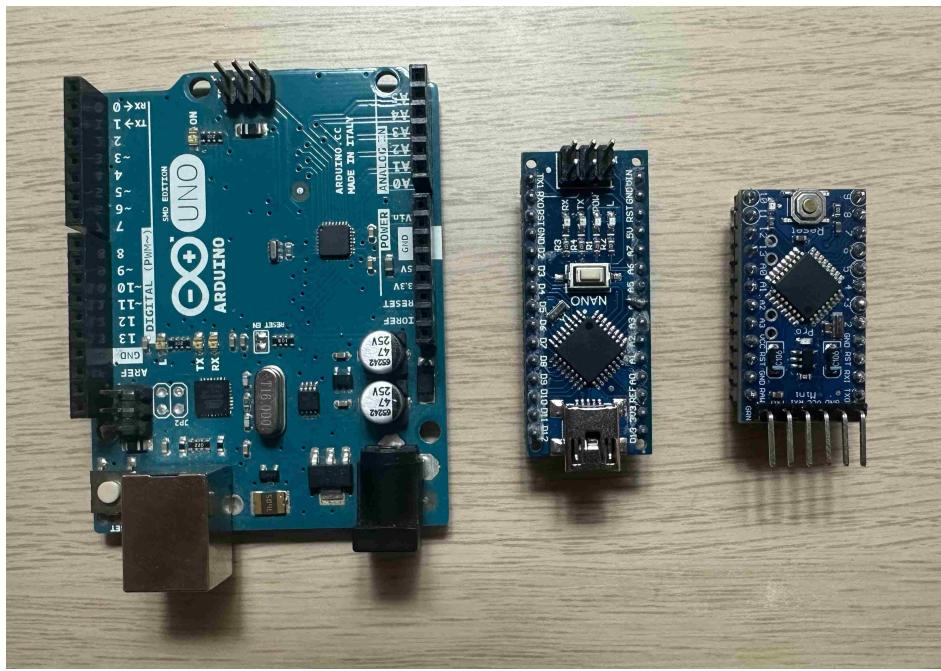


## **2. YAPILAN ARDUINO ÇALIŞMALARI**

### **2.1. Arduino Hakkında Genel Bilgiler**

Mühendislik çeşitli alt disiplinlerden (e.g., bilgisayar mühendisliği, elektrik-elektronik mühendisliği, makine mühendisliği, *vb.*) olılsa da mühendisler olarak bugün pratik bir proje yaptığımızda multidisipliner olarak çalışmak zorundayız. Sadece elektrik-elektronik mühendisliğinde değil neredeyse bütün alanlarda (hatta bazen sosyolojik meselelerde bile) ölçüm cihazı ve sensörler işin içeresine girdiğinde, yapılan hesaplamalarda bir beyin vazifesi gören hafif, küçük ve ucuz çipler olan mikrodenetleyicilere (microcontroller) ihtiyaç duyulmaktadır. Son on beş senede dünyada özellikle hobi projelerinde (*do it yourself - DIY*) en çok tercih edilen mikrodenetleyiciler Atmel şirketinin üretmiş olduğu 8-bit ATmega328 çipli Arduino'lardır. Kullanım kolaylığıyla teknik bilgisi olmayan kullanıcıları (e.g., sanatçilar) bile kendisine çekmeyi başaran Arduino, son zamanlarda yüksek işlem hızlarından dolayı 32-bitlik çipler (e.g., STM32) piyasaya sürülmüşne rağmen hâlen birçok uygulamada kullanılmaktadır. En yaygın ve popüler Arduino modelleri olan Uno, Nano ve Pro-Mini'yi Şekil 1'de görebilirsiniz. Bu çalışmanın hem tasarım hem bitirme bölümlerinde kurulan devrelerde her üç Arduino da kullanılmıştır.

Arduino'lar sensörlerden veri okumayı, bu verilerden alakalı değerleri hesaplamayı, ve bu parametreleri/değişkenleri görselleştirmeyi veya eyleyicileri sürekli sinyallere dönüştürüp (i.e., kapalı bir çevrim tasarlayıp) geri-beslemeli bir kontrol sistemi (e.g., robot) gerçeklemeyi muazzam kullanıcı desteği ve profesyonel dökümantasyona sahip yüksek seviyeli C++ API'si sayesinde inanılmaz kolaylaştırarak insanlığın gelişimine büyük katkıda bulunmuştur. Kendilerinden sonra gelen single board computer olarak geçen tek kartlı bilgisayarlar (e.g., Raspberry Pi, NVidia Jetson Nano) her ne kadar kullanım alanları daha çok bilgisayarlı görü (computer vision - CV), makine öğrenmesi (machine learning - ML) ve derin öğrenme (deep learning - DL) uygulamaları olsa da Arduino'ları tahtından edememiştir. Artık Arduino'lar da TinyML sayesinde sınırlı işlem gücüne sahip de olsa yapay zekâ (artificial intelligence - AI) uygulamaları koşturabilmektedir.



Şekil 1. Arduino Uno, Nano ve Pro Mini.

## 2.2. Arduino Projeleri

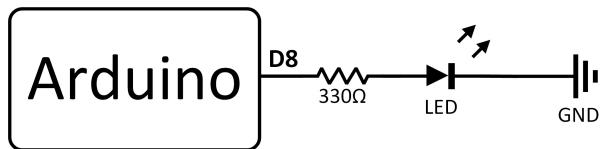
Bu bölümde EEM 216 dersinde yaptığımız örnek Arduino deneylerinin devre bağlantı şemalarını ve yazılımlarını bulabilirsiniz.

### 2.2.1. LED Yakma Devresi

Bu deneyde Arduino'nun dijital portunda tek bir bacak (pin) kullanacağız. Yukarıda Şekil 1'de ismi geçen Arduino modellerinde dijital port 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 olarak numaralandırılmıştır. İlk iki bacak olan D0 ve D1 pinlerinin üzerinde Rx ve Tx yazdigından genelde dijital port kullanan uygulamalar bu bacakları kullanmazlar. Burada D8 bacağını çıkış (output) olarak seçelim ve bir döngü içerisinde D8'i belirli bir süre yakalım ve söndürelim. Doğal olarak D8'i LED'e direkt değil bir direnç üzerinden bağlamalıyız ki LED'i yüksek akımdan dolayı yakmayıalı. Devre bağlantı şeması ve kodunu Şekil 2'te görebilir, koda ilgili kod deposundan<sup>1</sup> erişebilirsiniz.

