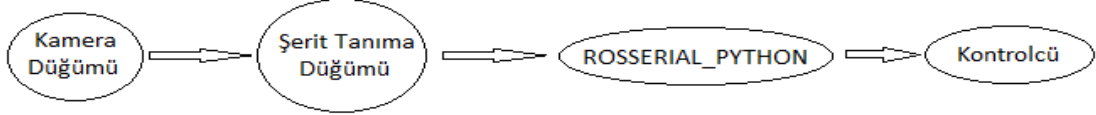


Şekil 2.1.2.3: Diferansiyel Tekel Modelli Araç



## 2.2 Şerit Tanıma Ve Araç Kontrolü

Şerit tanıma ve araç kontrolü için ROS tabanlı düğümler oluşturulmuştur. Kamera, şerit tanıma ve kontrolcü için ayrı ayrı düğümler oluşturulmuştur. Ek olarak “ROSSERIAL\_PYTHON” kullanılmıştır.



Şekil 2.2.1: Akış Şeması

### 2.2.1 Kamera Düğümü

Kamera düğümü, kameradan alınan görüntünün ROS yardımı ile video\_ başlığı altında yayınlanarak diğer düğümlere görüntü yayını sağlamaktadır.

ROS nesnesi ve yayıncı başlığı oluşturuldu ve webCam ayarları yapıldı.

```
ros::init(argc,argv,"video");
ros::NodeHandle nh;
image_transport::ImageTransport it(nh);
image_transport::Publisher pub;
pub = it.advertise("video_",1);

sensor_msgs::ImagePtr msg;
Mat image;

VideoCapture video(0);
ros::Rate loop_rate(20);
```

Daha sonra kameradan görüntü okunup yayınla “video\_” başlığı altında yayınlandı.

```
while(nh.ok()){
    video >> image;
    msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8",image).toImageMsg();
    pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
}
```

### 2.2.2 Şerit Algılama Düğümü

Şerit algılama düğümü, video\_ başlığı altında aldığı görüntüyü işleyerek görüntüdeki şeridi algılayıp, araç merkezi ile iki şerit arasındaki orta nokta arasındaki açıyı hesaplayıp, açı değerini “angle” başlığı ile yayınlamaktadır.

Şerit bulma işlemi için öncelikle gelen görüntünün tek kanallı olması gerekiyor. Bu işlem için OpenCV kütüphanesine ait olan `cvtColor()` metodu kullanıldı. Daha sonra ise görüntüdeki gürültülerden kurtulmak için `GaussianBlur()` metodu kullanıldı.



Şekil 2.2.2.1: `cvtColor()` işlemi



Şekil 2.2.2.2: `GaussianBlur()` işlemi

Bulanıklaştırma işleminden ardından şerit çizgilerini algılamak için kenar bulma işlemi gerçekleştirildi. Bunun için `Canny()` metodu kullanıldı. Bu işlem sonunda diğer nesnelerinde kenarları da algılandı bu kenarlardan kurtulmak için basit bir kırpma işlemi yapıldı bunun için `polygon_create()` metodu oluştururdu. Bu metot ile görüntünün belirli bir kısmı ile devam edilecek.

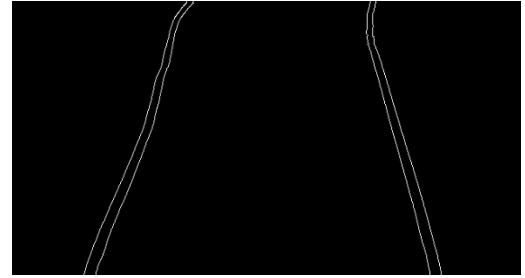
```
Mat Strip::polygon_create(Mat image){
    Mat mask = Mat::zeros(image.rows,image.cols,CV_8UC1);

    Point point[1][4];
    point[0][0]=Point(145,90);
    point[0][1]=Point(image.cols-145,90);
    point[0][2]=Point(image.cols-50,image.rows-50);
    point[0][3]=Point(50,image.rows-50);

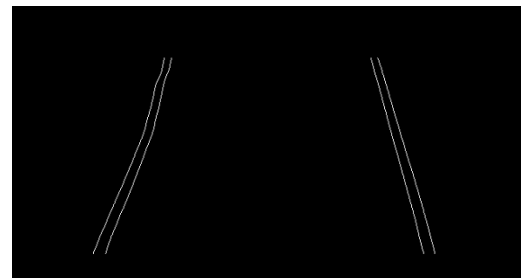
    const Point* ppt[1] = { point[0] };
    int npt[] = {4};

    fillPoly(mask,ppt,npt,1,Scalar(255,255,255),8);
    bitwise_and(image,mask,mask);

    return mask;
}
```



Şekil 2.2.2.3: `Canny()` Metodu



Şekil 2.2.2.4: `polygon_create()` Metodu

Kırpma işleminden sonra kalan kenarları algılamak için OpenCV kütüphanesine ait olan HoughLinesP() metodu ile kenarlar çizgi olarak algılanıp, line\_draw() metodu ile şeritler belirlendi.

