AKILLI SİSTEMLERİN TASARIMI ÖDEV RAPORU

Oğuzhan Dilbaz	**235129030**
Eren Partal	**235129027**
Samet Yüksel	**235129028**
Turgay Güven	**235103029**
Alpha Hadji Mansare	**235103010**

1. **Grup**

Kocaeli Üniversitesi
Fen Bilimleri Enstütüsü

İçindekiler

Özet

Bu çalışmada RPİ-bot ve ESP32-bot layka prototipleri kullanılarak nesne takibi ve engellerden kaçınma uygulamaları gerçekleştirilmiştir. Nesne takibi uygulaması RPİ-bot prototip robotu ile gerçekleştirilmiştir. Protorip ropot üzerinde bulunan kamare kullanılarak beyaz bir nesne takip edilmiştir. Engellerden kaçınma uygulaması ise ESP32-bot prototip robotu kullanılarak gerçekleştirilmiştir. Bu uygulamada ESP32-bot üzerine yerleştirlene iki adet ultrasonik ses algılayıcı sensörler kullanılarak engeller tespit edilerek, tespit edilen engellerden kaçınılmıştır.

1-Giriş

Ödev projesin de önceden belirlenmiş beyaz bir nesneyi takip eden ve önüne çıkan engellerden kaçınan iki farklı prototip robot geliştirilmiştir. Nesne takibi uygulamsında ilk olarak üzerinde bulunan kameredan beyaz bir nesne tespit edilerek nesneye olan referans uzaklığa ve nesnenin konumuna göre motorlara güç vererek nesnenin takip edilmesi sağlanmıştır. Bu uygulama RPİ-bot layka prototip robotu python dili kullanılarak gerçekleştirilmiştir.

Ödev çalışmamızdaki ikinci uygulamamız ise önüne çıkan engellere göre göre yönünü değiştirin robot uygulamsı geliştirmek. Bu çalışmada ultrasonik sensörler yardımı ile robot önündeki engelleri tespit edilerek yönünü değiştirmektedir.

2-ESP32-bot İle Engellerden Kaçınan Robot Uygulaması

ESP32-bot layka prototip robotun ön tarafına 30derece açıyla yerleştirilen iki adet HC-SR04 ses algılayıcı sensör yardımıyla robotun önündeki engeller tespit edilmiştir. Ses algılayıcı sensörün trig pinine bir sinyal gönderilir ve sensörün echo pinine gelen sinyal okunarak nesneye olan mesafe ölçülerek yeri tespit edilir. Robot önünde bulunan engeli tespit ettikten sonra yönünü değiştirerek ilerlemeye çalışır. Önüne engel çıkması durumunda işlem tekrarlanır.

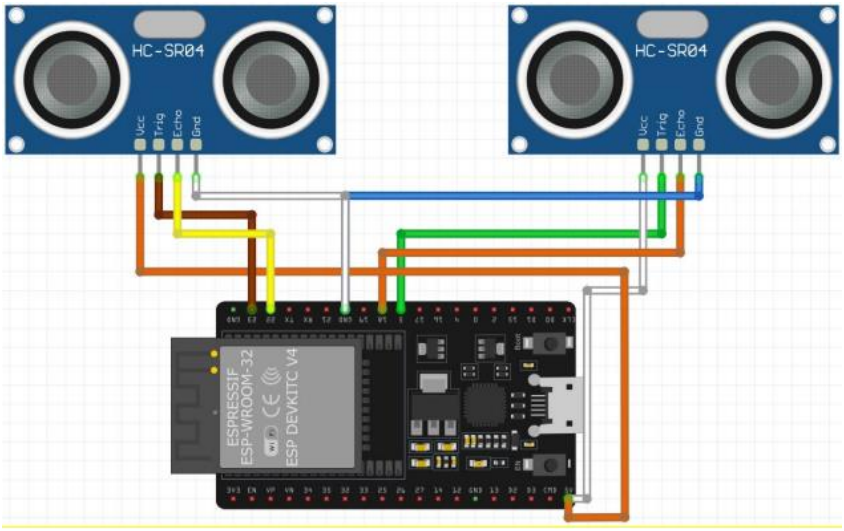
2.1-HC-SR04 ile ESP32 Bağlantısı

HC-SR04 ses algılıyacı sensörün üzerinde trig, echo, gnd ve vcc pinleri bulunmaktadır. Vcc pini sensörün beslemesi için kullanılır, Gnd pini topraklama için kullnılmaktadır. Echo ve trig pinleri ise sensörün işlevini yerine getiren pinlerdir. Sensör Trig pinine gelen

sinyal ile ses dalgası yollayarak echo pini ile dönen sinyalleri dinler. Robotumuzda iki adet sensör kullanılmış olup esp32 ile bağlantı tablosu ve şeması aşağıda verilmiştir.

