



TOBB Ekonomi ve Teknoloji Üniversitesi

ELE301 — Kontrol Sistemleri

Proje Raporu

Ad Soyad 3 Ad Soyad 1 Ad Soyad 2	Ege Gökçen Kaan Seyhan Ozan Özkaya
Öğrenci Numarası 1 Öğrenci Numarası 2 Öğrenci Numarası 3	221201022 221201034 211201009
Tarih	17/12/2024

İçindekiler

Projenin Amacı ve Genel Bilgiler	3
Projede Kullanılan Elektronik Bileşenler	3
Kontrolcü Algoritmaları ve Yazılım	10
Aç-Kapa Kontrol Algoritması	15
P-Kontrol Algoritması	17
PI-Kontrol Algoritması	19
PD-Kontrol Algoritması	21
PID-Kontrol Algoritması	23
Sonuçlar	25
Kaynakça	26
Ekler	26

Projenin Amacı ve Genel Bilgiler

Bu projede, temel kontrol eylemleri olan Aç-Kapa, Oransal (P), Oransal-İntegral (PI), Oransal-Türev (PD) ve Oransal-İntegral-Türev (PID) algoritmaları kullanılarak, kullanıcı tarafından belirlenen bir kat referansını hassas bir şekilde takip eden bir asansör sisteminin tasarlanması ve gerçekleştirilmesi hedeflenmiştir. Tasarımın en önemli amaçlarından biri, teorik olarak Laplace ve zaman alanlarında kapalı formda ifade edilen matematiksel denklemleri kullanarak, referans takibi fenomeninin temel prensiplerini açıklamak ve bu prensipleri fiziksel bileşenler yardımıyla gözlemlenebilir bir hale getirmektir.

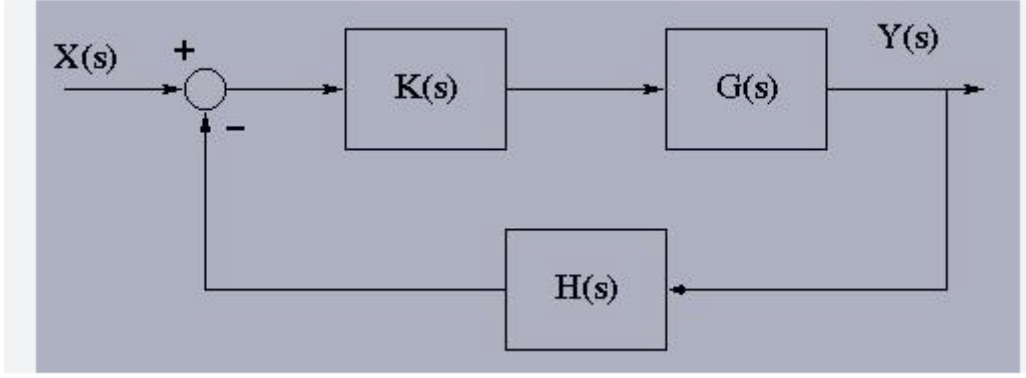
Projede yer alan asansör, kullanıcı tarafından belirlenen bir kat referansına hareket etmektedir ve bu hareket, kullanıcı tarafından seçilen kontrol algoritması doğrultusunda gerçekleştirilir. Kontrol algoritmalarının tasarımı ve uygulanması sırasında, her bir algoritmanın sistem üzerindeki etkileri teorik olarak analiz edilmiş ve bu analizler fiziksel sistemde deneysel olarak doğrulanmıştır. Bu bağlamda, kullanıcının belirlediği kontrol algoritmasının sisteme olan etkilerinin detaylı bir şekilde incelenmesi ve bu algoritma değişikliklerinin referans takibini nasıl etkilediğinin gözlemlenmesi büyük bir önem taşımaktadır.

Sonuç olarak, yapılan çalışma hem teorik analizlerin hem de fiziksel gerçeklemelerin bir arada değerlendirildiği bir tasarımı ortaya koymayı amaçlamakta ve kullanıcı seçimine bağlı olarak sistem dinamiklerinin nasıl şekillendiğini detaylı bir şekilde incelemeyi sağlamaktadır.

Projede Kullanılan Elektronik Bileşenler

Projenin teorik tasarım sürecinin gerçekleştirilebilmesi için çeşitli elektronik bileşenler kullanılmıştır.

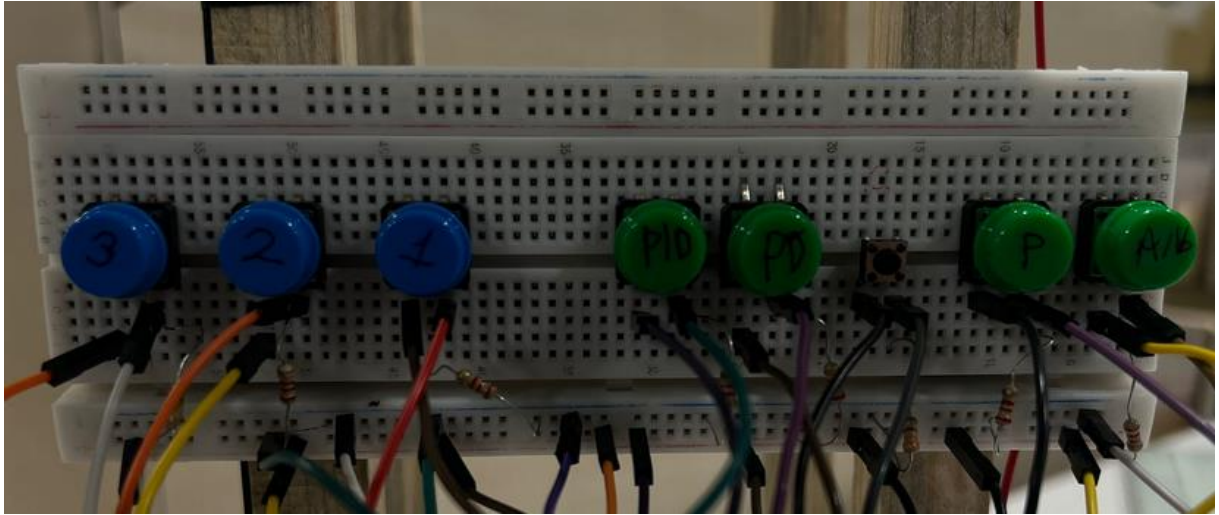
Projede yazılım dili kolaylığı sağladığından daha sonra açıklanacak olan projede kullanılacak komponent(elektronik bileşenler) ile uyumlu olduğundan ve dijital sinyallerin analizini rahat kıldığından ötürü , gömülü yazılım kartı olarak Arduino Uno tercih edilmiştir



Resim 2.1.1: Standart Kontrol Sistem Blok Diyagramı

Resim 1.1.1, projedeki asansör sisteminin içerdiği sistem bileşenlerinin zincirleme bağlantı şemasını göstermektedir.

Resim 1.1.1'deki $X(s)$ ile isimlendirilen sinyal, referansı notasyonize etmektedir. Referansın buton, potansiyometre, kumanda gibi bileşenlerle sisteme aktarılması gerektiği açıktır. Bu proje özelinde referans alım yöntemi olarak buton devresi tercih edilmiştir(yazılımsal özellikleri, kısım 2'de görece detaylı olarak açıklanacaktır).



Resim 2.1.2: Kullanıcıdan Referans Bilgisini Alan Düğme Devresi

Resim 2.1.2'deki devrede, toplam 8 adet tuş bulunmaktadır. Bu 8 adet tuşun soldaki 3 tanesi asansörün gitmesi gereken kat bilgisi referansı sinyalini içerirken sağdaki 5 tuş, kontrolcünün uygulayacağı sinyali dramatik olarak etkileyecek algoritma bilgisi referansını içerir. Devrede, olası yüksek akımlar ve

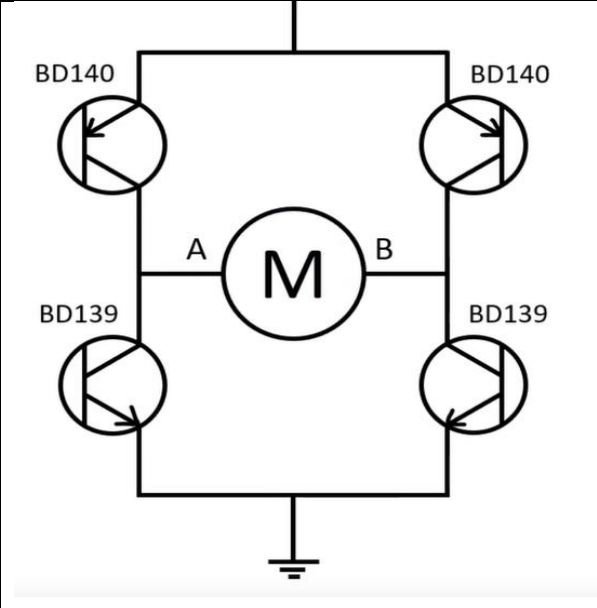
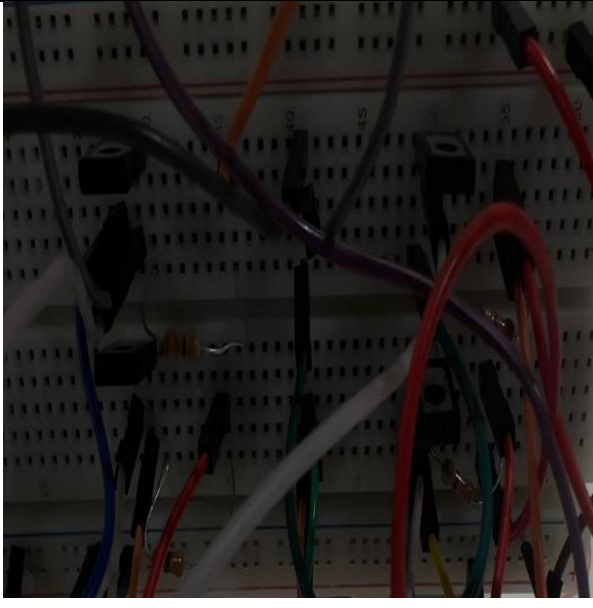
topraklama kontrolü için gerekli veri sayfası(datasheet) kaynakça [1]'de verilen $2.2\ k\Omega$ dirençler tercih edildi. Bu devrede butonların bir kapısına(pin) verilen sabit bir $V_{cc} = 5\ V$ sinyali, diğer kapısına üzerine basılmadığı sürece aktarmama özelliğinden faydalandı. Bu özellikler ile gerekli sinyal analiz ve düzeltimi önceden bahsedilen gömülü yazılım kartı olan Arduino Uno ile yapıldı. (yazılımsal içeriğe daha sonra değinilecektir). Kullanıcının basmadığı butonların aktaracağı sinyallerin egale edildiğine dikkat ediniz.