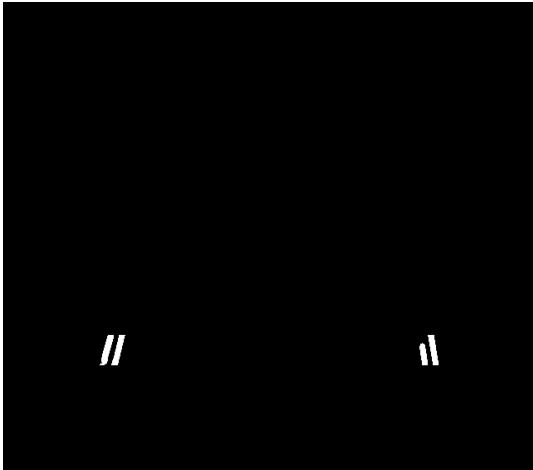
```
HoughLinesP(maskImg,lines,2,CV_PI/180,10,3,3);

void Strip::line_draw(Mat image){
    for(int i=0;i<lines.size();i++){
        line(image,Point(lines[i][0],lines[i][1]),Point(lines[i][2],lines[i][3]),Scalar(255,255,0),3);
    }
    circle(image,Point((320),center.y),10,Scalar(0,255,255),3);
}
```

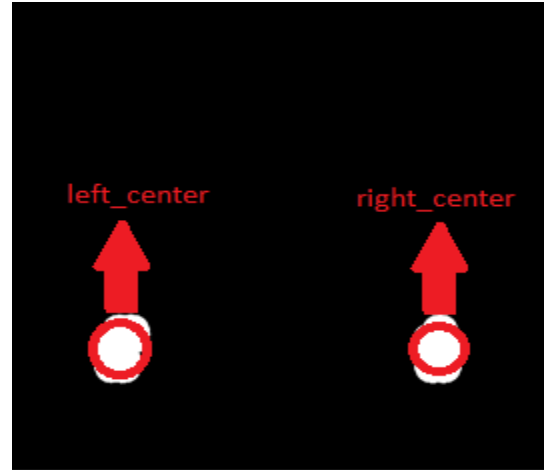


Şekil 2.2.2.5: line\_draw() Metodu

Şeritler çizildikten sonra iki şerit arasındaki orta noktanın bulunması için “midpoint()” metodu oluşturuldu. Bu metot ile şeritlerin belirli bir kısmı alınarak bu kısımların orta noktası bulunur, daha sonra iki şerit arasında ki mesafe bulunarak orta nokta belirlendi. Belirlenen nokta “center” adında bir değişkene atandı.