HC-SR04	ESP32
Trig (1.Sensör)	G5
Echo (1.Sensör	G18
Trig (2.Sensör)	G23
Echo (2. Sensör)	G22
Vcc (1. Sensör) Vcc (2. Sensör)	5V
Gnd (1.Sensör) Gnd (2.Sensör)	Gnd

Tablo-1: Sensör Esp32 pin bağlantıları



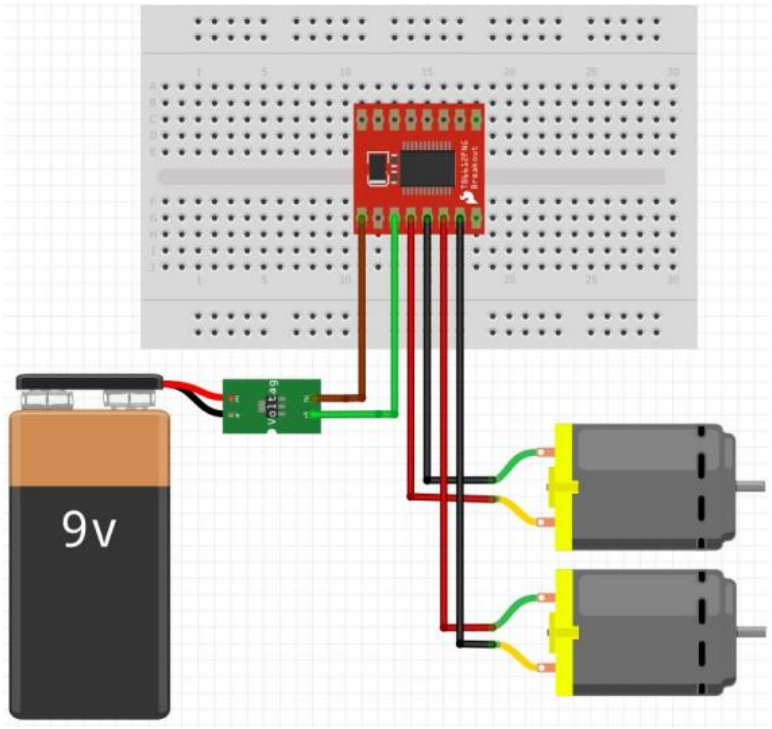
Şekil 1: Sensör Esp32 bağlantı şeması

2.2 Motor Sürücü Bağlantısı

Uygulamamızda robot üzerindeki sensörlerden aldığı bilgiye göre yönünü değiştirerek hareket eden bir robot uygulaması gerçekleştirilmiştir. Robotun hareketini üzerindeki iki tekerleğe bağlı iki dc motor sayesinde gerçekleştirmektedir. Motorlar PWM sinyali gönderilerek TB713A2 motor sürücüsü kullanılarak sürülmüştür.

2.2.1 Motor Sürücüsü ile Motor Bağlantısı

Robot üzerindeki iki motora tek motor sürücü bağlanmıştır. Sürücünün VM pini pilin + ucu ile beslenip GND pini – uca bağlanmıştır. A01 ve B01 pinleri sırası ile Motor A ve Motor B nin + ucuna A02 ve B02 ise sırası ile Motor A ve Motor B nin – ucuna bağlanmıştır. Aşağıda motorlar ile sürücü arasındaki bağlantıyı gösterir şema ile bağlantı tablosu verilmiştir.



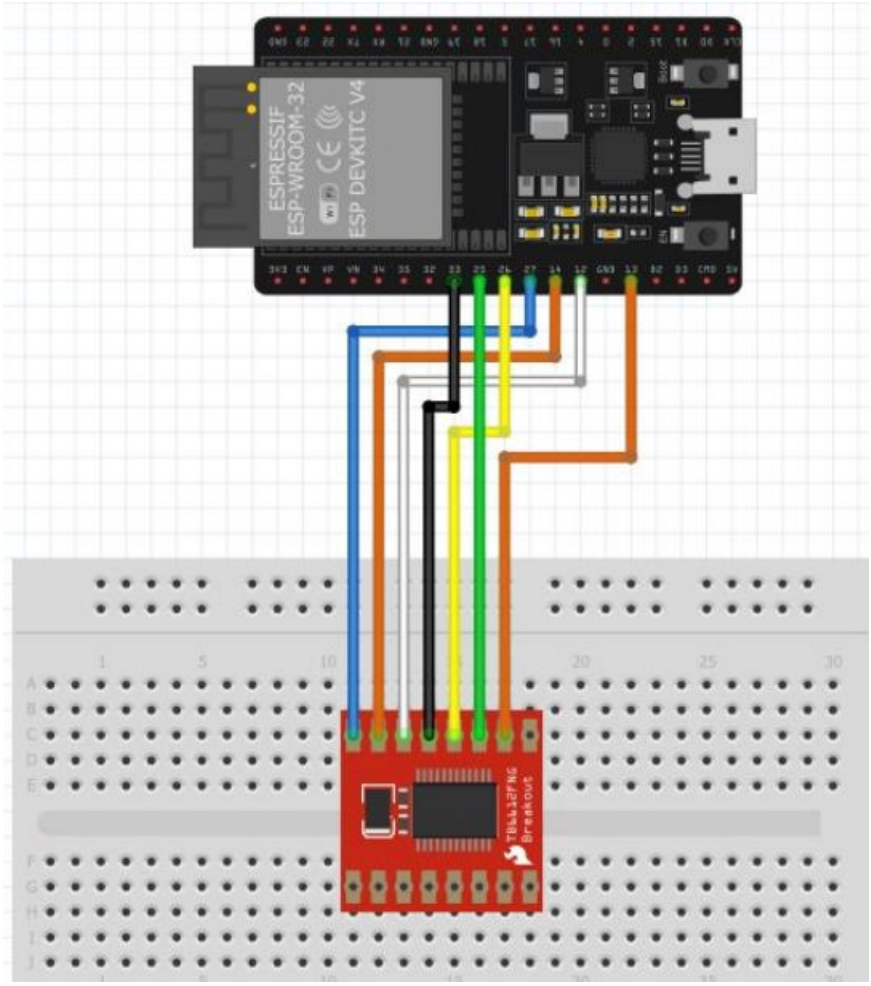
Şekil 2: Motor ile Motor Sürücüsü Arasındaki Bağlantı Şeması