---

<sup>1</sup><https://github.com/mtahakoroglu/gumushane-eem-kodlama>

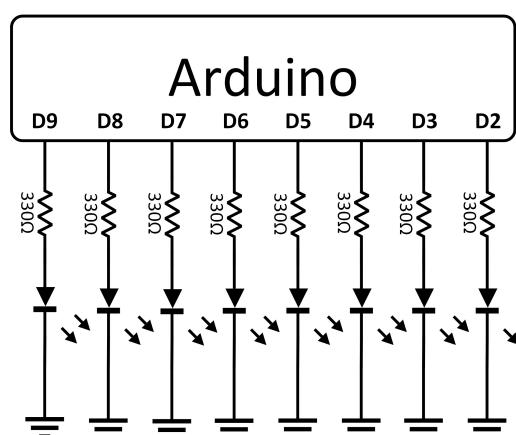


```
led_flash.ino
#define LED_PIN 8
void setup() {
    pinMode(LED_PIN, OUTPUT); // çıkış olarak seç
}
void loop() {
    digitalWrite(LED_PIN, HIGH); // LED'i yak
    delay(500); // ms cinsinden gecikme
    digitalWrite(LED_PIN, LOW); // LED'i söndür
    delay(500);
}
```

Şekil 2. LED yakma devresi.

### 2.2.2. Flaşör Devre

Sıradaki deneyde Arduino'nun dijital portunda sekiz bacağı birden çıkış olarak kullanacağız. Bu bacakları 2, 3, 4, 5, 6, 7, 8, 9 olarak seçelim. Algoritmik olarak önceki projeye göre farklı olan en önemli şey, kullanılan dijital pin sayısı birden fazla olduğu için bacakların kontrolünü **for** döngüleriyle gerçekleştiriyor olacağımızdır. Devrenin bağlantılarını ve kodunu Şekil 3'te görebilir, koda ilgili kod deposundan<sup>2</sup> erişebilirsiniz.



```
// D2'den D9'a kadar olan pinler, seçelim
int ledPins[] = {2, 3, 4, 5, 6, 7, 8, 9}; // 8 LED için pinler
int numLeds = 8; // LED sayısı
int f = 2; // frekans | saniyede kaç kere tur atsın
int delayTime = (1000/f)/(2*numLeds-2); // ms

void setup() {
    // LED pinlerini çıkış olarak ayarlıyoruz
    for (int i=0; i<numLeds; i++) {
        pinMode(ledPins[i], OUTPUT);
    }
}

void loop() {
    // LED'leri D2'den D9'a doğru yak
    for (int i=0; i<numLeds; i++) {
        digitalWrite(ledPins[i], HIGH); // LED'i yak
        delay(delayTime);
        digitalWrite(ledPins[i], LOW); // öbür LED'e geçmeden LED'i söndür
    }
    // LED'leri D9'dan D2'ye doğru geri yak (D9 ve D2 dâhil değil)
    for (int i=numLeds-2; i>0; i--) { // numLeds-2 çünkü D9 tekrar yanmamalı
        digitalWrite(ledPins[i], HIGH); // LED'i yak
        delay(delayTime);
        digitalWrite(ledPins[i], LOW); // öbür LED'e geçmeden LED'i söndür
    }
}
```

Şekil 3. Flaşör devre.

<sup>2</sup><https://github.com/mtahakoroglu/gumushane-eem-kodlama>

### 2.2.3. Kare Dalga Üreteci

Buraya kadar Arduino'nun dijital portundaki pinleri **setup()** fonksiyonu içinde **pinMode()** fonksiyonuyla çıkış (**OUTPUT**) olarak seçmeyi ve ardından **loop()** fonksiyonu içinde **digitalWrite()** fonksiyonuyla seçili bacaklarda çıkış olarak logic 0 (**LOW**) ve logic 1 (**HIGH**) sinyalleri üretmeyi başardık. Bu yeni deneyde dijital portu çıkış olarak seçmeye ek olarak analog giriş kullanacağız.

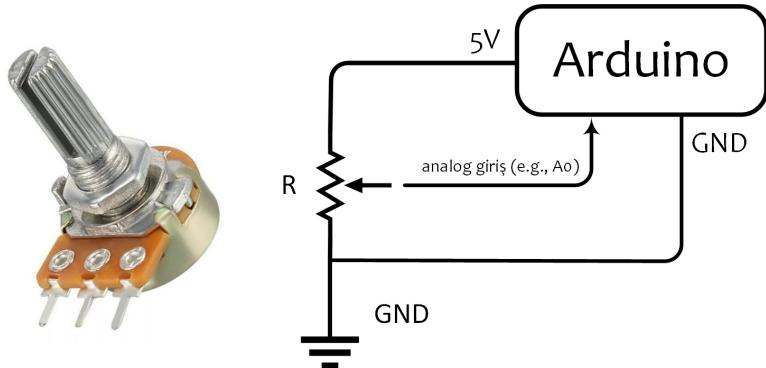
Ayarlı direnç olarak bilinen potansiyometre kullanarak Arduino'ya analogdan dijital dönüştürücü yardımıyla transfer edeceğimiz [0-5]V aralığındaki gerilim değeriyle D13 dijital pininde üreteceğimiz bir kare dalga sinyalinin frekansını (periyodunu) değiştireceğiz ve sinyal jeneratörü yapacağız. Oluşturduğumuz bu sinyali hem osiloskoba hem hoparlöre (veya buzzer'a) bağlayarak görsel ve işitsel olarak da gözlemleyeceğiz.

Devrelerde gerilim bölücü (voltage divider) olarak kullanılan potansiyometre;

- Üç bacağa sahiptir.
- Uç bacakları (yâni orta bacak dışındakiler) sıra gözetmeksizin Arduino'nun Vcc (5V) ve GND pinlerine bağlandığında orta bacağı [0-5]V aralığında analog bir değer üretir.
- Orta bacağı Arduino'ya analog girişlerden biriyle (e.g., A0) bağlandığında analogdan dijital çevirici (analog to digital converter - ADC) aracılığıyla [0-5V] aralığında sürekli (continuous) zamanlı analog sinyal, [0-1023] aralığında bir tam sayıya (integer) doğrusal olarak dönüştürülerek (linear mapping) ayrık (discrete) bir değer elde edilir. Artık Arduino'da bu tam sayı değeri kullanılarak birçok şey (e.g., joystick ile kanal/motor sinyali üretme) yapılabilir.