Şekil 2.2.2.6: Şeritlerin Belirli Bir Kısım



Şekil 2.2.2.7: Şeritlerin Orta Noktası

```

Point2f left_center,right_center;
float left_radius,right_radius;

minEnclosingCircle(count[0],left_center,left_radius);
minEnclosingCircle(count[1],right_center,right_radius);

center.x = fabs((int)((left_center.x - right_center.x) / 2));
center.y = fabs((int)((left_center.y - right_center.y) / 2));

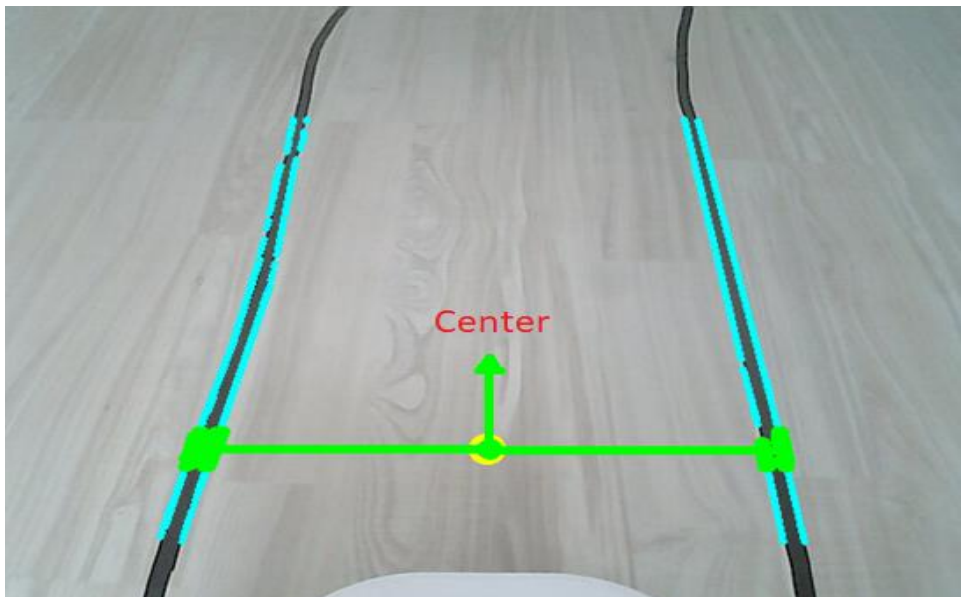
center.x = center.x + right_center.x;
center.y = center.y + right_center.y;
circle(orjImage,center,6,Scalar(0,255,0),3);
line(orjImage,left_center,right_center,Scalar(0,255,0),5);

double My=200;
double Mx;
double multiplier = 1;

if(320 > center.x){
    Mx = ((double)((320) - center.x))*multiplier;
    theta = ((atan(Mx/My))*(180/3.14159265359));
}else if(320 < center.x){
    Mx = ((double)(center.x - (320)))*multiplier;
    theta = ((-1)*(atan(Mx/My))*(180/3.14159265359));
}else{
    theta = 0;
}

```

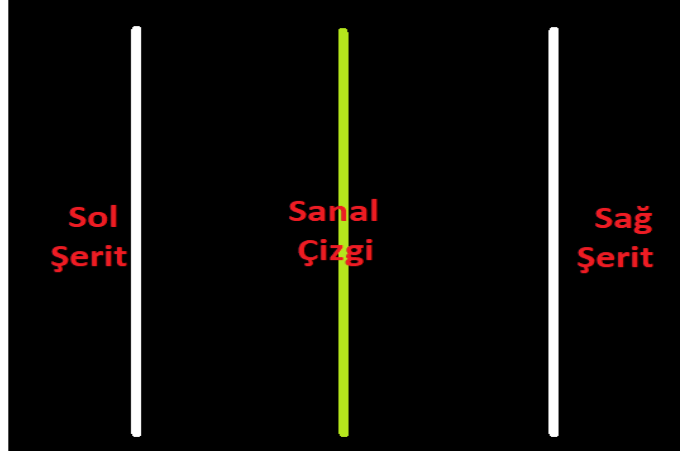
Sonuç olarak oluşan görüntü;



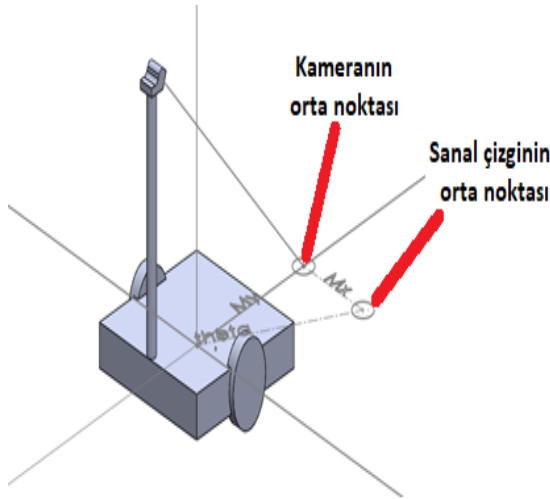
Şekil 2.2.2.8: İki Şerit Arasındaki Merkez Nokta

### 2.2.2.1 Diferansiyel Aracın Hareketi

Aracın hareket mantığı çizgi izleyen robot mantığına benzerdir. Burada iki şerit arasına sanal bir çizgi oluştururdu. Ve aracın orta noktasının bu çizgi üzerinde durması sağlanarak araç kontrollü sağlanmıştır.



Şekil 2.2.2.1.2: Örnek Sanal Çizgi



Şekil 2.2.2.1.3: Koodinat Sistemi

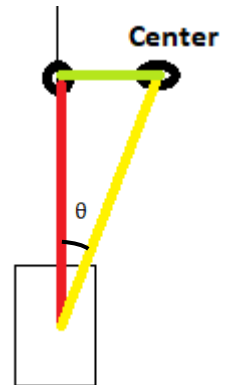
Kamera orta noktası ile aracın orta noktası arasında mesafe(My) sabit ve 200mm'dir. Sanal çizginin orta noktası ve kamerasının orta noktası arasındaki mesafe(Mx) değişkendir. Bu değişim aracın konumuna bağlı olarak değişir. Bu iki nokta arasındaki mesafe sıfır olması istenir. Bunun için matematiksel bir hesap ile aracın dönme açısı hesaplanır ve bu açı kontrolcüye gönderilir.