Motor Sürücüsü	Motor / Pil
VM	EXT Güç (V)
GND	EXT Güç GND
A01	Motor A (+)
A02	Motor A(-)
B01	Motor B(+)
B02	Motor B(-)

Tablo 2: Motor Sürücü ile Motor/Pil Pin Bağlantı Tablosu

2.2.2 Motor Sürücüsü ile ESP32 Bağlantısı

Motor sürücüsü motorlar ile ESP32 arasına motorları kontrol edebilmek için bağlanır. Yukarıda sürücü ile motor arasındaki bağlantıdan bahsetiştirk. Motorlar PWM sinyali kullanılarak hareket ettiriliyor ve her iki motor için farklı PWM sinyali kullanılılıyor. PWM sinyali motorlara iletmek için ESP32 nin G27 ve G13 pinleri sırası ile motor sürücünün PWMA ve PWMB kanallarına bağlanmıştır. Birinci motor için AIN1 ve AIN2 motor sürücü pinleri ESP32 deki sırası ile G12 ve G14 pinlerine BIN1 ve BIN2 motor sürücü pinleri ise ESP32 deki G26 ve G25 pinlerine sırası ile bağlanmıştır. Motorların çalışıp çalışmayacağını kontrol etmeye yarayan sürücü üzerindeki STBY pini ise ESP32 deki G33 pini ile bağlanmıştır. Aşağıda motor sürücü ile ESP arasındaki bağlantı şeması ile bağlantı tablosu verilmiştir.



Şekil 3: Motor Sürücü ESP32 Arasındaki Bağlantı Şeması

Motor Sürücü	ESP32
PWMA	G27
PWMB	G13
AIN1	G12
AIN2	G14
BIN1	G26
BIN2	G25
STBY	G33

Tablo 3: Motor Sürücü ile ESP32 Arasındaki Pin Bağlantıları

2.3 ESP32 Uygulama Kodları

Ödevde istenilen robot üzerindeki sensörlerden gelen sinyaller ile önündeki engeli algılayan ve yönünü değiştirerek hareket etmeye çalışan bir robot tasarlamak. Bu uygulama yapılırken ESP32 kartımız C++ dili kullanılarak programlanmıştır.

2.3.1 ESP32 Pin Tanımlaması

İlk olarak ESP32 kartının pinlerinin tanımlaması yapılmıştır. Pin tanımlaması yukarıdaki bölümlerde anlatıldığı gibi pinlerin hangi işlevleri yerine getireceğini belirlemek için yapılır. Pin tanımlaması için define komutu kullanılır.

```
#include "Arduino.h"
```

Sensörlerin bağlanacağı pinlerin tanımlanması;

```
#define echoPinR 18
#define trigPinR 5
#define echoPinL 22
#define trigPinL 23
```

Motor sürücüsünün bağlandığı pinlerin tanımlanması;

```
#define MotorR1 14
#define MotorR2 12
```

```
#define MotorL1 26
#define MotorL2 25
```

PWM sinyali için pin tanımlaması;

```
#define MotorA 27
#define MotorB 13
#define stby 33
```

2.3.2 Değişken ve Sabitlerin Tanımlanması

Bu bölümde uygulamada kullanılacak olan çeşitli değişkenler ve sabitler tanımlanmıştır.

8 bitlil PWM sinyalinin motorlara tam hız verebilmesi için max değeri 255 ve yarım yol verebilmesi içinde 127 değeri ilgili değişkenlere atanmıştır.

```
#define fullSpeedR 255
#define halfSpeedR 127
#define fullSpeedL 255
#define halfSpeedL 127
```

Robot ünündeki engeli sensörlerinden gönderdiği ultrasonik sinyaller aracılığı ile algılamaktadır. Robotun engele olan uzaklığının bulunabilmesi için ses sinyalinin havadaki hızına ihtiyaç vardır. Bunun içinde aşağıdaki sabit 0.034 belirlenmişitr.

```
#define SOUND_VELOCITY 0.034
```

Ultrasonik sensörle gönderilen ses yankısının engele çarpıp geldiği sürelere durationR ve durationL değişkenlerinde tutulmuştur. distanceCmR ve distanceCmL değişkenleri ise engelle olan mesafenin (cm cinsinden) tutulduğu değişkenlerdir.

```
long durationR;
long durationL;
float distanceCmR;
float distanceCmL;
bool firstCall = false;
```

PWM kanlları ile frekans ve PWM sinyalinin çözünürlüğü aşağıdaki sabitlerde belirlenmiştir.

```
const int pwmaChannel = 0;
const int pwmbChannel = 1;
const int freq = 30000;
const int resolution = 8;

static unsigned long previousMillis = 0;
```

2.3.3 Setup Fonksiyonu ile ESP32 nin Başlatılması

Oluşturulan Setup fonksiynu ile ESP32 nin input ve output pinleri Pinmode() ile belirlenip PWM kanalları ile sinyal frekansları belirlenmiştir. Serial.begin(115200) bu kod ile seri iletişim bağlantısı kurulur ve başlatılır. Setup fonksiyonuna ait kodlar aşağıda verilmiştir.

```
void setup() {
  //pinlerin atamaları yapıldı
  Serial.begin(115200);

  pinMode(echoPinR, INPUT);
  pinMode(trigPinR, OUTPUT);
  pinMode(echoPinL, INPUT);
  pinMode(trigPinL, OUTPUT);
  pinMode(MotorR1, OUTPUT);
  pinMode(MotorR2, OUTPUT);
  pinMode(MotorA, OUTPUT);
  pinMode(MotorL1, OUTPUT);
  pinMode(MotorL2, OUTPUT);
  pinMode(MotorB, OUTPUT);
  pinMode(stby, OUTPUT);

  ledcSetup(pwmaChannel, freq, resolution);
  ledcSetup(pwmbChannel, freq, resolution);
  // attach the channel to the GPIO to be controlled
```

```
  ledcAttachPin(MotorA, pwmaChannel);
  ledcAttachPin(MotorB, pwmbChannel);
}
```