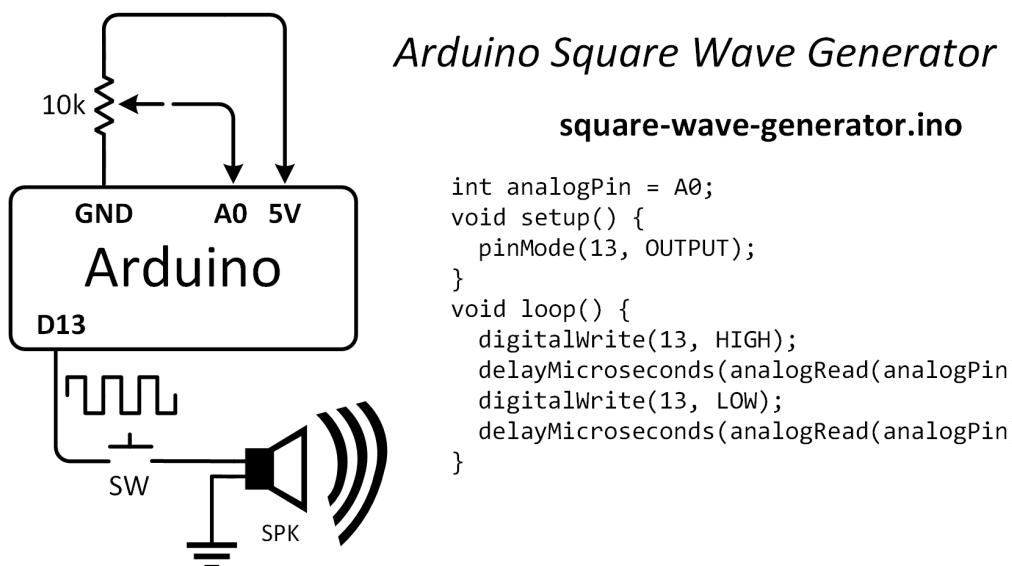
Bilindiği gibi potansiyometreler, iç mekân ışıklandırma sistemlerinde ışık şiddetini ayarlama, elektronik aletlerde ses düzeyini değiştirme, RC model araba/uçak kontrol eden uzaktan kumandalarda kanal sinyallerini oluşturma gibi pek çok önemli uygulamada vazife yapmaktadır. Örnek bir potansiyometreyi Şekil 4'de görebilirsiniz.

En sol ve en sağ bacakları daha önce de belirttiğimiz gibi sıra farketmeksizin GND ve 5V'a, orta bacağı ise Arduino'nun A0 analog giriş pinine Şekil 4'deki gibi bağlayıp yukarıda bahsedilen ADC işlemini gerçekleştireceğiz.



Şekil 4. Örnek bir potansiyometre ve Arduino bağlantıları.

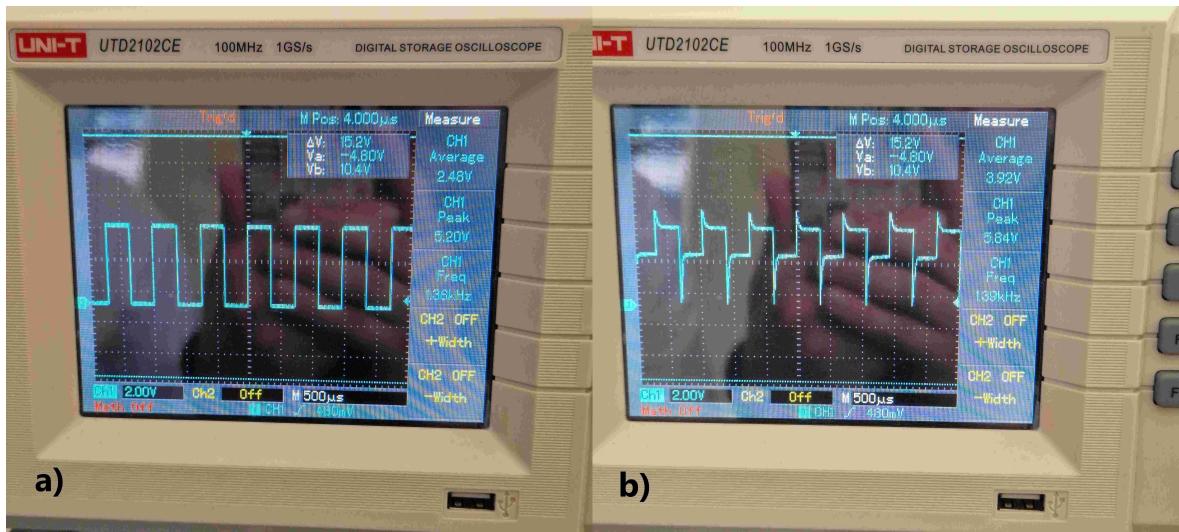
Buradaki deneyde D13 pininde ürettiğimiz sinyali hoparlör veya buzzer'a üstten basmalı anahtar (push-button switch) üzerinden bağlamak isabetli olacaktır zira sürekli sinyalin oluşturduğu sesi duymak istemiyoruz. Yâni anahtara bastığımız zamanlarda sesi duyacağız. Şekil 5'de kare dalga üreteci devre şemasında devre elemanları arasındaki bağlantıları ve Arduino kodunu görebilirsiniz.



Şekil 5. Kare dalga üreteci devre şeması ve kodu.

Şekil 5'te D13 no'lu dijital pinde üretilen kare dalga sinyalini osiloskopun probuna bağlayıp da gözlemlersek hoparlör/buzzer aracılığıyla duyduğumuz seste potansiyometre aracılığıyla meydana gelen frekans/periyot değişimini Şekil 6'daki gibi gözlemleriz. Şekil 6'te a) ile gösterilen sinyal, üstten basmalı anahtar açık devreyken (yâni anahtara basılmadığında) görüntülenen sinyaldir. Bu durumda hoparlör/buzzer devreye yük oluşturmadığından akım çekmeyecek ve dolayısıyla gerilim düşümüne (güç transferine) sebebiyet vermeyecektir. Sonuç olarak D13'de üretilen kare dalga sinyali aynen teorikte

olduğu gibi mükemmel bir biçimde (yâni bozulmadan) gözlemlenebilmiştir. Öte yan- dan push-button SW'e basıldığı anda devre tamamlandığı için hoparlör/buzzer akım çekecek ve oluşan gerilim düşümü (güç transferi) sonucu b) ile gösterilen şeke sahip bir kare dalga oluşacaktır. Anahtara basıldığında sinyalde gözlemlenen bu bozulmanın yaşanmaması için transistörler veya işlemel yükselteç (operational amplifier yâni op-amp) yardımıyla tampon (buffer) devresi kurularak D13 ve hoparlör/buzzer arasına konabilir.