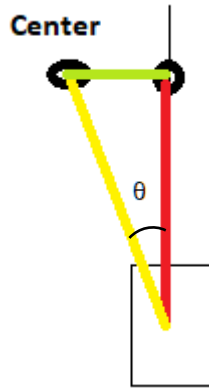
Eğer kamera merkezi sanal çizginin solunda ise araç sağa dönme hareketi yaparak ilerlemeyi sağlar.

$$M_x = 320 - \text{center} \cdot x$$

$$\theta = \arctan\left(\frac{m_x}{M_y}\right)$$

Şekil 2.2.2.1.4: Aracın Sağa Dönmesi





Eğer kamera merkezi sanal çizginin sağında ise araç sola dönme hareketi yaparak ilerlemeyi sağlar.

$$M_x = |320 - \text{Center} \cdot x|$$

$$\theta = (-1) \cdot \arctan\left(\frac{M_x}{M_y}\right)$$

Şekil 2.2.2.1.5: Aracın Sola Dönmesi

```
double My=200;
double Mx;
double multiplier = 1;

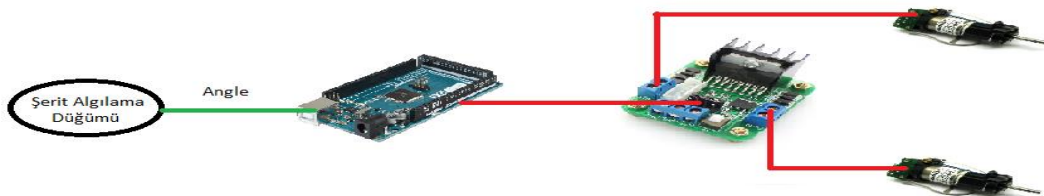
if(320 > center.x){
    Mx = ((double)((320) - center.x))*multiplier;
    theta = ((atan(Mx/My))*(180/3.14159265359));
}else if(320 < center.x){
    Mx = ((double)(center.x - (320)))*multiplier;
    theta = ((-1)*(atan(Mx/My))*(180/3.14159265359));
}else{
    theta = 0;
}
```

Daha sonra hesaplanan açı değeri ROSSERIAL ile kontrolcüye gönderilir.

```
angle.data = theta;
pub_.publish(angle);
```