2.3.4 Motor Fonksiyonları

Motorların kontrolü için iki fonksiyon oluşturulmuştur. Fonksiyonlardan biri motorların tam güç ile diğeri ise motorların yarım güç ile hareket etmesini sağlamaktadır. Motorların tam güç ile hareket etmesi için tamyollleri fonksiyonu yarım güç ile hareket etmesi için yarimyollleri fonksiyonları oluşturulmuştur. Bu fonksiyonlarda motorlara gönderilen PWM sinyallerinin değerleri değeştirilerek motorların tam güç ve yarım güç ile hareket etmeleri sağlanmıştır.

```
void tamyollleri() {
  digitalWrite(stby, HIGH); // Motorları etkinleştir
  ledcWrite(pwmaChannel, fullSpeedR);
  ledcWrite(pwmbChannel, fullSpeedL);

  digitalWrite(MotorR1, HIGH);
  digitalWrite(MotorR2, LOW);

  digitalWrite(MotorL1, HIGH);
  digitalWrite(MotorL2, LOW);

  Serial.print("TAM HIZ İLERİ GİDİYOR !!!!!");
}

void yarimyollleri() {
  digitalWrite(stby, HIGH); // Motorları etkinleştir
  ledcWrite(pwmaChannel, halfSpeedR);
  ledcWrite(pwmbChannel, halfSpeedL);
  digitalWrite(MotorR1, HIGH);
  digitalWrite(MotorR2, LOW);

  digitalWrite(MotorL1, HIGH);
  digitalWrite(MotorL2, LOW);
```

```
  Serial.print("YARIM HIZ İLERİ GİDİYOR !!!!!");
}
```

Robotun sağa dönüşü için sag() fonksiyonu sola dönüş için sol() fonksiyonları oluşturulmuştur. Roboton bu işlevi yerine getirebilmesi hangi yöne dönecekse o yöndeki tekerleğe bağlı motorların yarım hız ile geri diğer tekerleğe bağlı motorun ise yarım hız ile ileri hareket ettirilmesi gerekmektedir. Sag ve sol fonksiyon kodları aşağıda verilmiştir.

```
void sag() { // Robotun sağa dönme hareketi için fonksiyon tanımlıyoruz.

  digitalWrite(stby, HIGH); // Motorları etkinleştir

  ledcWrite(pwmaChannel, halfSpeedR);
  ledcWrite(pwmbChannel, halfSpeedL);

  // sağa dönüş için sağ motor geri
  digitalWrite(MotorR1, LOW);
  digitalWrite(MotorR2, HIGH);
  // sağa dönüş için sol motor ileri
  digitalWrite(MotorL1, HIGH);
  digitalWrite(MotorL2, LOW);

  Serial.print("SAĞA GİDİYOR !!!!!");
}

void sol() {
  digitalWrite(stby, HIGH); // Motorları etkinleştir
  ledcWrite(pwmaChannel, halfSpeedR);
  ledcWrite(pwmbChannel, halfSpeedL);

  // sola dönüş için sağ motor ileri
  digitalWrite(MotorR1, HIGH);
  digitalWrite(MotorR2, LOW);
  //sola dönüş için sol motor geri
```

```
  digitalWrite(MotorL1, LOW);
  digitalWrite(MotorL2, HIGH);

  Serial.print("SOLA GİDİYOR !!!!!");
}
```

Robotun geri gidebilmesi için geri() fonksiyonu tanımlanmıştır. Her iki tekerleğin geri dönebilmesi için motorların + uclarının bağlandığı A01 ve B01 pinleri LOW yani 0 – uçların bağlandığı A02 ve B02 pinleri HIGH yani 1 olarak belirlenmiştir.

```
void geri() { // Robotun geri yönde hareketi için fonksiyon tanımlıyoruz.
  digitalWrite(stby, HIGH); // Motorları etkinleştir

  ledcWrite(pwmaChannel, halfSpeedR);
  ledcWrite(pwmbChannel, halfSpeedL);

  // iki motor da geri
  digitalWrite(MotorR1, LOW);
  digitalWrite(MotorR2, HIGH);

  digitalWrite(MotorL1, LOW);
  digitalWrite(MotorL2, HIGH);
  Serial.print("GERİ GİDİYOR !!!!!");
}
```

Dur fonksiyonumuz motorların hareketsiz kalmasını sağlamak için kullandığımız fonksiyondur. Bu fonksiyon ile motorlara değeri 0 olan PWM sinyali gönderilir.

```
void dur() {
  digitalWrite(stby, LOW); // Motorları etkinleştir

  // tüm motorlar ve pwm değerleri sıfır olarak değiştirildi.
  ledcWrite(pwmaChannel, 0);
  ledcWrite(pwmbChannel, 0);
  //analogWrite(MotorA, 0);
  //analogWrite(MotorB, 0);
```

```
  digitalWrite(MotorR1, LOW);
  digitalWrite(MotorR2, LOW);
  digitalWrite(MotorL1, LOW);
  digitalWrite(MotorL2, LOW);
  Serial.print("DURDU!!!!!");
}
```