Resim 2.1.3: Redüktörlü DC Motor

Resim 2.1.1'deki sistem blok şemasında bulunan bütün bileşenlerin kontrol ettiği anahtar cihaz, Resim 2.1.3'te görülen DC motordur. DC motor, uçları arasındaki potansiyel farka göre bir açısal hızla dönme hareketi gerçekleştirmektedir ve dönme hareketinin bir ip yardımıyla asansörü yukarıya çekeceği açıktır. Bu motor blok şemasındaki $G(s)$ bloğunu temsil etmektedir. Bu projede kullanılacak DC motor, güç verimliliği açısından çalışma voltajı 6V olarak tercih edildi bunun yanında DC motor tercihini etkileyen bir diğer faktör redüktör oldu. Proje evrağında bir tasarım kısıtı verilmeyen redüktörlü motor, yükün stabilizasyonu, yüksek tork avantajı gibi parametrelerden dolayı tercih edildi. Bir sonraki paragrafta DC motoru çalıştırmak için gereken voltaj analizi ve devresinden bahsedilecektir.

Proje evrağında verilen bir tasarım kısıtı olan hazır sürücü kullanamama gerçeği, H-Bridge devresi kullanılarak giderilmiştir.

Teorik H-Bridge Şeması	Projede Kullanılacak Olan Fiziksel Devre
	

Tablo 2.1.1: H-Bridge Devresi Gösterimi

Fiziksel kurulan devrede, şematik devreden farklı olarak gerekli veri sayfası Kaynakça [1]'de verilen direnç kullanıldığı açıktır ayrıca devrede kullanılan transistörlerin veri sayfası(datasheet) Kaynakça [2]'deki gibidir. Bu direnç, istenmeyen polarma durumları ve akım kontrolü için tercih edildi. Bu devrenin 2 kullanım amacı mevcuttur. Anahtar amaç, DC motorun çalıştırılması için gerekli sinyalin uygulanması ihtiyacı iken bir diğer amaç ise basit kontrol eylem algoritmalarında gereken voltaj değerlerinin gerçekleştirilmesidir.

$$K_p \cdot e(t) \quad (1)$$

$$K_p \cdot e(t) + K_i \int e(t) dt \quad (2)$$

$$K_p \cdot e(t) + K_d \frac{d}{dt} e(t) \quad (3)$$

$$K_p \cdot e(t) + K_d \frac{d}{dt} e(t) + K_i \int e(t) dt \quad (4)$$

(1) (2) ve (3) (4) denklemleri, basit kontrol eylemlerinden sırasıyla P, PI, PD, PID kontrolcülerin zaman alanı fonksiyonlarını vermektedir. Bu fonksiyonlar, Resim 2.1.1'deki şemada $K(s)$ bloğuna karşılık gelmektedir. Bu bloğu elektronik bileşen olarak Arduino gömülü yazılım kartı temsil etmektedir. Projede kullanılan bileşenlerin ideal olmaması ve yüksek mertebeden transfer fonksiyonuna sahip cihazlar olmasından dolayı laplace uzayı yerine zaman alanında çalışılmıştır. Denklemlerde belirtilen $e(t)$ ifadesi, bu proje için kat referansı ile asansörün herhangi bir t anında yerden yüksekliğidir. $E(t)$ sinyalinin, zamanla değişen bir sinyal olduğuna dikkat edilmelidir. Zamanla değişen bu sinyallere göre uygulanacak potansiyel farkın bir pil veya DC gerilim ile sağlanamayacağı açıktır. Bu durumdan dolayı, bir sinyal modülasyonu olan “PWM” den faydalanılmıştır.

PWM (Pulse Width Modulation), bir sinyalin görev döngüsünü değiştirerek analog bir çıkışı simüle etmek veya güç kontrolü sağlamak için kullanılan bir dijital sinyal modülasyon yöntemidir. Bu projede, kullanılan gömülü yazılım kartı doğrudan bir analog sinyal oluşturmamasına karşın, bu modülasyon yöntemiyle analog potansiyel fark değerlerine dijital olarak yakınsamaktadır. Tablo 2.1.1'de verilen H-bridge devresi, motora uygulanacak potansiyel farkı güncellerken PWM sinyali devreye girmektedir. H-bridge devresi, uygulanan pwm sinyaline göre, asansörü yukarı çıkaran pozitif gerilim veya asansörü aşağı indiren negatif gerilim değerlerini aktarmaktadır.

Aç kapa kontrol eyleminde ara gerilim değerlerine ihtiyaç duyulmadığı için “PWM” sinyaline gerek duyulmayacağına dikkat ediniz.

PWM sinyalinin oluşturmumu ve h bridge devresine aktarımı bir sonraki kısımda verilmiştir.

Bu H-Bridge devresi, DC motorun hızını ve yönünü kontrol etmek amacıyla dört adet transistör (BD140-PNP ve BD139-NPN) kullanılarak tasarlanmıştır. Motorun uçları olan A ve B noktalarına bağlı transistör çiftleri, belirli kombinasyonlarla ilettime geçirilerek motorun dönüş yönü değiştirilir. Örneğin, sol üstteki BD140 ve sağ alttaki BD139 transistörleri ilettime geçtiğinde A noktası pozitif, B noktası negatif olur ve motor tek yönde döner; tersine, sağ üstteki BD140 ve sol alttaki BD139 ilettime geçirilirse motor ters yönde döner. Hız kontrolü ise seçilen transistör çiftlerine uygulanan PWM (Pulse Width Modulation) sinyali ile sağlanır; bu sinyalin görev döngüsü (duty cycle), motorun aldığı ortalama voltajı ayarlayarak dönüş hızını kontrol eder. Aynı kol üzerindeki iki transistörün aynı anda ilettime geçmesi

engellenmelidir, aksi takdirde kısa devre riski oluşur. Bu yapı, DC motorların çift yönlü ve hassas hız kontrolünü sağlayan enerji verimli bir çözüm sunar.

Projede kullanılan bir diğer bileşen ise denklem (1) (2) (3) ve (4) 'te bulunan $e(t)$ fonksiyonunun hesaplanması için kullanılan sensördür. Bu bileşen, Resim 2.1.1'de bulunan sistem şemasında $H(s)$ bloğuna karşılık gelmektedir. Mesafe ölçümü için lazer, termal gibi ölçüm faktörlerini benimseyen sensörler tercih edilebilir. Bu proje özelinde, ölçüm için ses dalgalarının gidiş-gelişi arasında geçen sürede ses hızıyla ne kadar mesafe kat edebileceği prensibine dayanan HC-SR04 Ultrasonik Sensör tercih edilmiştir.



Resim 2.1.4: Mesafe Sensörü

Mesafe sensörünün, asansör sistemine yerleştirilirken yerden yükseklik ölçümü yapılacağı için sistemin tabanına konması gerekir. Yazılımsal özelliklere bir sonraki bölümde değineceğine karşın bu bölümde, sensörün Arduino kartına, saniyede konumla ilgili yaklaşık 20 örnek yollar ve bir değişkene atanır.

Bu kısımda sonuç olarak Resim 2.1.1’de verilen bütün blokların bir elektronik bileşene karşılık geldiği açıklanmıştır. Bu bileşenler birbirlerine uygun port(kapılarla) bağlanılarak bir kontrol sistemi tamamlanmıştır. Bu sistemde anahtar görevi olan kontrolcü kartı olan Arduino, içine gömülen yazılımla gerekli referans takip ve sürekli hal hatası çözümlerini sağlamıştır.