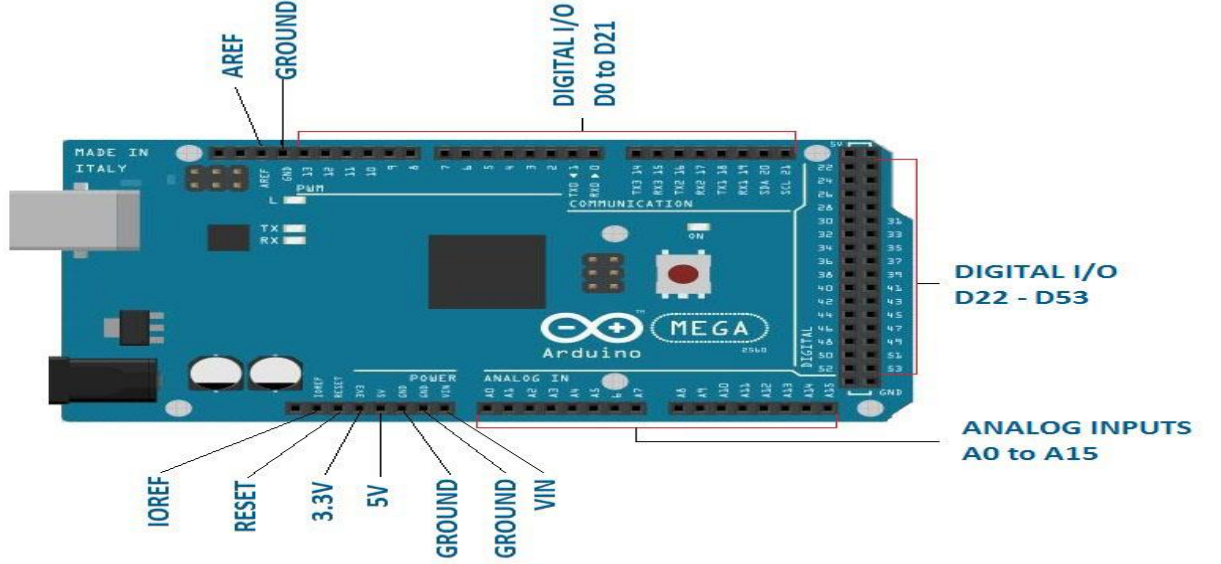
### 2.2.3 Kontrolcü Tasarımı

Şerit algılama düğümünden “angle” başlığı altında yayınlanan açı değeri “ROSSERIAL\_PYTHON” düğümü vasıtasıyla tasarlanan kontrolcüye gönderilir. Burada tasarlanan PI kontrolcü ile aracın kontrolü sağlanır.



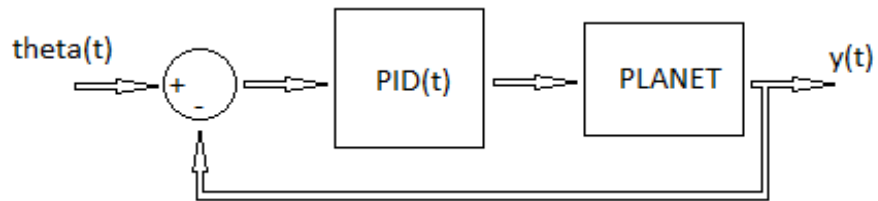
Şekil 2.2.3.1: Basit Devre Şeması

Kontrolcü için “ARDUINO MEGA” kiti kullanılmıştır. Bu kit, dört adet kesme pini ve çok sayıda dijital giriş-çıkış pini bulundurduğu için tercih edilmiştir.



Şekil 2.2.3.2: Arduino Mega

Sistemin kontrolü için bir PI kontrolcü tasarlandı. Değerler deneme yöntemi ile bulunmuştur. Denemeler sonucunda  $K_p = 5$  ve  $K_i = 0.003$  seçilmiştir.



Şekil 2.2.3.3: Kontrolcü Modeli

Kontrolcü için calcPID() adında bir fonksiyon oluşturuldu.

```

float calcPID(float angle){

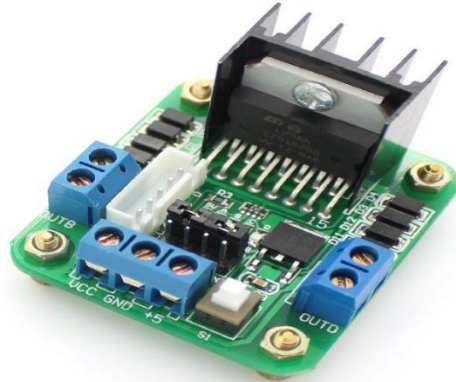
    float errorDiff;
    float output;
    double dt=(double) (now-LastTime);

    error = 0 - angle;
    errorInt += error * dt;
    errorDiff = (error - errorLast)/dt;
    output = Kp*error + Ki*errorInt + Kd*errorDiff;
    errorLast = error;
    LastTime=now;

    return output;
}

```

Kontrolcü sonucu hesaplanan değeri PWM sinyali olarak sürücü kartına yollanmıştır. Sürücü kartı olarak L298N kiti kullanılmıştır. Bu kit iki adet çıkışa ve her çıkış 2A çıkış verdiği için seçilmiştir.



Şekil 2.2.3.4: L298 Sürücü Kiti

Bu kitin sürdüğü motorlar ise Namiki firmasının ürettiği fırçasız dc motor olan ve üzerinde redüktör ve enkoder olan bir motordur. Motor mil çıkışı 9600rpm iken redüktör çıkışı 120 rpm'dir. 5kg tork ile çalışabiliyor. Çalışma gerilimi 12v'tur.



Şekil 2.2.3.5: DC Motor



## 2.3 Sonuç Ve Önerme

Tasarlanan araç modeli istenilen sonucu vermiştir. Mobil araç iki şeridi algılayarak bu iki şerit ortasında sanal bir çizgi oluşturarak bu çizgiyi takip etmiştir.

Bu sistem sadece görüntü işleme tabanlı olarak şerit algılaması yapmaktadır. Bu yöntemi geliştirerek yani yapay zeka ve buna benzer sistemler ile bu algıma daha başarılı olabilir. Buna ek olarak araç sadece şerit algılaması yapmaktadır. Ama aracın çevresini algılayıp, buna göre kararlar vermesi sağlanabilir. Belirlenen bir hedefe gönderilebilir. Trafik tabelalarını algılayabilir ve bunlara göre tepki verebilir. Bu tür çalışmalar dünya çapında çok sayıda firma tarafından gerçekleştirilmektedir. Yukarıda yazılan bütün sistemler bu tasarlanan araca entegre edilebilir.