Şekil 6. Kare dalga sinyalinin osiloskop ekranındaki görüntüsü.

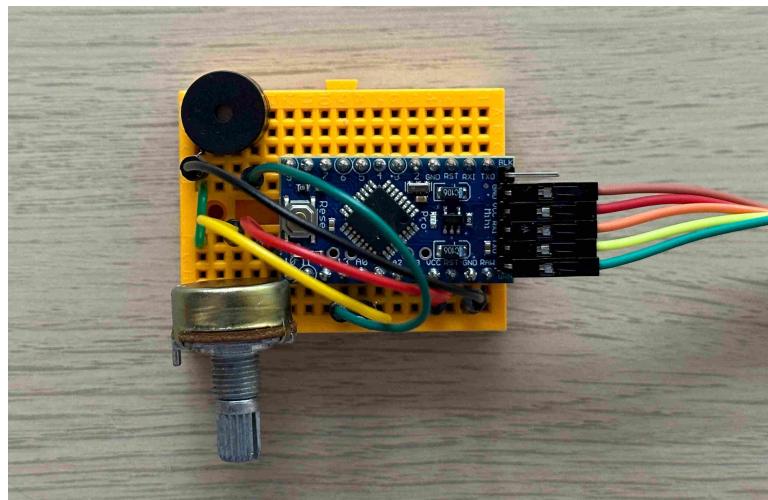
Kare dalga sinyali üretici deneyinin Arduino ve devre elemanlarıyla kurulmuş hâlini Şekil 7'de görebilirsiniz. Yapılan deneyin videosuna da ilgili kod deposundan ulaşabilirsiniz.<sup>3</sup> Burada Yapay Zekâ'dan faydalanan Python'da **matplotlib** paketi kullanılarak bilgisayar ekranında osiloskop işlevi görecek bir kod oluşturulabilir.

#### 2.2.4. Dijital Giriş Okuma

Bu bölümde ikinci deney olarak yaptığımız flaşör devre çalışırken üstten basmalı anahtar yardımıyla D10 pininden dijital bir sinyal okuyup animasyon hâlinde yanan LED'leri durdurmak veya donmuş hâldeki LED animasyonunu devam ettirmek istiyoruz. Bunu yapmak için D10 pinine ait **pull-up** rezistörünü **setup()** fonksiyonunun içinde

```
pinMode(buttonPin, INPUT_PULLUP);
```

<sup>3</sup><https://www.youtube.com/shorts/tjAw2EqMBL8>

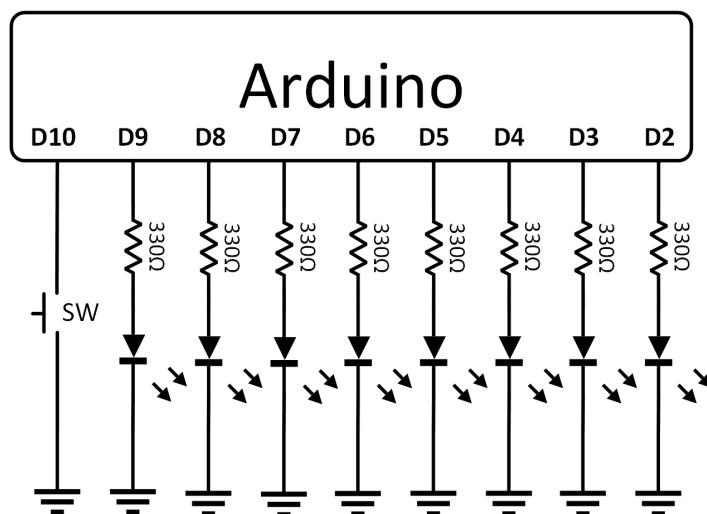


Şekil 7. Kare dalga sinyali jeneratörü devresi.

fonksiyonunu çağrıarak aktif hâle getirmek ve ardından **loop()** fonksiyonunun hemen ilk satırında

```
digitalRead(buttonPin)
```

fonksiyonu çağrıarak sürekli D10'dan gelecek olan sinyali Arduino'ya okumamız lâzım. Burada **buttonPin** isimli değişken 10 olarak önceden tanımlanıyor (yâni D10 pini). Şekil 8'da devre bağlantılarını görebilir ve kodun tamamını aşağıda bulabilirsiniz.



Şekil 8. Kare dalga sinyali jeneratörü devresi.

```
1 const int ledPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
2 const int numLeds = 8;
3 const int buttonPin = 10;
```

```

4
5 int currentLed = 0; // Sunulan LED konumu
6 int direction = 1; // 1: ileri, -1: geri
7 bool flashing = true; // LED yanıp sonme durumu
8 int buttonState; // Mevcut buton durumu
9 int lastButtonState = HIGH;
10 unsigned long lastDebounceTime = 0;
11 const unsigned long debounceDelay = 50;
12 int f = 2;
13 unsigned long delayTime = (1000 / f) / (2 * numLeds - 2);
14 unsigned long previousMillis = 0;
15
16 void setup() {
17     for (int i = 0; i < numLeds; i++) pinMode(ledPins[i],
18         OUTPUT);
19     pinMode(buttonPin, INPUT_PULLUP);
20 }
21
22 void loop() {
23     int reading = digitalRead(buttonPin);
24
25     // Debounce kontrolu
26     if (reading != lastButtonState) {
27         lastDebounceTime = millis();
28     }
29
30     if ((millis() - lastDebounceTime) > debounceDelay) {
31         if (reading != buttonState) {
32             buttonState = reading;
33             if (buttonState == LOW) {