Bu kısımda elektronik bileşenlerin haricinde mekanik tasarımdan da bahsedilmesi gerekir. Mekanik tasarım olarak 3 katlı bir asansör tercih edilmiştir. Her kat arası eşit ve 20 cm olarak belirlenmiştir. Ayrıca sensörün fiziksel varlığı gerçeğinden dolayı en alt kat olan 1. katın altına 10 cm boşluk bırakılmıştır. Toplam olarak 70 cm’lik bir yapı söz konusudur. Asansörün kabini yaklaşık olarak 10 cm yer kaplamaktadır ve bu boyut katlar arasında herhangi bir karışıklık olmaması amacıyla seçilmiştir. Kabin ağırlığı olarak yaklaşık 400 g ölçümü gerçekleştirilmiştir. Projede istenen tasarım kısıtı olan kabinin ağırlığının iki katını taşıması isteri için kabin 800 g ağırlık taşıyabilecek şekilde tasarlanmıştır. Kabini minimize bir sürtünme ile harekete geçirmek amacıyla misina kullanılmıştır ve daha önceden bahsedilen redüktörlü motor kullanımından dolayı yeterli güç kazancı zaten sağlandığından dolayı herhangi bir makara sistemi tercih edilmemiştir.

Kontrolcü Algoritmaları ve Yazılım

Bu kısımda, Resim 2.1.1’de verilen K(s) bloğunu temsil eden Arduino gömülü yazılım kartına asansörün referans takibi için ne gibi kodlar yazıldığı hakkında bilgiler verilecektir. Bu kısımda kodlar, parça parça resimler halinde verilmiştir. Kodun tüm hali ek [1] kısmında mevcuttur.

```
// Buton ve motor pinleri
const int buttonPins[8] = {2, 3, 4, 5, 6, 7, 8, 9}; // 8 butonun bağlı olduğu pinler
int lastButtonStates[8] = {LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW}; // Önceki buton durumları
const int pwmpos = 10;
const int pwmmeg = 11;
const int trigPin = 12;
const int echoPin = 13;

// Durum ve kontrol değişkenleri
int kontrolcu = 0;
int selectedAlgorithm = 0;
int selectedKat = 0;
int currentKat = 0;
bool algorithmSelected = false;
bool katSelected = false;
bool isMoving = false;
bool salinim = false;
int kontrolcuSinyali=0;
float integral=0.0;
float Kd = 1000.0;
float turev=0;
int sonMesafe=0;
int fs=20;

// Kat mesafeleri (cm cinsinden)
const int katMesafeleri[3] = {15, 30, 50};
```

Resim 2.2.1: Proje Esnasında Kullanılacak Değişkenlerin Oluşturumu

Bu kodda ilk olarak “buttonPins” adında butonlardaki sinyali analiz etmek için oluşturulan 8 boyutlu bir tamsayı(integer) dizisi oluşturulmuştur. Ardından uygulanacak “PWM” sinyallerinin Arduino’nun hangi pininden uygulanacağını tespiti için “pwmpos” ve “pwmmeg” değişkenleri tanımlanmıştır. Bu tanımlamadan sonra “trigPin” ve “echoPin” değişkenleri kullanılacak mesafe sensörünün Arduino’nun hangi pininden analiz edileceğini belirlemek için tanımlanmıştır bu pinler, giden ve gelen ses dalgalarını analiz etmek üzerine kurulmuş amaç içerir. Ardından “kontrolcu” değişkeni tanımlanmıştır. Bu değişken aldığı değere göre kullanıcının hangi kontrolcü algoritmasını içermektedir. Bu tanımlardan sonra, asansörün bulunduğu konumu gösteren “currentKat” değişkeni tanımlandı. Ardından asansörün o an hareket edip etmediği ve algoritma ve kat bilgisinin kullanıcı tarafından seçilip seçilmediği kontrol eden 4 adet bool değişken tanımlandı. Projede anahtar bir parametre olan “sonMesafe” değişkeni tanımlandı. Bu değişken daha sonra detaylı olarak açıklanacak türev kontrol algoritmasında öne çıkacaktır.

Constant int(sabit tamsayı) olarak katMesafeleri dizisi tanımlandı burada göze çarpan bir durum mevcuttur. Kat 1'in anlatılan mekanik tasarıma göre 10 cm yükseklikte bulunması gerekirken 15 cm yükseklikte olması gerçeği, bir optimizasyon parametresidir. Projede karşılaşılan bir sorun olan sensör boşluğunun tasarımında 10 cm'nin yeterli olmamasından dolayı bu şekilde bir optimizasyon yöntemi seçilmiştir.

```
void setup() {  
    for (int i = 0; i < 8; i++) {  
        pinMode(buttonPins[i], INPUT);  
    }  
  
    pinMode(pwmpos, OUTPUT);  
    pinMode(pwmneg, OUTPUT);  
    pinMode(trigPin, OUTPUT);  
    pinMode(echoPin, INPUT);  
  
    digitalWrite(pwmpos, LOW);  
    digitalWrite(pwmneg, LOW);  
  
    Serial.begin(9600);  
}
```

Resim 2.2.2: Void Setup Bloğu

Bu kod bloğunda, Resim 2.2.1'de tanımlanan değişkenlerin, Arduino'nun ilgili pinlerden sinyal üreteceği mi yoksa veri alacağı mı sorusuna göreceli olarak bir sınıflandırmaya tabi tutulduğu görülmektedir. Bu bağlamda, Arduino'nun dış çevreye sinyal iletmek için kullanacağı pinlerin output, dış çevreden veri almak için kullanacağı pinlerin ise input olarak tanımlandığı dikkat çekmektedir.

```

void loop() {
    // Kullanıcıdan algoritma ve kat seçimi al
    if (!algorithmSelected || !katSelected) {
        checkButtons();
    }

    // Seçim yapıldıktan sonra asansör kontrolü
    if (algorithmSelected && katSelected) {
        moveElevator();
    }
}

```

Resim 2.2.3: Void Loop Bloğu

Resim 2.2.3'te yer alan kod bloğunda, Arduino'nun çalışma mekanizmasının temelini oluşturan ve belirli bir frekans aralığında sürekli olarak işlemleri tekrarlayan Void Loop bloğu net bir şekilde görülmektedir. Bu döngü, Arduino'nun genel işleyişinde kritik bir rol üstlenir ve tüm komutların ardışık bir düzen içerisinde işlenmesini sağlar. Kodun daha kolay okunabilmesi ve düzenlenebilmesi için, döngü içinde yer alan işlemlerin modüler bir yapıya sahip olacak şekilde fonksiyonlar halinde tanımlanmış olması özellikle tercih edilmiştir. Bu yaklaşım, kodun hem okunabilirlik hem de sürdürülebilirlik açısından disiplinli bir yapıda kalmasını mümkün kılmaktadır. Bu kod yapısında dikkat edilmesi gereken en önemli hususlardan biri, algoritmanın tasarım gereği, kullanıcı bir kat seçimi yapmadan ya da algoritma fonksiyonunu başlatmadan asansör mekanizmasının harekete geçmeyecek şekilde programlanmış olmasıdır. Bu durum, sistemin kullanıcı etkileşimleriyle uyumlu çalışmasını sağlarken, asansörün gereksiz hareketlerini de engelleyerek daha verimli bir kullanım sunmaktadır.

```

void checkButtons() {
  for (int i = 0; i < 8; i++) {
    int buttonState = digitalRead(buttonPins[i]);
    if (buttonState == HIGH && lastButtonStates[i] == LOW) {
      delay(50); // Debounce

      if (i < 5 && !algorithmSelected) {
        selectedAlgorithm = i + 1;
        printAlgorithmName(selectedAlgorithm);
        algorithmSelected = true;
      }
      else if (i >= 5 && !katSelected) {
        selectedKat = i - 4;
        Serial.print("Kat ");
        Serial.print(selectedKat);
        Serial.println(" seçildi!");
        katSelected = true;
      }
    }
    lastButtonStates[i] = buttonState;
  }
}

```

Resim 2.2.4: Kullanıcıdan Referans Alma Fonksiyonu

Bu kod bloklarından hemen sonra, kullanıcıdan referans verisi alma ve bu alınan veriyi bilgisayar ortamına sinyalsel olarak aktarma görevini üstlenen checkButtons adlı fonksiyon yer almaktadır. Bu fonksiyon, 0'dan 7'ye kadar ilerleyen bir for döngüsü ile dikkat çekmektedir. Döngü, daha önce tanımlanmış olan 8 elemanlı buttonPins dizisinin tüm elemanlarına sırasıyla erişebilmek amacıyla tasarlanmıştır. Bu dizinin ilk 5 elemanı, algoritma bilgilerinin işlenmesine yönelik referanslarla ilişkilirken, son 3 elemanı ise kat referansı almak için yapılandırılmıştır. Böylece, dizideki her bir elemanın görev tanımı net bir şekilde ayrılmıştır.

Fonksiyonun içinde, tuşlarda meydana gelebilecek debounce kaynaklı süreksizlikleri engellemek amacıyla kullanılan delay(50) komutu öne çıkmaktadır. Bu komut, herhangi bir tuşa basılma durumunda, 50 milisaniye boyunca ilgili tuşu etkisiz hale getirerek gürültü kaynaklı hataları önlemektedir. Bu sayede, buton durumları daha güvenilir bir şekilde okunabilmektedir.

Debounce işleminin ardından gelen bir if bloğu, ilgili butona daha önce basılıp basılmadığını sorgulamak üzere tasarlanmıştır. Bu sorgu, butona basılma durumunu kontrol eden buttonState değişkeninin değeri üzerinden gerçekleştirilir. Eğer ilgili butona daha önce basılmamışsa ve buttonState değişkeni HIGH olarak okunuyorsa, bu durum, kullanıcının ya bir algoritma referansı ya da bir kat referansı girmek üzere butona bastığını göstermektedir. Bu analiz, kullanıcı etkileşimlerini doğru bir şekilde algılamak için kritik bir rol oynamaktadır.