## EK-1

```
1  #include <ros/ros.h>
2  #include <cv_bridge/cv_bridge.h>
3  #include <image_transport/image_transport.h>
4
5  #include <opencv2/highgui/highgui.hpp>
6
7  using namespace cv;
8
9  int main(int argc, char **argv){
10
11     ros::init(argc,argv,"video");
12     ros::NodeHandle nh;
13     image_transport::ImageTransport it(nh);
14     image_transport::Publisher pub;
15     pub = it.advertise("video_",1);
16
17     sensor_msgs::ImagePtr msg;
18
19     Mat image;
20
21     VideoCapture video(0);
22
23     ros::Rate loop_rate(20);
24
25     while(nh.ok()){
26         video >> image;
27         msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8",image).toImageMsg();
28         pub.publish(msg);
29         ros::spinOnce();
30         loop_rate.sleep();
31     }
32 }
```

## EK-2

```
1  #include <ros/ros.h>
2  #include <std_msgs/Float32.h>
3  #include <cv_bridge/cv_bridge.h>
4  #include <image_transport/image_transport.h>
5  #include <opencv2/opencv.hpp>
6  #include <opencv2/core/core.hpp>
7  #include <opencv2/highgui/highgui.hpp>
8  #include <algorithm>
9  #include <iostream>
10 #include <string>
11 #include <ctime>
12 #include <cmath>
13
14
15 using namespace cv;
16 using namespace std;
17
18
19
20 class Strip{
21     ros::NodeHandle nh_;
22     ros::Publisher pub_;
23     image_transport::ImageTransport it_;
24     image_transport::Subscriber sub_;
25
26 public:
27
28     Strip() : it_(nh_){
29         sub_ = it_.subscribe("image_raw", 1, &Strip::imageCallback, this);
30         pub_ = nh_.advertise<std_msgs::Float32>("angle", 1000);
31     }
32
33
34     ~Strip(){
35         destroyAllWindows();
36     }
37
38     void imageCallback(const sensor_msgs::ImageConstPtr& msg);
39     void strip_find(Mat image);
40     Mat polygon_create(Mat image);
41     void line_draw(Mat image);
42     void midpoint(Mat image, Mat cInImage, Mat orjImage);
43     void calculate_angle();
44
45 public:
46     std_msgs::Float32 angle;
47     vector<Vec4i> lines;
48     Point center;
49     double theta;
50 };
```

```

52
53 void Strip::imageCallback(const sensor_msgs::ImageConstPtr& msg){
54     Mat image;
55     cv_bridge::CvImagePtr bridge_image;
56
57     try{
58         bridge_image = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
59         image = bridge_image->image;
60         Strip::strip_find(image);
61     }catch(cv_bridge::Exception& ex){
62         cout<<"Okuma Hatası..."<<endl;
63     }
64 }
65
66 }
67
68 void Strip::strip_find(Mat image){
69     Mat grayImg;
70     Mat maskImg;
71     Mat orjImg;
72     Mat clnImg = image.clone();
73
74     cvtColor(image,grayImg,CV_BGR2GRAY);
75     GaussianBlur(grayImg,grayImg,Size(11,11),3,3);
76     Canny(grayImg,grayImg,10,100);
77     maskImg = polygon_create(grayImg);
78     HoughLinesP(maskImg,lines,2,CV_PI/180,10,3,3);
79     line_draw(image);
80     midpoint(maskImg,clnImg,image);
81     imshow("Show",image);
82     waitKey(30);
83 }

```

```

84
85 Mat Strip::polygon_create(Mat image){
86     Mat mask = Mat::zeros(image.rows,image.cols,CV_8UC1);
87
88     Point point[1][4];
89     point[0][0]=Point(145,90);
90     point[0][1]=Point(image.cols-145,90);
91     point[0][2]=Point(image.cols-50,image.rows-50);
92     point[0][3]=Point(50,image.rows-50);
93
94     const Point* ppt[1] = { point[0] };
95     int npt[] = {4};
96     fillPoly(mask,ppt,npt,1,Scalar(255,255,255),8);
97     bitwise_and(image,mask,mask);
98     return mask;
99 }
100
101 void Strip::line_draw(Mat image){
102
103
104     for(int i=0;i<lines.size();i++){
105         line(image,Point(lines[i][0],lines[i][1]),Point(lines[i][2],lines[i][3]),Scalar(255,255,0),3);
106     }
107     circle(image,Point((320),center.y),10,Scalar(0,255,255),3);
108 }
109
110
111