```

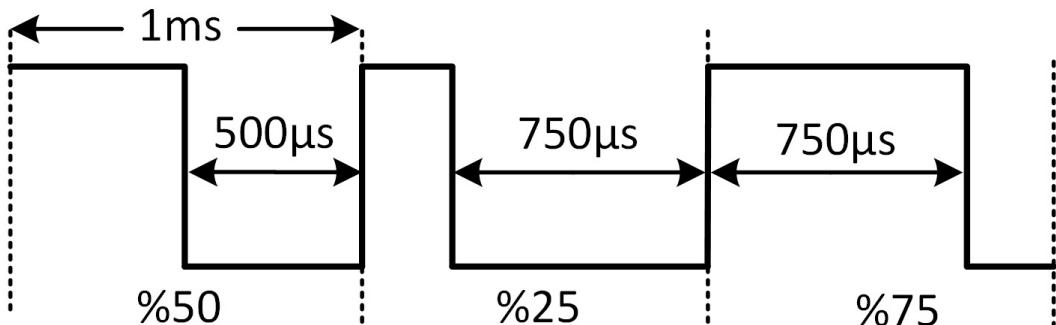
```

34         flashing = !flashing; // Butona basildiginda durdur/
            baslat
35     }
36 }
37 }
38
39 lastButtonState = reading;
40
41 // Eger flashing aktifse ve zaman dolduysa LED guncelle
42 if (flashing && (millis() - previousMillis >= delayTime)) {
43     previousMillis = millis();
44
45     // Onceki LED'i sondur
46     for (int i = 0; i < numLeds; i++) digitalWrite(ledPins[i
        ], LOW);
47
48     // Su anki LED'i yak
49     digitalWrite(ledPins[currentLed], HIGH);
50
51     // Sonraki LED'i belirle
52     currentLed += direction;
53
54     // Uclara ulastiginda yon degistir
55     if (currentLed >= numLeds) {
56         currentLed = numLeds - 2;
57         direction = -1;
58     } else if (currentLed < 0) {
59         currentLed = 1;
60         direction = 1;
61     }
62 }
```

### 2.2.5. PWM Sinyali ve Duty Cycle

Şu ana kadar dijital portun bacaklarını çıkış olarak seçmeyi **digitalWrite()**, giriş olarak seçmeyi **digitalRead()** ve [0-5]V aralığından analog bir sinyali Arduino'ya giriş olarak (tam sayı şeklinde) okumayı da **analogRead()** fonksiyonu ile gerçekleştirdik. Dijital ve analog port isminin direkt olarak geçtiği komutlardan sadece **analogWrite()** fonksiyonunu kullanmadık.

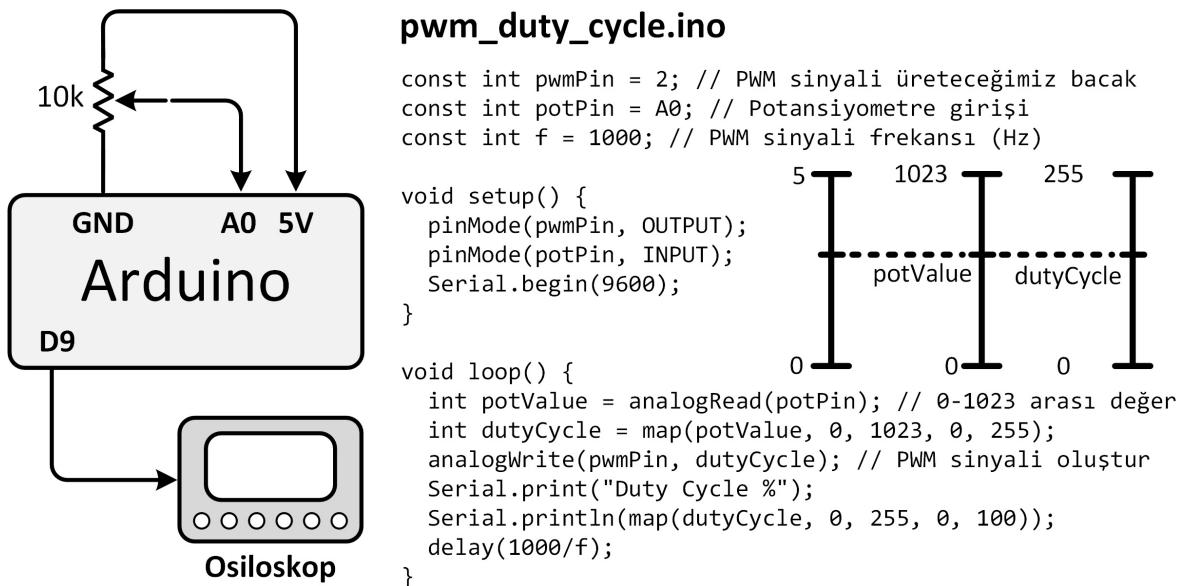
Robotikte karşımıza çıkan motorlardan bazıları DC motor, adım (step) motor, servo motor ve fırçasız (brushless) motorlardır. DC motorları sürmek için gereken akım veya gerilim değerini analog olarak üretmek yerine bu etkiyi oluşturacak (aynen Devre 2 Lab. dersinde gördüğümüz etkin değer hesabı gibi) bir dijital sinyal oluşturulmak yaygındır. Darbe Genişlik Modülasyonu (Pulse Width Modulation - PWM) denilen bu sinyalde logic 1 (+5V) süresinin logic 0 (GND) süresine oranını (periyodun içinde olara düşünün) artırıp azaltarak motora uygulanan voltajın ağırlıklı ortalamasını değiştirebilmekte ve böylece motorun dönüş hızını kontrol edebilmekteyiz. Frekansı 1kHz olan (yani periyodu 1ms) örnek bir PWM sinyalini ve farklı duty cycle değerlerini Şekil 9'da görebilirsiniz.



Şekil 9. PWM sinyali değişken duty cycle değerleri.

İlgili devre ve kod Şekil 10'da görülmektedir. Kodu çalıştırdıktan sonra osiloskop yoksa bile seri port ekranını açıp PWM sinyalinin duty cycle yüzdesini gözlemleyebilirsiniz.

## Arduino PWM Signal Generation – Duty Cycle



Şekil 10. PWM sinyali oluşturma (`analogWrite()` ile).