2.3.5 Mesafe Fonksiyonu

Mesafe fonksiyonumuz ESP32 ye bağlı olan sensörlerden verileri okuyup karşısındaki engelle robot arasındaki mesafeyi hesaplayan fonksiyondur. Sensörden gönderilen ses sinyalinin engelden yansıma süresi bir değişkene durationR = pulseIn(echoPinR, HIGH) kodu ile kaydedilir (Her iki sensör içib ayrı). Kaydedilen bu süre sesin havadaki yayılım hızının önceden kaydedildiği sabit ile çarpılarak 2 ye bölünür. distanceCmR = durationR * SOUND_VELOCITY / 2 yazılan kod ile engele olan mesafe hesaplanıp bir değişkene kaydedilir.

```
void mesafe() {
  // Clears the trigPin
  digitalWrite(trigPinR, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinR, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinR, LOW);

  durationR = pulseIn(echoPinR, HIGH);
  distanceCmR = durationR * SOUND_VELOCITY / 2;

  digitalWrite(trigPinL, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinL, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinL, LOW);

  durationL = pulseIn(echoPinL, HIGH);
```

```
  distanceCmL = durationL * SOUND_VELOCITY / 2;

  Serial.print("distanceCmR:");
  Serial.println(distanceCmR);

  Serial.print("distanceCmL:");
  Serial.println(distanceCmL);
}
```

2.3.6 Çalışma Fonksiyonu

Robotun hareketinin tanımlanan fonksiyonlar kullanılarak gerçekleştirildiği fonksiyondur. Robot hareketini sensörlerden algılanan sinyal ile mesafe fonksiyonu kullanılarak hesaplanan değer ile referans alınan değerler karşılaştırılarak gerçekleştirmektedir.

Fonksiyonu parça parça ele aldığımızda ilk olarak 200 milisaniye de bir mesafe fonksiyonu çalıştırılarak her iki sensör yardımıyla robotun karşısındaki engele olan mesafesi ölçülür.

```
while(millis() - previousMillis >= 200) {
    ;
  }
  mesafe();
```

Alınan mesafe değeri ile referans değerleri karşılaştırlır. İlk karşılaştırma ile robotun karşısındaki engelden 50 cm den uzakta olması durumunda her iki motor için tamyolileri fonksiyonu çalıştırılır.

```
if ((distanceCmR >= 50) && (distanceCmL >= 50)) { //  2 si de 50+ ise tam hız ileri git
    // iki motor tam yol ileri
    tamyolIleri();
  }
```

İkinci koşulumuzda sağ sensör ile ölçülen mesafenin 20 cm den az olması durumunda robot sağa döner ve yarımyolileri fonksiyonu çalıştırılarak robot ileri hareket ettirilir.

```
else if ((5 <= distanceCmL && distanceCmL <= 20)) {
   // sol sensör 20cmden daha yakında nesne algıladıysa sağa dön
   previousMillis = millis();
   while(millis() - previousMillis <= 100){
    //dönüş işleminde motorların zarar görmemesi için her dönüşte önce motorları durdur
     dur();
   }
   previousMillis = millis();
    while(millis() - previousMillis <= 300) {
     //sağa dön
     sag();
   }
   previousMillis = millis();

   while(millis() - previousMillis <= 50) {
     dur();
     // dönüş işlemlerinde bir taraf ileri giderken diğer taraf geri geldiği için yine motorları komple durdur
   }
   yarimyollleri();
   // dönüş işlemi bittikten sonra yarım hız ile yola devam et
 }
```

Üçüncü durumda ise bu kez sol sönsörden okunan mesafe 20 cm altında ise önce robot sola döner ve ardından yarım yol ileri gider.

```
else if ((5 <= distanceCmR && distanceCmR <= 20)) {
   // sağ sensör 20cmden daha yakında nesne algıladıysa sola dön
   while(millis() - previousMillis <= 100) {
     dur();
     previousMillis = millis();
   }
   while(millis() - previousMillis <= 300) {
     sol();
     previousMillis = millis();
```

```
  }
  while(millis() - previousMillis <= 50) {
   dur();
   previousMillis = millis();
  }
  yarimyolIleri();

 }
```

Son durumda ise robotun önün de 5 cm den az bir mesafede cisim algılanması durumunda önce dur fonksiyonu ile robot durur. Ardından sag fonksiyon ile sağa dönerek yarımyolileri fonksiyonu ile ileri gitmesi sağlanır.

```
else if (distanceCmR < 5 || distanceCmL < 5) {
  // ani nesne çıkması durumunda dur-geri git- dur- sağa dön- dur- yarım hız ilerle
  while(millis() - previousMillis <= 100) {
   dur();
   previousMillis = millis();
  }
  previousMillis = millis();
  while (millis() - previousMillis <= 500) {
   geri();

  }
  previousMillis = millis();
  while(millis() - previousMillis <= 100) {
   dur();

  }
  previousMillis = millis();
  while(millis() - previousMillis <= 200) {
   sag();
  }
  previousMillis = millis();
  while(millis() - previousMillis <= 100) {
   dur();
```