Söz konusu if bloğunun içinde, kullanıcı tarafından basılan butonun hangi tür referansa (algoritma mı yoksa kat mı) karşılık geldiğini belirlemek için iki ayrı if bloğu daha bulunmaktadır. Bu iç içe geçmiş if blokları, basılan butonun referans türünü detaylı bir şekilde analiz eder ve alınan bilginin doğru sınıflandırılmasını sağlar. Yapılan analizler sonucunda, gerekli referansın tanımlanmasıyla birlikte, ilgili bool ifadeler, daha önce standart olarak atanmış olan 0 değerinden 1 değerine geçirilir. Bu işlem, referansın algılandığını ve işlemeye hazır olduğunu göstermek için yapılır.

Sonuç olarak, bu fonksiyon, kullanıcıdan gelen sinyallerin güvenilir bir şekilde işlenmesini ve algoritma ile kat referanslarının ayrımını yaparak sistemin doğru bir şekilde çalışmasını sağlayan önemli bir bileşendir.

```
// Mesafe ölçen fonksiyon
long measureDistance() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH);
    long distance = duration * 0.034 / 2; /
    return distance;
}
```

Resim 2.2.5: Mesafe Ölçüm Fonksiyonu

Resim 2.2.5'teki fonksiyon, ultrasonik sensörün “trig” ve “echo” pinleriyle çalışarak, asansörün yerden yüksekliğini santimetre cinsinden döndürür. Sensör, “trig” pinine 10 mikrosaniyelik bir tetik sinyali göndererek ses dalgalarını yayar; bu dalgalar bir nesneye çarptıktan sonra geri yansır ve “echo” pini tarafından geri dönüş süresi ölçülür. Geri dönüş süresi, dalganın toplam gidip gelme zamanını temsil eder. Fonksiyon, bu süreyi kullanarak mesafeyi hesaplar: süre, ses hızına (yaklaşık 0.0343 cm/ μ s) bölünür ve gidiş-dönüş mesafesini bulmak için ikiye ayrılır. Sonuç, sensörün ölçtüğü mesafeyi güvenilir bir şekilde santimetre cinsinden döndürür.

Bu kod blokları ve fonksiyonların ardından projenin ana misyonu olan asansör hareketi ile ilgili olan görece büyük boyutlu olan “moveElevator” fonksiyonu yer almaktadır.

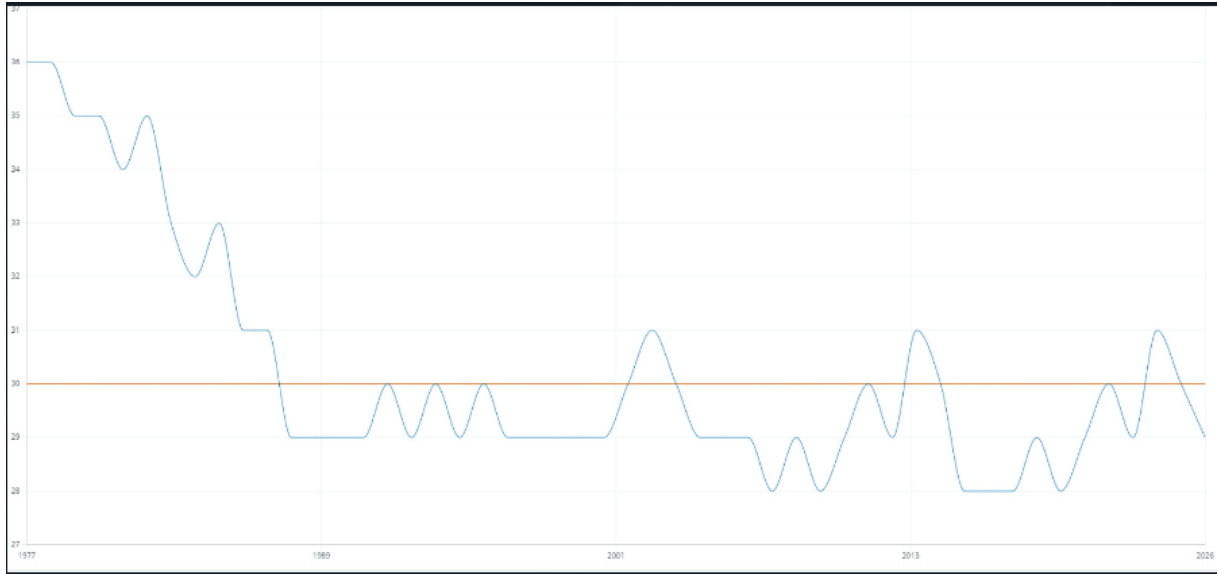
moveElevator fonksiyonunda, asansör hareket ederken ve hedef katına ulaştığında sürekli olarak bir döngü içinde kalır. Kullanılan ilk kontrol algoritması ise Aç-kapa algoritması olup, asansörün düzgün ve istenilen şekilde çalışmasını sağlamak amacıyla temel bir kontrol mekanizması olarak devreye girer.

Aç-Kapa Kontrol Algoritması

```
if (kontrolcu == 1) {  
    integral=0;  
    int santimkat = katMesafeleri[selectedKat - 1] - distance;  
    Serial.print(distance); // Mesafe  
    Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için  
    Serial.println(katMesafeleri[selectedKat - 1]);  
    delay(4);  
  
    if (santimkat > 0.1) {  
        digitalWrite(pwmpos, HIGH);  
        digitalWrite(pwmneg, LOW);  
        if (abs(santimkat) < 2) {  
            currentKat = selectedKat;  
            algorithmSelected = false;  
            katSelected = false;  
        }  
    } else if (santimkat < 0.1) {  
        digitalWrite(pwmpos, LOW);  
        digitalWrite(pwmneg, HIGH);  
        if (abs(santimkat) < 2) {  
            currentKat = selectedKat;  
            algorithmSelected = false;  
            katSelected = false;  
        }  
    }  
}
```

Resim 2.2.6: Aç-Kapa Kontrol Algoritması

Resim 2.2.6’da verilen algoritma bir if bloğu ile başlar. Bu if bloğu kontrolcünün değerini sorgulamaktadır. Kontrolcü değişkeni 1’e eşit ise bu kullanıcının aç-kapa algoritması seçtiğini göstermektedir. Projenin anahtar değişkenlerinden biri olan ve Resim 2.2.1’deki blok şemasında referans ile sensörün çıktısı arasındaki hatayı temsil eden değişken “santimkat” değişkenidir. Bu değişkenin eşik değeri olarak seçilen 0.1 değerinden küçük olup olmadığını sorgulayan iki tane if bloğu mevcuttur. Bu blokların ilki santimkat değişkeninin pozitif olduğu durumları içerir. “santimkat” değişkeninin pozitif olması fiziksel olarak asansörün istenilen katın aşağısında olmasına karşılık gelir. Bu durumda önceden bahsedilen h-bridge devresinin pozitif pinine “HIGH” sinyali verilmiştir ve asansörün yerçekimine ters yönlü harekete geçmesi sağlanmıştır. Bu durum ile karşı kutup (antipodle) olan durum ise santimkat değişkeninin negatif olduğu durumdur. Tahmin edilebileceği üzere bu durumda da asansörün aşağı yönlü hareketi sağlanmıştır. Sonuç olarak aç-kapa kontrolcünde kullanıcı tekrardan bir referans verene kadar sürekli bir salınım olacağı açıktır.



Resim 2.2.7: Aç-Kapa Kontrol Analiz Grafiği

Resim 2.2.7’de, turuncu renkli çizgi y-ekseninde 30 cm yüksekliği temsil ederek asansör sistemindeki kat 2’nin konumunu gösterir; bu referans değer, asansörün hedeflediği noktadır. Aynı grafikte, mavi renkli çizgi ise measureDistance fonksiyonundan gelen "distance" değişkeninin zamana bağlı değerlerini belirli bir baud rate ile grafik üzerinde gösterir. Analizde, başlangıçta kat 3’te bulunan asansörün kat 2’ye doğru yönelme hareketi incelenmiştir. Grafik, asansör hareketinin belirli bir süre

sonunda turuncu çizgi olan referans değere yaklaştığını, ancak bu değerin etrafında salınım yapmaya başladığını ortaya koymuştur. Bu salınım, aç-kapa kontrol sistemlerinin karakteristiği olarak teorik bilgiyle uyumlu bir durumdur. Aç-kapa kontrol mekanizması gereği, sistem referans noktasına tam olarak sabitlenemez; zaman sonsuza ilerlese bile referans değere küçük bir hata payıyla yaklaşır, ancak tamamen oturmaz. Bu durum, kontrol algoritmasının doğasına uygun bir sonuçtur ve beklenmedik bir anomali ya da hata içermez. Resim 2.2.7’deki grafiğin Arduino IDE derleyicisinin “serial plotter” özelliği ile çizdirildiğine dikkat ediniz. Daha detaylı ve uzun süre örneği içeren bir analiz için Matlab kullanılması gerekmektedir.

Aç kapa algoritmasından sonra basit kontrol eylemlerinin ikincisi olarak sayılabilecek P kontrol algoritması asansör hareketi kontrolü için tercih edilmiştir.