### 2.2.6. Potansiyometre ile Servo Motor Kontrolü

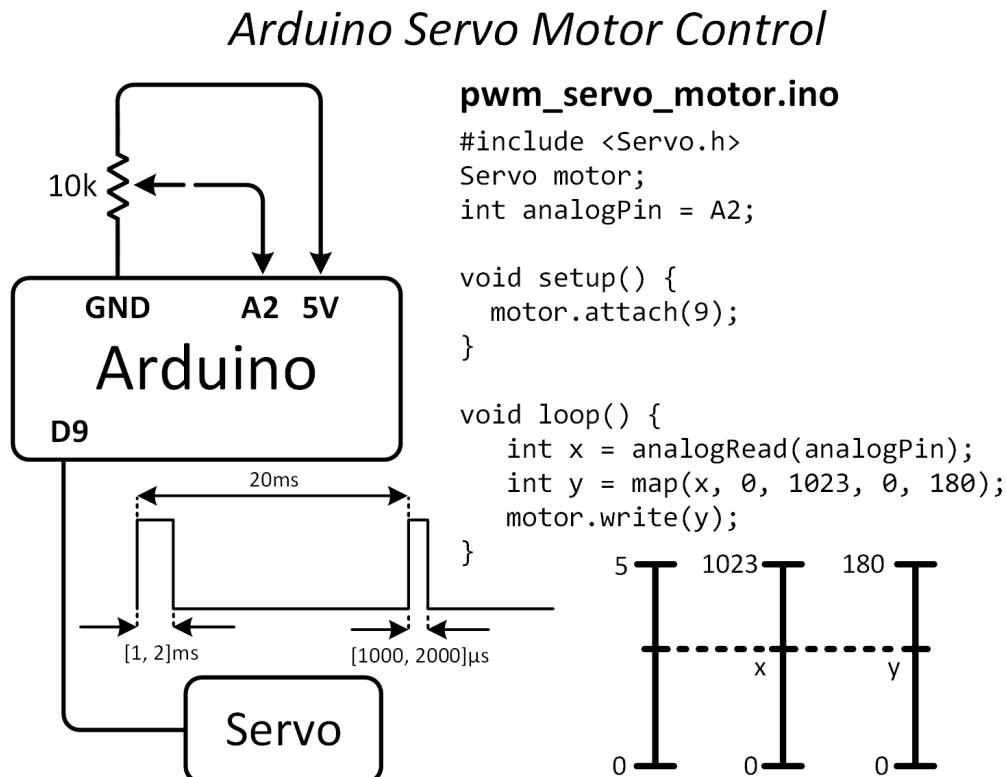
Remote Control (RC) ile kontrol edilen oyuncaklar ve cihazlarda (e.g., model uçak, araba, drone) sıkılıkla kullanılan servo ve firçasız (brushless) motorları kontrol etmek için Arduino'nun zamanlayıcılarını (timers) kullanarak özel olarak oluşturduğu darbe genişlik modülasyonu (pulse width modulation - PWM) denilen bir metot var. Buradaki PWM sinyalinin yapısı yukarıda `analogWrite()` fonksiyonuyla ürettiğimiz sinyalin yapısından farklı. Bazı yerlerde **kanal sinyali** olarak da geçiyor. Arduino kullanıcıları bu sinyali kolayca oluşturmak için özel bir kütüphane kullanıyorlar: **Servo**. Arduino üzerindeki dijital pinlerden 3, 5, 6, 9, 10, 11 no'lu olanlar (Arduino üzerinde ~ işaretinden anlayabilirsiniz) Servo kütüphanesi kullanarak PWM (kanal) sinyali üretmek için kullanılabilir. Şekil 11'de D9 pininin PWM çıkışının seçildiğini görebilirsiniz. Burada servo motorun açısal konumunu kontrol ediyoruz. Bunun için potansiyometreyi A2 girişinden

```
int x = analogRead(A2)
```

komutuyla okuyarak  $[0-5]V \rightarrow [0-1023]$  dönüşümünü yapıyoruz. Ardından Şekil 11'da da görülen

```
int y = map(x, 0, 1023, 0, 180)
```

doğrusal dönüşümüyle servo motora uyguluyoruz.



Şekil 11. PWM sinyali ile servo motor kontrolü.

Şekil 11'daki devre bağlantılarında görüldüğü gibi servo motorun kablolarından kahverengi olan Arduino'nun GND bacağına, orta kırmızı kablo +5V bacağına ve son olarak da turuncu kablo Arduino D9 bacağından üretilen PWM sinyaline bağlanıyor.

### 2.2.7. Joystick ile Fırçasız Motor Kontrolü

Bu kısımda bir önceki deneyde kullandığımız Servo kütüphanesi ile servo motora uyguladığımız PWM sinyalinin aynısını üretip drone'larda yaygın olarak tercih edilen bir fırçasız motora (brushless motor) uygulayacağız. Ancak bu sefer bu işi RC alıcı-verici (receiver-transmitter) devrelerinde karşımıza çok çıkan bir kavram olan **PWM sinyalinin ms veya us olarak nitelendirilmesi** kavramını öğrenmek amacıyla osiloskop ekranında PWM sinyalini gözlemleyerek yapacağız. Osiloskoba erişimin olmadığı durumlarda, üretilen PWM sinyalinin nümerik değerini kontrol edebilmek için sinyale tekabül eden değeri mikrosaniye cinsinden seri port ekranına da yazdıracağız.

Elimizde bulunan eski bir drone kumandasının içine bir tane Arduino Pro Mini koy-

duk. Drone kumandasında dört adet potansiyometre var. Yâni iki adet joystick ediyor. Bunlardan birisi drone'u aşağı-yukarı hareket ettirmek için kullanılan **THROTTLE** denilen kanal. Biz bu kanala tekabül eden potansiyometreyi yukarıda servo motor kodunda olduğu gibi A2 girişinden okuyup ardından yine benzer bir doğrusal dönüşümle ama bu sefer  $[0.9-2.1]\text{ms} = [900-2100]\mu\text{s}$  aralığına çevireceğiz.