```

```

112 void Strip::midpoint(Mat image,Mat clnImage,Mat orjImage){
113
114     Mat circle_mask = Mat::zeros(image.rows,image.cols,CV_8UC1);
115     Mat line_mask = Mat::zeros(image.rows,image.cols,CV_8UC1);
116     Mat poly_mask = Mat::zeros(image.rows,image.cols,CV_8UC1);
117
118     for(int i=0;i<lines.size();i++){
119         line(line_mask,Point(lines[i][0],lines[i][1]),Point(lines[i][2],lines[i][3]),Scalar(255,255,255),5);
120     }
121

```

```

122 Point point[1][4];
123 point[0][0]=Point(0,340);
124 point[0][1]=Point(image.cols,340);
125 point[0][2]=Point(image.cols,370);
126 point[0][3]=Point(0,370);
127
128 const Point* ppt[1] = { point[0] };
129 int npt[] = {4};
130 fillPoly(poly_mask,ppt,npt,1,Scalar(255,255,255),8);
131 bitwise_and(poly_mask,line_mask,poly_mask);
132
133 vector<Vec4i> Lines;
134 HoughLinesP(poly_mask,Lines,2,CV_PI/180,10,3,3);
135
136 for(int i=0;i<Lines.size();i++){
137     circle(circle_mask,Point(Lines[i][0],Lines[i][1]),20,Scalar(255,255,255),-1);
138     circle(circle_mask,Point(Lines[i][2],Lines[i][3]),20,Scalar(255,255,255),-1);
139     line(orjImage,Point(Lines[i][0],Lines[i][1]),Point(Lines[i][2],Lines[i][3]),Scalar(0,255,0),5);
140 }
141
142 vector<vector<Point>> count;
143 vector<Vec4i> hier;
144 findContours(circle_mask,count,hier,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
145 if(count.size() == 2){
146     Point2f left_center,right_center;
147     float left_radius,right_radius;
148
149     minEnclosingCircle(count[0],left_center,left_radius);
150     minEnclosingCircle(count[1],right_center,right_radius);
151     imshow("circle",circle_mask);///
152     center.x = fabs((int)((left_center.x - right_center.x) / 2));
153     center.y = fabs((int)((left_center.y - right_center.y) / 2));
154

```

```

155     center.x = center.x + right_center.x;
156     center.y = center.y + right_center.y;
157     circle(orjImage,center,6,Scalar(0,255,0),3);
158     cout << center << endl;
159     line(orjImage,left_center,right_center,Scalar(0,255,0),5);
160
161
162     double My=200;
163     double Mx;
164     double multiplier = 1;
165     if(320 > center.x){
166         Mx = ((double)((320) - center.x))*multiplier;
167         theta = ((atan(Mx/My))*(180/3.14159265359));
168     }else if(320 < center.x){
169         Mx = ((double)(center.x - (320)))*multiplier;
170         theta = ((-1)*(atan(Mx/My))*(180/3.14159265359));
171     }else{
172         theta = 0;
173     }
174 }
175
176
177
178 cout << thita << endl;
179 angle.data = theta;
180 pub_.publish(angle);
181 }

```

```
182
183
184 ☐ int main(int argc, char **argv){
185     ros::init(argc,argv,"strip_node");
186     Strip strip;
187     ros::spin();
188 }
```

## EK-3

```
1  #include <ros.h>
2  #include <std_msgs/Float32.h>
3
4  const int trigPin = 21;
5  const int echoPin = 20;
6  long duration;
7  double distanceCm;
8
9  const int wheel_inB = 27;
10 const int wheel_inA = 29;
11 const int wheel_inC = 31;
12 const int wheel_inD = 33;
13 const int wheel_left = 9;
14 const int wheel_right = 8;
15
16 float Kp = 5;
17 float Ki = 0.003;
18 float Kd = 0;
19 float error, errorLast, errorInt;
20
21 double now;
22 double LastTime = 0;
23 ros::NodeHandle nh;
24
25 float calcPID(float angle){
26     float errorDiff;
27     float output;
28     double dt=(double)(now-LastTime);
29
30     error = 0 - angle;
31     errorInt += error * dt;
32     errorDiff = (error - errorLast)/dt;
33     output = Kp*error + Ki*errorInt + Kd*errorDiff;
34     errorLast = error;
35     LastTime=now;
36
37     return output;
38 }
39
40 void messageCb( const std_msgs::Float32 angle_msg){
41     now=millis()*0.001;
42     float ctrl = calcPID((float)(angle_msg.data));
43     float ff = 70;
44     motor_speed(ff-ctrl,ff+ctrl);
45 }
46
47 int pwm_max = 200;
48
```

```

49 void motor_speed(int spdL,int spdR){
50     spdR = -spdR;
51     if(spdL < 0){
52         analogWrite(wheel_left,constrain(-spdL, 0, pwm_max));
53         digitalWrite(wheel_inA,0);
54         digitalWrite(wheel_inB,1);
55     }else{
56         analogWrite(wheel_left,constrain(spdL, 0, pwm_max));
57         digitalWrite(wheel_inA,1);
58         digitalWrite(wheel_inB,0);
59     }
60     if(spdR < 0){
61         analogWrite(wheel_right,constrain(-spdR, 0, pwm_max));
62         digitalWrite(wheel_inC,0);
63         digitalWrite(wheel_inD,1);
64     }else{
65         analogWrite(wheel_right,constrain(spdR, 0, pwm_max));
66         digitalWrite(wheel_inC,1);
67         digitalWrite(wheel_inD,0);
68     }
69 }
70
71 ros::Subscriber<std_msgs::Float32> sub("angle", messageCb );