P-Kontrol Algoritması

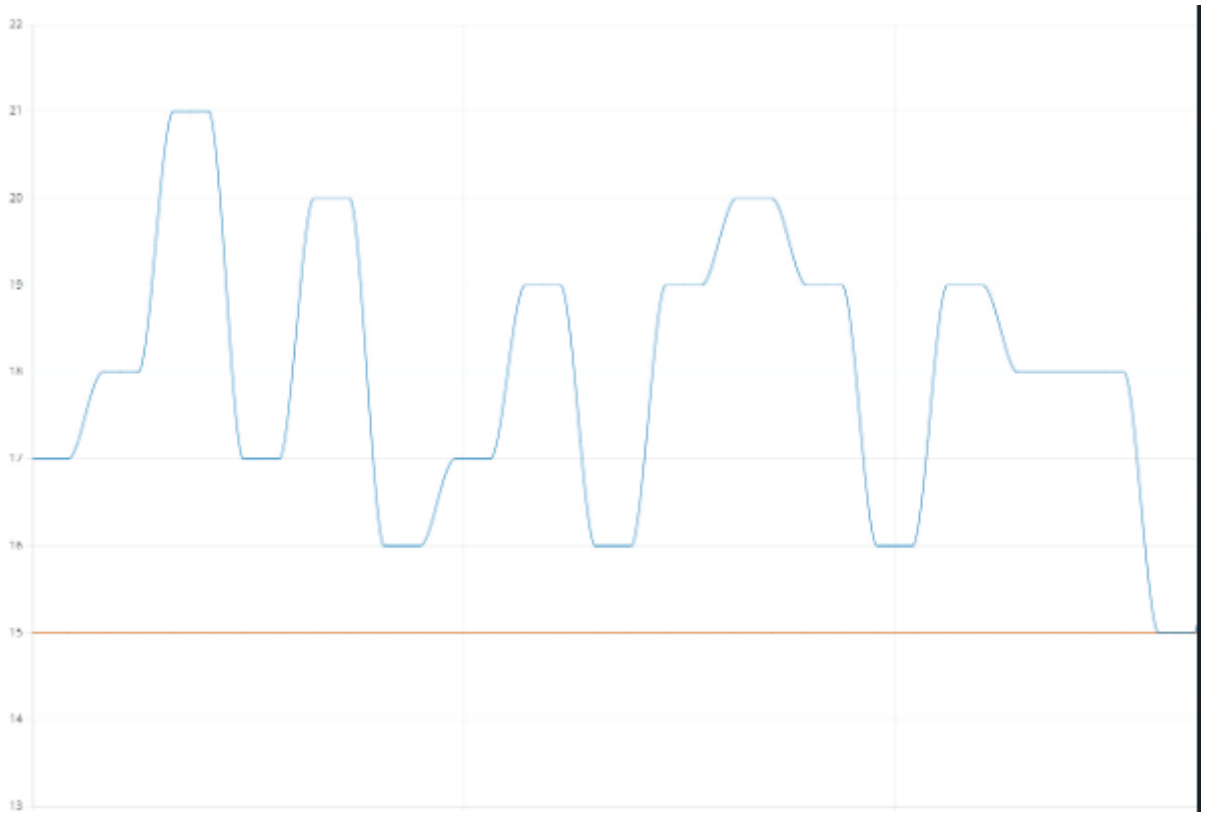
```
if (kontrolcu == 2) { // P Kontrol
int santimkat = katMesafeleri[selectedKat - 1] - distance;
integral =0;
int Kp=14;
    Serial.print(distance); // Mesafe
Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
Serial.println(katMesafeleri[selectedKat - 1]);

    kontrolcuSinyali = Kp*abs(santimkat);

    Serial.println(distance);
    //Serial.print("Kontrolcu Sinyali : ");
    //Serial.println(kontrolcuSinyali);
    if (santimkat > 0) {
        analogWrite(pwmpos, constrain(kontrolcuSinyali,190,255));
        digitalWrite(pwmneg, LOW);
        if (abs(santimkat) < 2) {
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    } else if (santimkat < 0) {
        digitalWrite(pwmpos, LOW);
        analogWrite(pwmneg, constrain(kontrolcuSinyali,50,255));
        if (abs(santimkat) < 2) {
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    }
    if(abs(santimkat) <= 0.1){
        digitalWrite(pwmpos, LOW);
        digitalWrite(pwmneg, LOW);
        currentKat = selectedKat;
        algorithmSelected = false;
    }
```

Resim 2.2.8: P-Kontrol Algoritması

Bu algoritma türünde, ilk defa hatanın işaretine ek olarak hatanın değeri de uygulanacak sinyale dramatik bir etki etmiştir. “kontrolcuSinyali” adında yeni bir değişken tanımlanmıştır. Bu değişken, hataya göre kontrolcünün referans takibi için sisteme uygulayacağı gerilimi notasyonize etmektedir. Ve tanımlanmasında $K_p \times santimkat$ işlemi kullanılmıştır. Bu tanımlamada santimkat değişkeninin değeri yani hata azaldıkça, uygulanacak kontrolcü sinyalinin oransal kontrolcü kazancı ile doğru orantılı olarak azalacağı gözlemlenmelidir bu durum fiziksel olarak asansörün kata yaklaştıkça yavaşlayacağı anlamına gelmektedir. Aç kapa algoritmasında olduğu gibi if blokları ile asansör kata yaklaştıktan sonra kontrol için gerekli bool değişkenleri false haline getirilmiştir. Bu durum kullanıcının yeni bir kat referansı girmesine olanak tanır.



Resim 2.2.9: P Kontrol Hareket Analiz Grafiği

Resim 2.2.9’den de görüldüğü gibi asansöre yüksekliği 15 cm olan turuncu çizgi kat 1 referansı verilmiştir ve asansörün o kata yaklaşım hareketi mavi eğri ile modellenmiştir. Grafiğin sağ tarafına ilerledikçe mesafenin değişiminin azaldığını gözlemleyiniz. Eğride görünen salınımlar, sensörün ideal olmamasından ve ortam koşullarından kaynaklanmaktadır. Bu kısımda çeşitli kontrollü deney ve süre ölçümleri sonucunda oransal kontrolcü kazancı (K_p) değerinin 14 olması tercih edilmiştir. Bu değer daha azı veya daha fazlası asansör hareketinde istenilen optimizasyonu karşılamamıştır.

PI-Kontrol Algoritması

Aç kapa ve p kontrol algoritmalarından sonra yeni bir kontrolcü tipi olan PI(oransal integral) kontrolcü devreye girmiştir. Bu kontrolcü sistemin çıkışı yani asansörün yerden yüksekliği değişkeni ile referans ile arasında oluşabilecek sürekli hal hatasını hatanın işareti ve büyüklüğünün yanında geçmişine bakarak giderir. Zaman geçtikçe integral değişkeninin artacağı açıktır.

```
,
if (kontrolcu == 3) { // PI Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;

    Serial.print(distance); // Mesafe
    Serial.print("\t"); |
    Serial.println(katMesafeleri[selectedKat - 1]);

    int Kp=1;
    float Ki =0.1;

    //Serial.println(santimkat);

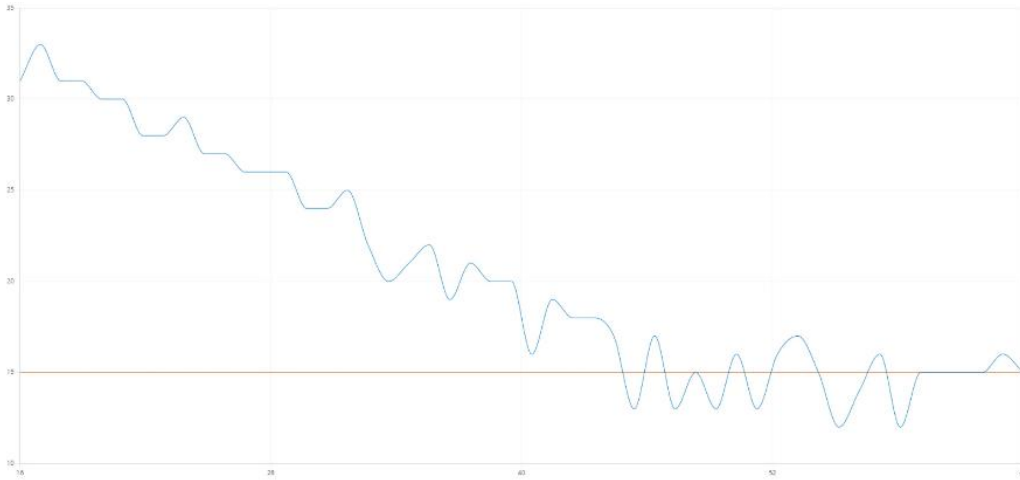
    if (santimkat > 0.05) {

        integral +=abs(santimkat);
        kontrolcuSinyali = Ki*abs(integral)+Kp*abs(santimkat);
        analogWrite(pwmpos, constrain(kontrolcuSinyali,190,255));
        digitalWrite(pwmneg, LOW);
        if (abs(santimkat) < 2) {
            integral=0;
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    } else if (santimkat < 0.05) {

        integral +=abs(santimkat);
        kontrolcuSinyali = Ki*abs(integral)+Kp*abs(santimkat);
        digitalWrite(pwmpos, LOW);
        analogWrite(pwmneg, constrain(kontrolcuSinyali,50,255));
        if (abs(santimkat) < 2) {
            integral=0;
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    }
}
```