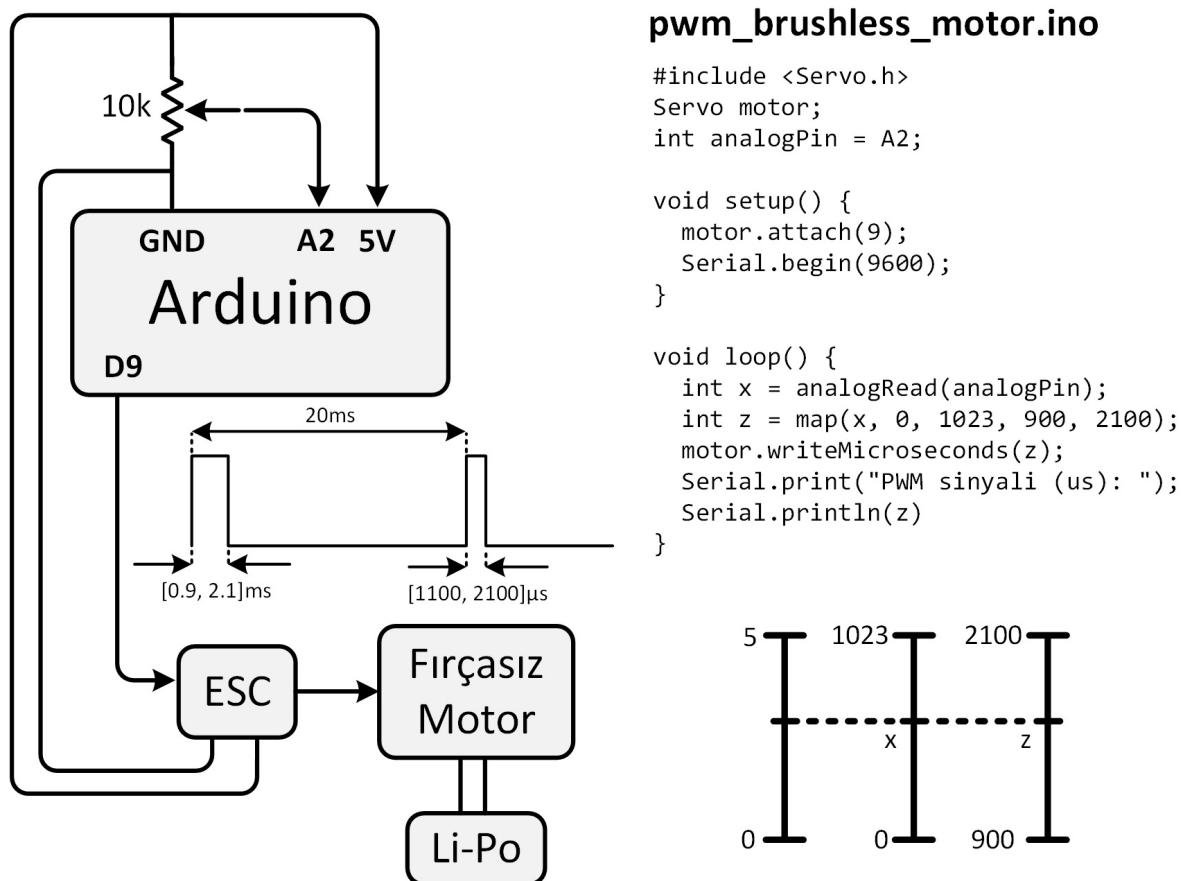
```
int z = map(x, 0, 1023, 900, 2100)
```

Son olarak da kanal sinyalini mikrosaniye birimi cinsinden oluşturacağız.

```
motor.writeMicroseconds(z)
```

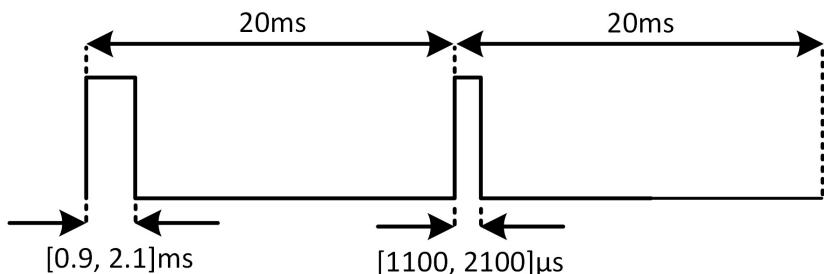
Böylece servo motora uyguladığımız PWM sinyalinin benzerini bu sefer bir fırçasız motorun ESC denilen motor sürücü devresine bağlayarak joystick veya potansiyometre aracılığıyla fırçasız motorun hızını kontrol edeceğiz. Şekil 12'de devre bağlantılarını ve kodu görebilirsiniz.

### *Arduino Brushless Motor Control*



Şekil 12. PWM sinyali ile fırçasız motor kontrolü.

Servo kütüphanesiyle hem servo motor için hem firçasız motor için PWM sinyali ürettiğimizde aslında sinyalin formu her zaman Şekil 13'deki gibi olmaktadır. PWM sinyalının minimum ve maksimum değerleri bazı alıcı-vericilerde [1000-2000]μs aralığında (e.g., Flysky), bazlarında [1100-1900]μs aralığında (e.g., Spektrum), bazlarında ise [900-2100]μs aralığında olabilmektedir.<sup>4</sup> Kullanılan alıcı-verici markasına göre uygun aralıkta sinyal üretmek gerekmektedir.



Şekil 13. Servo ve firçasız motor kontrolü için üretilen örnek bir PWM sinyali.

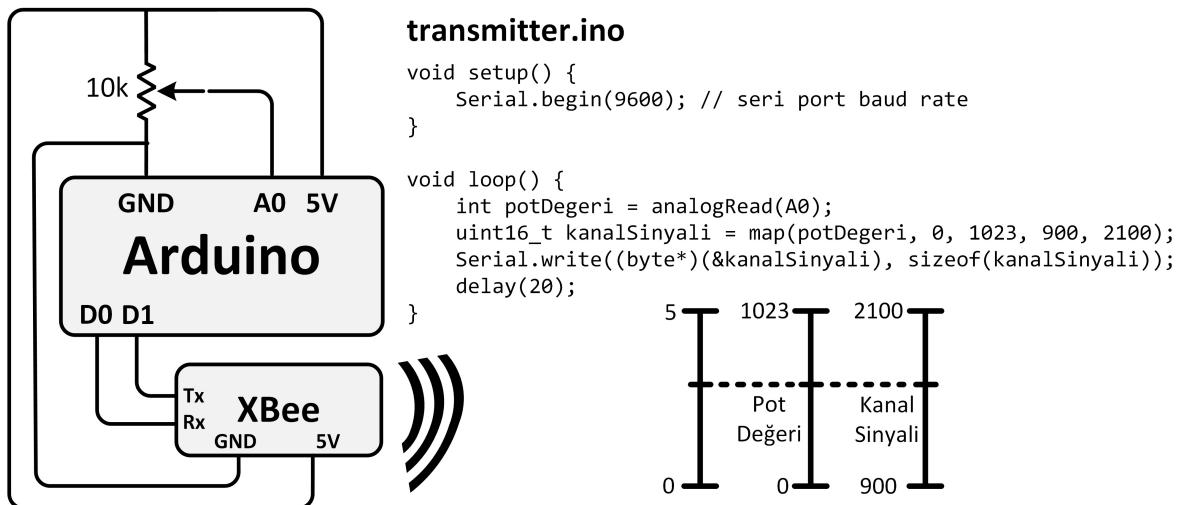
### 2.2.8. XBee Kablosuz Modül ile Firçasız Motor Kontrolü