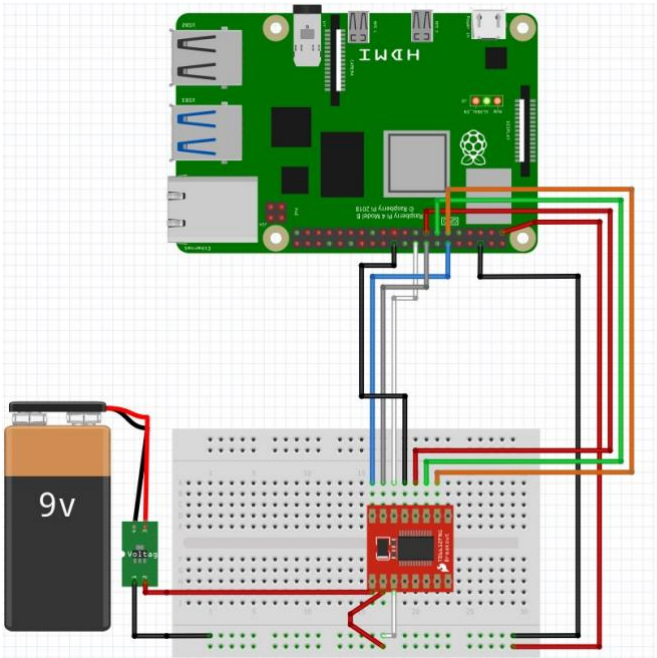
```
  }
yarimyollleri();
}
```

3 RPİ-bot ile Nesne Takibi Uygulaması

Bu çalışmada RPİ-bot protatip robotu ile beyaz bir nesneyi takip eden bir uygulama gerçekleştirilmiştir. Robotun üzerinde bulunan kameradan alınan görüntü işlenerek cismin roboto göre konumu tespit edilerek cisim yönünde hareket gerçekleştirilmiştir. Bu uygulama gerçekşeştirlirken pyhon dili kullanılmıştır.

3.1 RPİ ile Motor Sürücüsü Bağlantısı

Motorları kontrol etmek için kullanılan PWM sinyallerininin RPİ den gönderilebilmesi için RPİ nin 11 ve 12 inci pinleri sırası ile motor sürücünün PWMB ve PWMA pinlerine bağlanmıştır. Motor sürücünün AI1 ve AI2 pinleri birinci motorların girişleri olarak RPİ nin sırası ile 16 ve 18 inci pinlerine bağlanmıştır. İkinci motorun girişleri olan BI1 ve BI2 ucları RPİ nin 15 ve 13 üncü pinlerine bağlanmıştır.



VM	EXT POWER(V)
GND	EXT POWER(GND)
GND	Raspi(GND)
VCC	Raspi(3.3v)
A01	MotorA(+)
A02	MotorA(-)
B01	MotorB(+)
B02	MotorB(-)
PWMA	Raspi (Pin 12)
AI1	Raspi (Pin 16)
AI2	Raspi (Pin 18)
STBY	Raspi (Pin 22)
BI1	Raspi (Pin 15)
BI2	Raspi (Pin 13)
PWMB	Raspi (Pin 11)

Şekil 4: RPİ ile Motor Sürücü Bağlantı Şeması ve Pin Bağlantıları

3.2 RPİ Uygalama Kodları

Ödevde istenen uygulama RPİ-bot ile önünde algıladığı beyaz renkte bir cismi tespit ederek onu takip etmesidir. Bu uygulama gerçekleştirlirken ilk olarak RPİ ye bağlı kamereadan takip edilecek olan beyaz nesne tespit edilerek referans uzaklık belirlenmiştir. RPİ-bot refans alınan uzaklık değeri ve cismin kendisine göre olan kunumuna göre kendi yönünü belirleyerek cismin hareketi süresince hareket ederek cisim ile arasındaki referans uzaklığı korur.

3.2.1 Kullanılan Kütüphaneler

Uygulamanın geliştirilmesi sırasında Open Cv, İmutils, Math ve RPİ.GPU kütüphaneleri kullanılmıştır. Kütüphanelerin uygulamaya eklenişini gösteren kodlar aşağıdadır.

```
import imutils
import cv2
import math
from time import sleep
import RPi.GPIO as GPIO
```

Uygulamada kameradan görüntü algılamak ve üzerinde işlem yapabilmek için open cv ve open cv nin yardımcı kütüphanesi olan imutils kullanılmıştır. Matematiksel hesaplamaların yapılabilmesi için Math kütüphanesi ve RPİ pinlerinin kontrolü için de RPİ.GPIO kütüphanesi kullanılmıştır.

3.2.2 RPİ Pin Atamaları

Bu bölümde motor sürücüsünün kontrol edilmesi için kullanılan RPİ pinleri program üzerinde atamaları yapılmıştır. Atamalar RPI üzerindeki fiziksel pin numaralarına göre yapılacağı ve uyurıların algılanmıyacağı belirtilmiştir.

```
GPIO.setmode(GPIO.BOARD)      # Set GPIO mode to BCM
GPIO.setwarnings(False);
# Setup Pins for motor controller
GPIO.setup(12, GPIO.OUT)    # PWMA
GPIO.setup(16, GPIO.OUT)    # AIN2
GPIO.setup(18, GPIO.OUT)    # AIN1
GPIO.setup(22, GPIO.OUT)    # STBY
GPIO.setup(15, GPIO.OUT)    # BIN1
GPIO.setup(13, GPIO.OUT)    # BIN2
GPIO.setup(11, GPIO.OUT)    # PWMB
```

3.2.2 RPİ ile PWM Sinyali Oluşturulması

Motorların konrol edilmesi için iki farklı PWM sinyali kullanılmaktadır. Bu sinyaller için kodumuzda iki adet PWM nesnesi oluşturulmuştur.

```
pwma = GPIO.PWM(12, pwmFreq)    # pin 18 to PWM
pwmb = GPIO.PWM(11, pwmFreq)    # pin 13 to PWM
pwma.start(100)
pwmb.start(100)
```