```

```

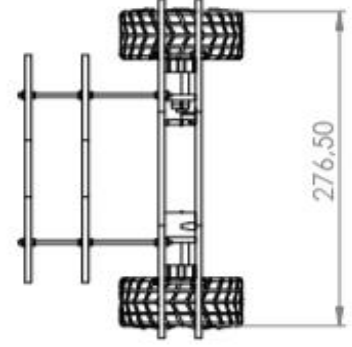
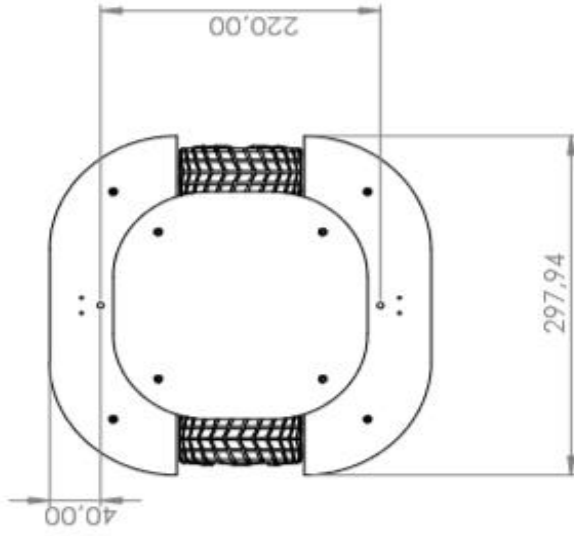
72
73 void setup() {
74
75     pinMode(wheel_inA,OUTPUT);
76     pinMode(wheel_inB,OUTPUT);
77     pinMode(wheel_inC,OUTPUT);
78     pinMode(wheel_inD,OUTPUT);
79     pinMode(trigPin, OUTPUT);
80     pinMode(echoPin, INPUT);
81
82
83     nh.initNode();
84     nh.subscribe(sub);
85
86 }
87
88 void loop() {
89
90     nh.spinOnce();
91     delay(10);
92
93 }

```

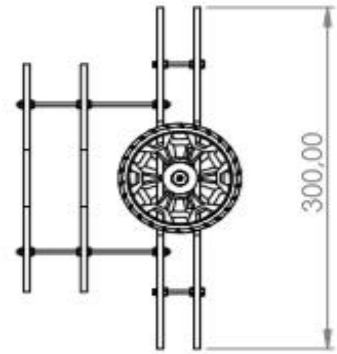
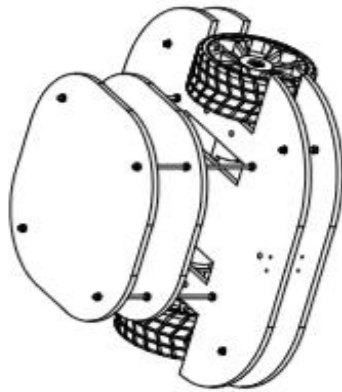




## EK-5



Tekerlik Ölçüleri:  
85mm - 38mm



## **KAYNAKLAR**

<http://dusunenrobot.com/ros-robot-operating-system-nedir/> --(ROS Nedir)

<https://wmaraci.com/nedir/cplusplus> --(C++)

<http://mesutpiskin.com/blog/opencv-nedir.html> --(OpenCV)

<https://polen.itu.edu.tr/bitstream/11527/5250/1/7811.pdf> ---(Diferansiyel Tekel Modeli)