Resim 2.2.10: PI Kontrol Algoritması

Bu algoritmada yeni bir değişken olan integral değişkeni göze çarpmaktadır. Bu değişken asansörü hareket halinde tutan “moveElevator” fonksiyonundaki sonsuz döngüde her döngü başına güncellenmektedir. Bu güncelleme hatayı temsil eden “santimkat” değişkeninin her döngüdeki değerinin başta 0 olarak tanımlanan integral değişkenine eklenmesiyle olmaktadır. Eğrinin altında kalan alanın pozitif olması gerektiğinden dolayı ekleme işleminde “abs” fonksiyonu tercih edilmiştir. Ardından denklem (2)’ye göre bir kontrolcü sinyali oluşturulur. Kontrolcü sinyalinde integral değişkeni, integral kontrolcü kazancı olan K_i ile çarpılmaktadır. Döngünün sürekli döndüğünden ve hatanın birikimli olarak artmasından dolayı integral kazanç katsayısının görece düşük (0.1) seçildiğine dikkat ediniz. Ardından asansörün referansın aşağısında mı yukarısında mı olduğuna göre gerekli pwm sinyalleri tanımlanmıştır. “analogWrite” fonksiyonu ile pozitif pwm “pwmpos” değişkenine constrain ile 190 ile 255 arasında pwm sinyalleri atanmıştır. Burada “constrain” fonksiyonunun tercih edilme sebebi, p kontrolcüsündeki hata azalımından dolayı kontrolcü sinyalindeki hatırı sayılır düşüşün fiziksel koşullarda asansörü hiç hareket ettirememesidir. Constrain fonksiyonu ile kontrolcü fenomeni olan sinyal düşüşü etkilenmemiştir ancak belirli bir aralıkta tutulmuştur.



Resim 2.2.11: PI Kontrol Hareket-Analiz Grafiği

Resim 2.2.11’deki grafik dikkatle incelendiğinde, zaman ilerledikçe hata azaldıkça orantısız (P) kontrolcünün etkisinin zayıfladığı, buna karşılık integral kontrolcünün birikimli etkisinin giderek arttığı açıkça görülmektedir. Asansör, hedeflenen kat referansına ulaştığında, kontrol mekanizmasının doğası gereği referans değerin etrafında salınımlar yaptığı fark edilmiştir. Bu salınımlar, hatanın sürekli olarak artıp azalmasına bağlı olarak integral kontrolcünün birikim yapmasından kaynaklanmaktadır ve bu durum kontrol algoritmasının doğasına uygun bir sonuçtur. Ancak, beklenmeyen ani veya büyük

salınımların önüne geçmek amacıyla, hata birikimini temsil eden “integral” değişkeni, asansör hedef kata ulaştığında sıfırlanmış ve bu değişkenin yalnızca belirli bir hata bandı içerisinde hareket etmesi sağlanmıştır. Bu yaklaşımla, sistemin stabilitesi artırılmış ve hem gereksiz osilasyonlar hem de aşırı birikimden kaynaklanan kontrol hataları önlenmiştir.

PD-Kontrol Algoritması

Önceki kontrolcülerde hatanın değişim hızına hiç değinilmediğine dikkat ediniz. Bu kısımda, hatanın anlık değişimine göre kontrolcü sinyalinde değişiklik elde edilecektir. Bu değişikliğin tek başına bir işe yaramayacağı için orantısız(p) kontrolcü ile kombin edildiğine dikkat ediniz.

```
if (kontrolcu == 4) { // PD Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;
    int Kp=6;
    int Kd=15;
    Serial.print(distance); // Mesafe
    Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
    Serial.println(katMesafeleri[selectedKat - 1]);
    turev = (sonMesafe)/fs;

    kontrolcuSinyali = Kp*abs(santimkat)+Kd*(turev);

    if (santimkat > 0) {

        sonMesafe=(santimkat);
        kontrolcuSinyali = Kd*(turev)+Kp*abs(santimkat);
        analogWrite(pwmpos, constrain(kontrolcuSinyali,190,255));
        digitalWrite(pwmneg, LOW);
        if (abs(santimkat) < 2) {

            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    } else if (santimkat < 0) {

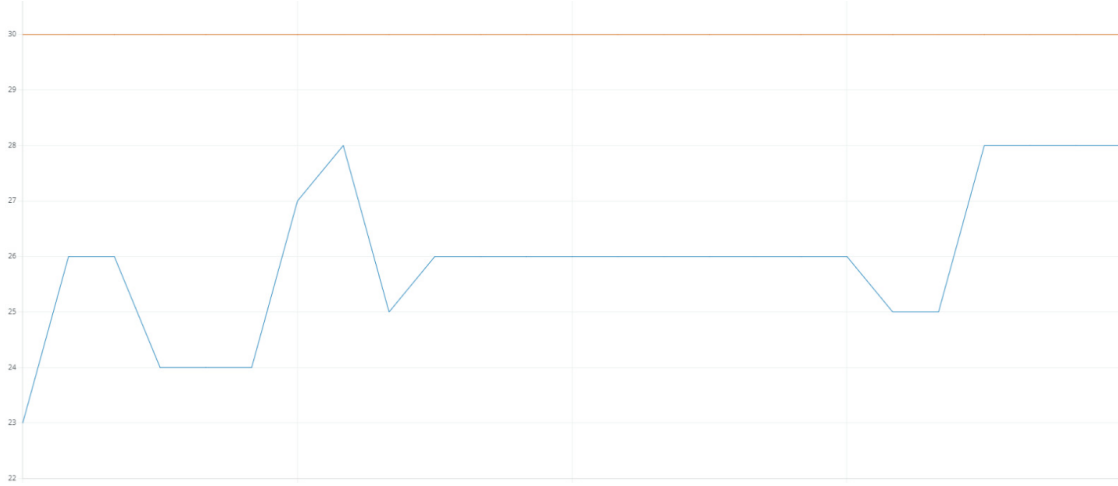
        sonMesafe=abs(santimkat);
        kontrolcuSinyali = Kd*(turev)+Kp*abs(santimkat);
        digitalWrite(pwmpos, LOW);
        analogWrite(pwmneg, constrain(kontrolcuSinyali,50,255));
        if (abs(santimkat) < 2) {

            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    }
}
```

Resim 2.2.11: PD-Kontrol Algoritması

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (5)$$

Bu algoritmada temel kurulum formatının önceki algoritmalarla aynı olmasına karşılık $\text{turev} = (\text{sonMesafe}/\text{fs})$ dikkat çekmektedir. Bu ifadede sonMesafe, hatanın önceki değeri ile arasındaki farka karşılık gelirken, fs ifadesi sensör ölçüm frekansı olan 20 Hz değerine denk gelmektedir. Fiziksel olarak sonsuz ve sıfır kavramları elde edilemeyeceği için denklem (5) 'te bulunan h ifadesi 0'a kadar gönderilemese de saniyenin 20'de birine gönderilmiştir. Bu durumun fiziki olarak türev gerçeğine görece iyi bir yakınsama sunacağı açıktır. Diğer kontrolcülerden farklı olarak türev kontrolcü sürekli hal hatasına herhangi bir etkide bulunamaz çünkü hatanın değerinden bağımsızdır.



Resim 2.2.12: PD Kontrol Algoritması Hareket-Analiz Grafiği

Resim 2.2.12 incelendiğinde, kat referansı olarak belirlenen 30 cm yüksekliğindeki 2. kata yönelen asansörün, zamanla bulunduğu konumdan yükselmeye başladığı görülmektedir. Ancak PD kontrolcünün bileşenlerinden olan orantısal (P) kontrol, hata azaldıkça etkisini kaybetmiş ve devreye giren türev (D) kontrolcü, hatanın miktarı yüksek olsa bile P kontrolcünün zayıflayan etkisi nedeniyle asansörün hareketini yavaşlatmıştır. Bu durum, hatadaki değişim tespitinin giderek minimal bir seviyeye inmesine yol açmıştır. Neticede, sistemin doğası gereği sürekli hal hatası ortaya çıkmış, grafik incelendiğinde asansörün 30 cm'lik referans yüksekliğine tam olarak ulaşamadığı ve yaklaşık 26 cm seviyesinde kalarak durakladığı gözlemlenmiştir. Bu sonuç, sürekli hal hatasının belirgin bir örneğini temsil

etmektedir. Bu kısımda türevsel kontrolcü kazancı ve oransal kontrol kazancı $K_p = 6$ ve $K_d = 15$ olarak seçilmiştir. Bu seçimde K_p , türev etkisinin fiziksel olarak daha rahat gözlemlenebilmesi için türev kazanç katsayısına göre daha düşük tutulmuştur.