3.2.3 Motor Kontrol Fonksiyonları

Motor kontrolü için ilk olarak moturun ileri, geri, sağa dönme ve sola dönme hareketlerini gerçekleştireceği dört tane fonksiton tanımlanmıştır. Bu fonksiyonlar parametre olarak motorun hızını belirten spd parametresini almaktadır. Fonksiyonlar içinde birde motoru hareketlendirmek için kullanılan runMotor adında alt fonksiyon vardır.

```
def forward(spd):
    print("ileri")
    runMotor(0, spd, 0)
    runMotor(1, spd, 0)


def reverse(spd):
    print("geri")
    runMotor(0, spd, 1)
    runMotor(1, spd, 1)


def turnLeft(spd):
    print("sol")
    runMotor(0, spd, 0)
    runMotor(1, spd, 1)


def turnRight(spd):
    print("sag")
    runMotor(0, spd, 1)
    runMotor(1, spd, 0)
```

Alt fonksiyon runMotor fonksiyonumuz algılanan nesneye göre motorları hareketlendirmeye yarayan fonksiyondur. Fonksiyon hangi motor olduğu (0,1) motorun hızını belirten spd ve motorun ileri geri yönü (0,1) parametrelerini alır.

```
def runMotor(motor, spd, direction):
    GPIO.output(22, GPIO.HIGH);
    in1 = GPIO.HIGH
    in2 = GPIO.LOW

    if(direction == 1):
        in1 = GPIO.LOW
        in2 = GPIO.HIGH

    if(motor == 0):
        GPIO.output(18, in1)
        GPIO.output(16, in2)
        pwma.ChangeDutyCycle(spd)
    elif(motor == 1):
        GPIO.output(15, in1)
        GPIO.output(13, in2)
        pwmb.ChangeDutyCycle(spd)
```

Yukarıda kodları verilen fonksiyonda in1 ve in2 adında iki değişken tanımlanarak GPIO lojik 1 ve 0 ile tanımlanmıştır daha sonra fonksiyonun bir parametresi olan ve motorların ileri geri yönde hareketini belirleyen direction parametresinin değerine göre (1 e eşit olması durumunda) in1 ve in2 değişkenlerinin lojik değerleri değiştirilir. Direction parametresi 0 olduğunda motor ileri yönde 1 olduğunda ise geri yönde döndüğünü belirtir. Fonksiyonun yine bir parametresi olan motor (sağ veya sol motor sırası ile 1 , 0) değerine göre pwm kanalları belirlenerek GPIO nun hangi çıkış pinlerinin kullanılacağı belirlenmektedir.

Motor kontrol fonksiyonlarında son olarak motorStop fonksiyonu vardır. Bu fonksiyon motorların durdurulması içindir.

```
def motorStop():
    print("dur")
    GPIO.output(22, GPIO.LOW)
```

Yukarıdaki fonksiyonları kullanarak robota hangi yönde nasıl hareket edeceğini beşirten bir fonksiyon tanımlanmıştır. Bu fonksiyon avg (algılanan cismin konumu) ve distance (algılanan cismin uzaklığı) olmak üzere iki tane parametre alır. Kod incelendiğinde uzklığımız 60 birimin altında olması durumunda spd değişkeni yani motorun hızını

belirleyen değişken 50 olarak ayarlanarak motorun yarım hızda dönmesi 6 birimin üzerindeki durumlar için de 100 olarak ayarlanarak motorların tam hızda dönmesi sağlanmıştır. Mesafe 50 ve altında ise motorlar durdurulmuştur. Mesafenin 50 nin üzerinde olduğu durmlarda ise cismin konumu sorgulanarak oba göre robot sağa ve solo dönmesi sağlanmıştır. Aşağıda fonksiyona ait kodlar verilmiştir.

```
def decideToDirection(avg, distance):

    if distance < 60:
        spd = 50
    else:
        spd = 100
    if distance <= 50:
        motorStop()
        reverse(spd)
    else:
        if 220 <= avg and avg <= 420:
            forward(spd)
        elif avg < 220:
            turnLeft(spd)

        elif  420 < avg:
            turnRight(spd)
```

3.2.4 Kamera İle Cisim Tespiti

İlk olarak kameredan algılanan nesnenin robota olan konumu ve uzaklığını hesaplayan bir fonksyon tanımlanmıştır. İşlenecek olan görüntünün bir kopyası çıkartılır. Cismin odak uzaklığının hesaplanabilmesi için gray_width ve gray_height adında iki değişken tanımlanır ve sırasıyla görüntünün genişlik ve yüksekliği atanır. Algılanan nesnenin koordinatları alınır ve odak uzaklığı hesaplanır.

```
# My HSV Values
hsvLower = (77, 50, 132)
hsvUpper = (94, 93 ,218)


camera=cv2.VideoCapture(0)
while True:
        (grabbed, frame) = camera.read()

        frame = imutils.resize(frame, width=600)
        blurred = cv2.GaussianBlur(frame, (7, 7), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

```
        mask = cv2.inRange(hsv, hsvLower, hsvUpper)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)