PID-Kontrol Algoritması

Sürekli hal hatası ve salınım gibi olaylar göz önüne alındığında hata değişimi, hata miktarı ve hatanın geçmişinin hepsinin kullanılması uygun görülmüştür.

```
if (kontrolcu == 5) { // PID Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;
    int Kp=6;
    Serial.print(distance); // Mesafe
    Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
    Serial.println(katMesafeleri[selectedKat - 1]);
    int Kd=40;
    int Ki = 0.08;
    integral +=abs(santimkat);

    turev =(sonMesafe)/fs;

    kontrolcuSinyali = Kp*abs(santimkat)+Kd*(turev)+Ki*abs(integral);

    if (santimkat > 0) {

        sonMesafe=(santimkat);
        kontrolcuSinyali = Kd*(turev)+Kp*abs(santimkat)+Ki*abs(integral);
        analogWrite(pwmpos,constrain(kontrolcuSinyali,190,255));
        digitalWrite(pwmneg, LOW);
        if (abs(santimkat) < 2) {
            integral=0;

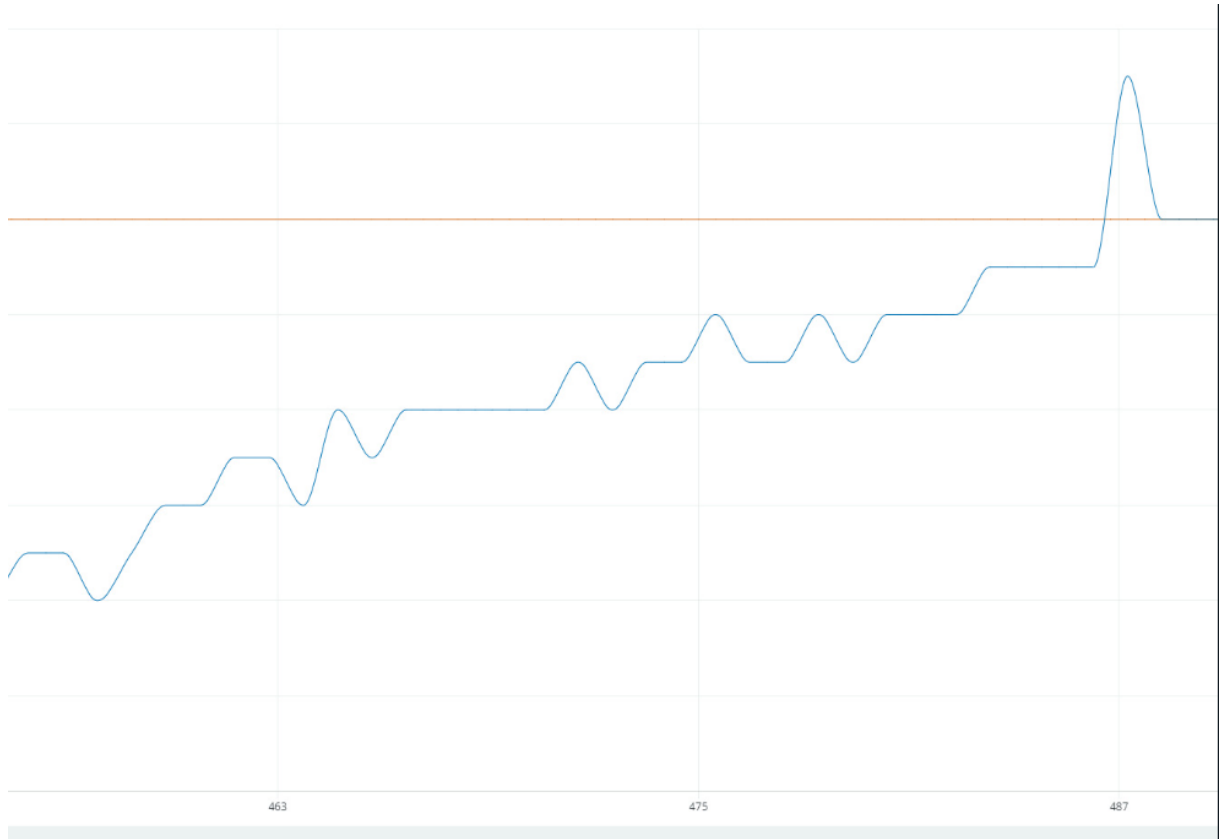
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    } else if (santimkat < 0) {

        sonMesafe=(santimkat);

        kontrolcuSinyali = Ki*abs(integral)+Kp*abs(santimkat)+Kd*(turev);
        digitalWrite(pwmpos, LOW);
        analogWrite(pwmneg,constrain(kontrolcuSinyali,50,255));
        if (abs(santimkat) < 2) {
            integral=0;
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    }
}
```

Resim 2.2.13: PID Kontrolör Algoritması

Bu kısımda temel algoritma formatı, önceki kısımlarla aynıdır ancak bu sefer kontrolcü sinyalinin hem türev hem hata birikimi hem de hatanın değeri kontrolcü kazanç katsayılarıyla orantılı bir lineer kombinasyon işlemi ile oluşturumu elde edilmiştir. Bu durumda analogWrite ve digitalWrite komutları ile asansörün referans katın aşağısında mı yukarısında mı olduğuna göre uygun pwm sinyali verilmiştir. Bu algorithmada yine “constrain” tercih edilmiştir. Bu tercih kontrolün temel doğasında bulunmasada fiziksel sürtünme ve sensör hataları için uygun bir çözüm olmuştur.



Resim 2.2.14: PID Kontrol Hareket-Analiz Grafiği

Resim 2.2.14'teki grafik incelendiğinde sisteme butonlar aracılığı ile verilen yüksekliği 50 cm olan 3.kat referansını asansörün mavi eğri ile takip ettiği görülmüştür. Bu eğrideki görece küçük salınımlar sensör hatasından kaynaklanmaktadır ve teoride zarf salınımı bulunmayan bir eğri olması gerektiği açıktır. Grafiğin en sağ tarafı detaylı olarak incelendiğinde, mavi eğrinin kısa bir süre için turuncu çizgiyi geçtiği görülmektedir. Bu durum, aşım miktarı olarak adlandırılır ve beklenen bir durumdur çünkü aşım gerçekleştikten sonra mavi eğri turuncu çizgi olan kat referansına oturmuştur. Bu tarz aşım ve oturma gözlemlenen sistemlerde eksik sönüm durumu mevcuttur ve sistemin sönüm

oranının (ζ) 1 değerinden düşük olması durumunda gerçekleşir. Seçilen türev integral ve orantısal kontrolcü kazançlarında integral kazancı birikimli bir ifade olacağı için görece düşük integral kazancı ayrıca türev etkisinin rahat gözlemlenmesi için görece büyük bir türev kontrol kazancı tercih edilmiştir orantısal kontrolcü kazancı ise bu ikisinin ortasında bir değer olarak seçilmiştir.

Bütün algoritmalarda sistemlerin kararlı olduğu açıktır çünkü sisteme uygulanan voltaj sınırlıdır ve sınırlı bir voltajla sonsuzluk elde edilemeyeceği bir gerçektir. Referans etrafında görülen minimal salınımlar, sistemin kararsızlığından değil ideal olmayan ortam koşulları, fiziki kısıtlar ve sensör hatalarından olmaktadır.

Sonuçlar

Bu projede gerçekleştirilen çalışmalar, iki temel ana başlık altında detaylı olarak ele alınmıştır. Mekanik tasarım kısmında, sistemin hareket performansını iyileştirmek ve enerji kayıplarını en aza indirmek amacıyla sürtünme gibi olumsuz faktörleri minimize edecek tasarım kararları dikkatle alınmıştır. Burada, sistemin stabilitesini artırmak ve akıcı bir hareket sağlamak adına, mekanik bileşenlerin seçimi ve düzenlenmesi büyük bir özenle yapılmıştır.

Elektronik tasarım sürecinde ise proje kapsamındaki önemli bir kısıt olarak hazır sürücü kartlarının kullanımının mümkün olmaması gerçeğiyle karşılaşmış ve bu nedenle, tamamen özelleştirilmiş bir H-Bridge devresi tasarlanıp devreye alınmıştır. Tasarım sürecinde kullanılan elektronik bileşenlerin işlevleri, performans kriterleri ve projeye katkıları kapsamlı bir şekilde tanıtılmış ve analiz edilmiştir. Her bir bileşenin seçimi, tasarım ihtiyaçları ve işlevsellik doğrultusunda titizlikle gerçekleştirilmiştir.

Projenin en dikkat çekici kısımlarından biri olan kontrol algoritmaları, 2.2 numaralı başlık altında ayrıntılı olarak ele alınmıştır. Bu algoritmaların işlevselliğini ve doğruluğunu analiz etmek adına, her bir kontrol mekanizması için asansörün anlık yüksekliği ile hedeflenen referans değerin zamana bağlı olarak değişimini gösteren grafikler hazırlanmıştır. Bu grafiklerin çizimi, veri görselleştirmenin etkin bir şekilde yapılabilmesi için Arduino'nun "serial plotter" aracı kullanılarak gerçekleştirilmiştir.

Analiz sürecinde, özellikle minimal salınım fenomeni olarak adlandırılan durum dikkat çekmiş ve bu durumun altında yatan temel nedenler detaylı bir şekilde incelenmiştir. Teorik kontrol bilgileri ile elde edilen grafikler arasında oluşan farklar titizlikle değerlendirilmiş, kontrol algoritmalarının sistem üzerindeki etkisi somut verilerle desteklenmiştir. Nihayetinde, projenin her bir aşamasında yapılan

çalışmalar, sistematik bir şekilde ele alınarak gerek mekanik gerek elektronik tasarım süreçlerinde karşılaşılan zorluklar çözülmüş ve elde edilen sonuçlar kapsamlı bir şekilde sunulmuştur.

Kaynakça

- https://www.seiect.com/Catalog/SEI-CF_CFM.pdf [1]
- <https://www.alldatasheet.com/datasheet-pdf/view/1960182/SY/BD140.html> [2]
- <https://www.alldatasheet.com/datasheet-pdf/view/16177/PHILIPS/BD139-16.html> [3]

Ekler

```
// Buton ve motor pinleri
const int buttonPins[8] = {2, 3, 4, 5, 6, 7, 8, 9}; // 8 butonun
bağlı olduğu pinler
int lastButtonStates[8] = {LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW};
// Önceki buton durumları
const int pwmpos = 10;
const int pwmmeg = 11;
const int trigPin = 12;
const int echoPin = 13;

// Durum ve kontrol değişkenleri
int kontrolcu = 0;
int selectedAlgorithm = 0;
```

```

int selectedKat = 0;
int currentKat = 0;
bool algorithmSelected = false;
bool katSelected = false;
bool isMoving = false;
bool salinim = false;
int kontrolcuSinyali=0;
float integral=0.0;
float Kd = 1000.0;
float turev=0;
int sonMesafe=0;
int fs=20;

// Kat mesafeleri (cm cinsinden)
const int katMesafeleri[3] = {15, 30, 50};

void setup() {
    for (int i = 0; i < 8; i++) {
        pinMode(buttonPins[i], INPUT);
    }

    pinMode(pwmpos, OUTPUT);
    pinMode(pwmneg, OUTPUT);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    digitalWrite(pwmpos, LOW);
    digitalWrite(pwmneg, LOW);

    Serial.begin(9600);
}

void loop() {

```

```

// Kullanıcıdan algoritma ve kat seçimi al
if (!algorithmSelected || !katSelected) {
    checkButtons();
}

// Seçim yapıldıktan sonra asansör kontrolü
if (algorithmSelected && katSelected) {
    moveElevator();
}
}

// Butonları kontrol eden fonksiyon
void checkButtons() {
    for (int i = 0; i < 8; i++) {
        int buttonState = digitalRead(buttonPins[i]);
        if (buttonState == HIGH && lastButtonStates[i] == LOW) {
            delay(50); // Debounce

            if (i < 5 && !algorithmSelected) {
                selectedAlgorithm = i + 1;
                printAlgorithmName(selectedAlgorithm);
                algorithmSelected = true;
            }
            else if (i >= 5 && !katSelected) {
                selectedKat = i - 4;
                Serial.print("Kat ");
                Serial.print(selectedKat);
                Serial.println(" seçildi!");
                katSelected = true;
            }
        }
        lastButtonStates[i] = buttonState;
    }
}

```

```

}

// Algoritma isimlerini yazdıran fonksiyon
void printAlgorithmName(int algorithm) {
    switch (algorithm) {
        case 1:
            Serial.println("Aç Kapa algoritması seçildi!");
            kontrolcu = 1;
            break;
        case 2:
            Serial.println("P algoritması seçildi!");
            kontrolcu = 2;
            break;
        case 3:
            kontrolcu=3;
            Serial.println("PI Kontrol algoritması seçildi!");
            break;
        case 4:
            kontrolcu=4;
            Serial.println("PD Kontrol algoritması seçildi!");
            break;
        case 5:
            kontrolcu=5;
            Serial.println("PID Kontrol algoritması seçildi!");
            break;
    }
}

// Mesafe ölçen fonksiyon
long measureDistance() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);

```

```

delayMicroseconds(10);
digitalWrite(trigPin, LOW);

long duration = pulseIn(echoPin, HIGH);
long distance = duration * 0.034 / 2; // Mesafeyi santimetre
cinsinden hesapla
return distance;
}

// Asansörü hedef kata taşıyan fonksiyon
void moveElevator() {
    isMoving = true;
    long distance = measureDistance();

    while (true) {
        // Aç-Kapa kontrol algoritması

        if (kontrolcu == 1) {
            integral=0;
            int santimkat = katMesafeleri[selectedKat - 1] - distance;
            Serial.print(distance); // Mesafe
            Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
            Serial.println(katMesafeleri[selectedKat - 1]);
            delay(4);

            if (santimkat > 0.1) {
                digitalWrite(pwmpos, HIGH);
                digitalWrite(pwmneg, LOW);
                if (abs(santimkat) < 2) {
                    currentKat = selectedKat;
                    algorithmSelected = false;
                    katSelected = false;
                }
            }
        }
    }
}

```

```

    }
    } else if (santimkat < 0.1) {
        digitalWrite(pwmpos, LOW);
        digitalWrite(pwmneg, HIGH);
        if (abs(santimkat) < 2) {
            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    }
}

}

if (kontrolcu == 2) { // P Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;
    integral = 0;
    int Kp=14;

    Serial.print(distance); // Mesafe
    Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
    Serial.println(katMesafeleri[selectedKat - 1]);

    kontrolcuSinyali = Kp*abs(santimkat);

    Serial.println(distance);
    //Serial.print("Kontrolcu Sinyali : ");
    //Serial.println(kontrolcuSinyali);
    if (santimkat > 0) {
        analogWrite(pwmpos, constrain(kontrolcuSinyali, 190, 255));
        digitalWrite(pwmneg, LOW);
        if (abs(santimkat) < 2) {
            currentKat = selectedKat;
            algorithmSelected = false;

```

```

        katSelected = false;

    }
    } else if (santimkat < 0) {
        digitalWrite(pwmpos, LOW);
        analogWrite(pwmneg, constrain(kontrolcuSinyali, 50, 255));
        if (abs(santimkat) < 2) {

            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
            if(abs(santimkat) <= 0.1){
                digitalWrite(pwmpos, LOW);
                digitalWrite(pwmneg, LOW);
                currentKat = selectedKat;
                algorithmSelected = false;

            }
        }
    }

}

if (kontrolcu == 3) { // PI Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;

    Serial.print(distance); // Mesafe
    Serial.print("\t");
    Serial.println(katMesafeleri[selectedKat - 1]);

    int Kp=1;
    float Ki =0.1;

```



```

//Serial.println(santimkat);

if (santimkat > 0.05) {

    integral +=abs(santimkat);
    kontrolcuSinyali = Ki*abs(integral)+Kp*abs(santimkat);
    analogWrite(pwmpos,constrain(kontrolcuSinyali,190,255));
    digitalWrite(pwmneg, LOW);
    if (abs(santimkat) < 2) {
        integral=0;
    }
    currentKat = selectedKat;
    algorithmSelected = false;
    katSelected = false;
}
} else if (santimkat < 0.05) {

    integral +=abs(santimkat);
    kontrolcuSinyali = Ki*abs(integral)+Kp*abs(santimkat);
    digitalWrite(pwmpos, LOW);
    analogWrite(pwmneg,constrain(kontrolcuSinyali,50,255));
    if (abs(santimkat) < 2) {
        integral=0;
    }
    currentKat = selectedKat;
    algorithmSelected = false;
    katSelected = false;
}
}

```

```
}
```

```
if (kontrolcu == 4) { // PD Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;
    int Kp=6;
    int Kd=15;
    Serial.print(distance); // Mesafe
    Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
    Serial.println(katMesafeleri[selectedKat - 1]);
    turev = (sonMesafe)/fs;

    kontrolcuSinyali = Kp*abs(santimkat)+Kd*(turev);

    if (santimkat > 0) {

        sonMesafe=(santimkat);
        kontrolcuSinyali = Kd*(turev)+Kp*abs(santimkat);
        analogWrite(pwmpos, constrain(kontrolcuSinyali,190,255));
        digitalWrite(pwmneg, LOW);
        if (abs(santimkat) < 2) {

            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    } else if (santimkat < 0) {
```

```

        sonMesafe=abs(santimkat);
        kontrolcuSinyali = Kd*(turev)+Kp*abs(santimkat);
        digitalWrite(pwmpos, LOW);
        analogWrite(pwmneg,constrain(kontrolcuSinyali,50,255));

        if (abs(santimkat) < 2) {

            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
    }

}

if (kontrolcu == 5) { // PID Kontrol
    int santimkat = katMesafeleri[selectedKat - 1] - distance;
    int Kp=6;

    Serial.print(distance); // Mesafe
    Serial.print("\t"); // Tab karakteri, iki veriyi ayırmak için
    Serial.println(katMesafeleri[selectedKat - 1]);

    int Kd=40;
    int Ki = 0.08;

    integral +=abs(santimkat);

    turev =(sonMesafe)/fs;

    kontrolcuSinyali =
    Kp*abs(santimkat)+Kd*(turev)+Ki*abs(integral);

    if (santimkat > 0) {

        sonMesafe=(santimkat);
    }
}

```

```

        kontrolcuSinyali =
Kd*(turev)+Kp*abs(santimkat)+Ki*abs(integral);

        analogWrite(pwmpos,constrain(kontrolcuSinyali,190,255));
        digitalWrite(pwmneg, LOW);

        if (abs(santimkat) < 2) {
            integral=0;

            currentKat = selectedKat;
            algorithmSelected = false;
            katSelected = false;
        }
        } else if (santimkat < 0) {

            sonMesafe=(santimkat);

            kontrolcuSinyali =
Ki*abs(integral)+Kp*abs(santimkat)+Kd*(turev);

            digitalWrite(pwmpos, LOW);
            analogWrite(pwmneg,constrain(kontrolcuSinyali,50,255));

            if (abs(santimkat) < 2) {
                integral=0;

                currentKat = selectedKat;
                algorithmSelected = false;
                katSelected = false;
            }
        }

    }

    distance = measureDistance();
    delay(275);
    checkButtons();
}

```

```
    isMoving = false;  
}
```