        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)[-2]
        center = None
        text=""
        if len(cnts) > 0:
                bbox = cv2.boundingRect(mask)
                distance = calcDistanceByNightmare(bbox,frame)
                c = max(cnts, key=cv2.contourArea)
                ((x, y), radius) = cv2.minEnclosingCircle(c)
                M = cv2.moments(c)
                center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
                if radius > 10:
                        cv2.circle(frame, (int(x), int(y)), int(radius),
                                (0, 255, 255), 2)
                        cv2.circle(frame, center, 5, (0, 0, 255), -1)
                        print(center,radius,"\ndistance ==>> ",distance)
                        if radius > 250:
                                print("stop")
                        else:
                                if(center[0]<150):
                                        print("Left")
                                        text="Left"
                                elif(center[0]>450):
                                        print("Right")
                                        text="Right"
                                elif(radius<250):
                                        print("Front")
                                        text="Front"
                                else:
                                        print("Stop")
                        decideToDirection(center[0],distance)
                else:
                    motorStop()

        cv2.putText(frame,text,(50,100),cv2.FONT_HERSHEY_SIMPLEX,1.0,(255, 0, 0),2 )
        #cv2.imshow("Frame", frame)
        #key = cv2.waitKey(1) & 0xFF
        #if key == ord("q"):
        #        break

#camera.release()
#cv2.destroyAllWindows()
```

3.2.5 Kamera İle Mesafe Tespiti

calcDistanceByNightmare Fonksiyonunun Matematiksel Açıklaması
Amaç:

Bu fonksiyon, bilinen yatay görüş açısına (HFOV) sahip bir kamera tarafından çekilen bir görüntüdeki tespit edilen bir nesnenin tahmini uzaklığını hesaplar.

Varsayımlar:

İlgi çekici nesne kabaca dikdörtgendir ve görüntü yüksekliğinin önemli bir kısmını doldurur.
Nesnenin gerçek dünyadaki yüksekliği bilinmektedir (KNOWN_ROI_MM).
Adımlar:

Görüntü Boyutları:

width: Görüntünün genişliği piksel cinsinden (görüntünün şeklinden alınır).
height: Görüntünün yüksekliği piksel cinsinden (görüntünün şeklinden alınır).
Nesne Tespiti:

x, y, w, h: Tespit edilen nesnenin sınırlayıcı kutusunun koordinatları ve boyutları (ayrı bir nesne algılama işleminden elde edildiği varsayılır).
Odak Uzaklığı Hesaplaması:

DFOV_DEGREES: Kameranın yatay görüş açısı dereceler cinsinden (sabit bir değer olduğu varsayılır).

focal_value: Aşağıdaki formül kullanılarak hesaplanan kameranın odak uzaklığı:

focal_value = (gray_height / 2) / tan(radians(DFOV_DEGREES / 2))
Açıklama:

Görüntü yüksekliği (gray_height) 2'ye bölünür çünkü görüş açısı (FOV) genellikle manzara yönünde olduğunda görüntü yüksekliğinin yarısına uygulanır.

tan(radians(DFOV_DEGREES / 2)), HFOV'un yarısının radyan cinsinden tangantını hesaplar. Bu değer, nesnenin görüntüdeki boyutunu gerçek dünyadaki açısal boyutuna ilişkilendirir.
Görüntü yüksekliğini tanganta bölmek bize piksel cinsinden odak uzaklığını verir. Bu, kameranın görüntü sensörü ile lensin odak noktası arasındaki mesafeyi temsil eder.
Mesafe Tahmini:

KNOWN_ROI_MM: İlgi çekici nesnenin bilinen yüksekliği milimetre cinsinden.

dist: Nesnenin kameradan milimetre cinsinden tahmini uzaklığı, aşağıdaki formül kullanılarak hesaplanır:

dist = KNOWN_ROI_MM * focal_value / h
Açıklama:

Nesnenin bilinen yüksekliği (KNOWN_ROI_MM) odak uzaklığı (focal_value) ile çarpılır ve ardından nesnenin piksel cinsinden yüksekliğine (h) bölünür. Bu oran, nesnenin görüntüdeki boyutunu gerçek dünyadaki bilinen boyutuna ilişkilendirir ve bize nesnenin uzaklığı hakkında bir tahmin verir.
Birim Dönüşümü:

dist_in: Santimetre cinsinden tahmini mesafe, aşağıdaki dönüştürme faktörleri kullanılarak hesaplanır:

dist_in = dist / 25.4 (milimetreden ince)
dist_in = dist_in * 2.54 (inçten santimetreye)
Açıklama:

Milimetre cinsinden mesafe (dist) ilk önce 25.4'e bölünerek ince dönüştürülür. Ardından, 2.54 ile çarpılarak santimetreye dönüştürülür.
Metin Yerleşimi:

Tahmini mesafe santimetre cinsinden (round(dist_in)) bir dizeye dönüştürülür ve görselleştirme amaçlı olarak OpenCV'nin cv2.putText fonksiyonu kullanılarak görüntüye yerleştirilir.
Özetle:

Bu fonksiyon, kameranın HFOV'unu, nesnenin bilinen yüksekliğini ve görüntü boyutlarını kullanarak nesnenin uzaklığını tahmin eder. Bir tahmin sunarken, kamera eğimi, nesne yönü ve potansiyel ölçüm hataları gibi faktörler doğruluğu etkileyebilir.

```
DFOV_DEGREES = 54  # kamera fov değeri
KNOWN_ROI_MM = 110  # robot yüksekliği
def calcDistanceByNightmare(detectedObject,image):

    # get image dimensions:
    image = image
    width = image.shape[1]
    height = image.shape[0]
    x,y,w,h = detectedObject
    focal_value = (height / 2) / math.tan(math.radians(DFOV_DEGREES / 2))
    dist = KNOWN_ROI_MM * focal_value / h
    dist_in = dist / 25.4
    dist_in = dist_in * 2.54

    cv2.putText(image, 'Distance:' + str(round(dist_in)) + ' cm',
                    (5, 300), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
    return dist_in
```