Bir önceki deneyde potansiyometre (veya joystick) ile okuduğumuz analog voltaj değerlerini kanal sinyali hâline getirip Arduino'nun PWM sinyali üretebilen bacaklarından biri olan D9 üzerinden firçasız motora bağlı Electronic Speed Controller (ESC) ünitesine uygulamış ve Li-Po batarya ile güç verdigimiz firçasız motor ve pervane ikilisinin hızını kontrol etmiştik. Burada aynı işi bu sefer kablosuz (wireless) olarak yapmak istiyoruz. Bunun için potansiyometreden veri okuyup kanal sinyali hâline getirdiğimiz kısmı verici (transmitter), kanal sinyalinden PWM sinyalini oluşturup ESC'ye uyuladığımız kısmı da alıcı (receiver) olarak ayarlayacağız. Ardino'lara kolayca entegre olabilen XBee kablosuz RF modül Arduino'nun seri portu ile uyumlu çalıştığından kablosuz veri gönderip almada kodumuz karmaşık hâle gelmeyecek.<sup>5</sup> Aynı sadelikteki bir alıcı ve verici koduya bu işi yapacağız. Şekil 14'te verici devresinin bağlantılarını ve kodunu, Şekil 15'te ise Li-Po batarya ile güç verilerek çalıştırılan bir Arduino Nano ve XBee verici devresinin fotoğrafını görebilirsiniz.

<sup>4</sup>Bu değerler servo motor için PWM sinyali üretmede kullandığımız `write()` fonksiyonunda daha farklı bir aralıktaydı.

<sup>5</sup>Kablosuz haberleşmeyi XBee'lere göre çok daha ucuz olan nRF24l01 modülü ile yaparsanız kodun daha karmaşık hâle geldiğini görebilirsiniz. Ayrıca pratikte nRF24l01'ler XBee'lere kıyasla daha fazla sorun çıkarmaktadır. Yani performans olarak XBee'ler nRF24l01'lere göre daha üstündür.

## XBee Wireless Brushless Motor Control - TRANSMITTER

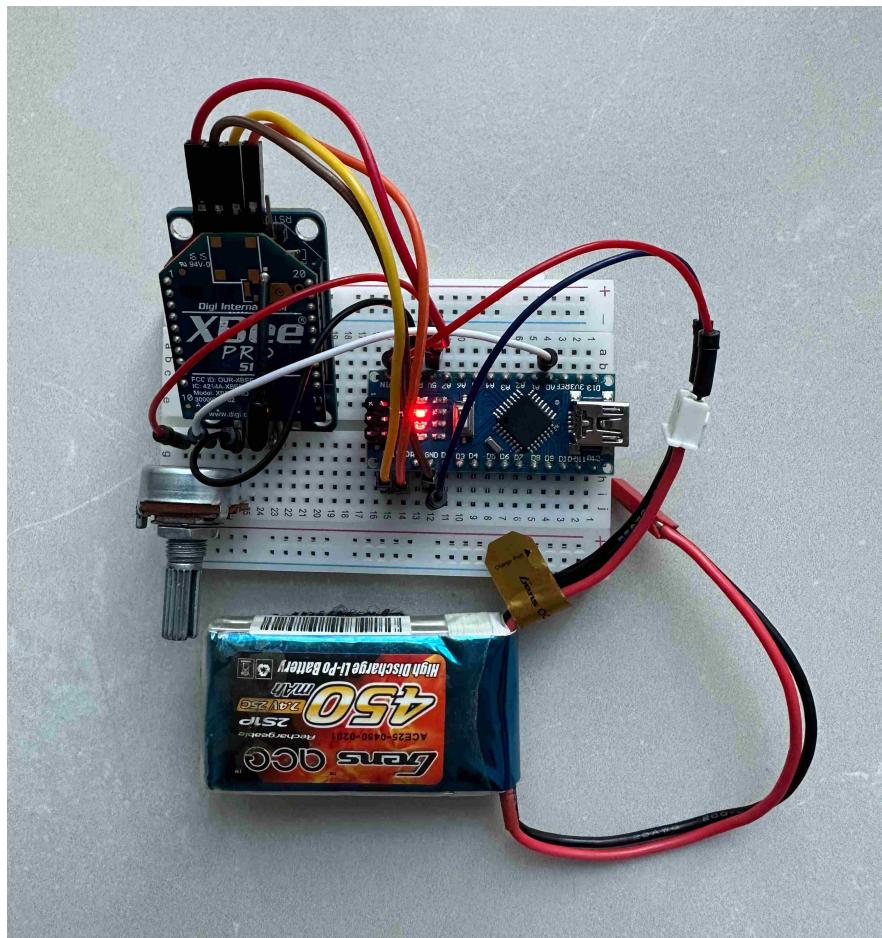


Şekil 14. Kablosuz fırçasız motor kontrolü verici devresi bağlantıları ve kodu.

Şekil 14'teki konfigürasyonda XBee kablosuz modülünden veri göndermek için seri port kullanıldığından Şekil 15'te mecburen Li-Po batarya ile devreye güç verdik. Eğer USB ile güç verseydik devrenin XBee'den kablosuz biçimde kanal sinyalini yollaması için bu sefer Software Serial olan olarak bilinen kütüphaneyi kullanmamız gerekecekti ki bu işlem kodu biraz daha uzun yapacaktı. Ayrıca Şekil 14'teki D0 ve D1 pini bağlantılarını D2 ve D3 olarak değiştirmemiz de gerekecekti. Bu yüzden yukarıdaki konfigürasyonda kalacağız.

Alici (receiver) devresini ise bizim kurmamız gerekmiyor. Yine de burada bir bütünlük arz etmesi açısından onu da inceleyelim. Burada kullandığımız alıcı devresinde Arduino Uno kullandığımızdan dolayı XBee shield kullanarak XBee kablosuz modülü Arduino'ya bağladık. Fırçasız motoru süren ESC devresinin PWM sinyali bağlantısını yine önceki deneylerde olduğu gibi D9 no'lu pin üzerinden yapıp gücünü üç hücreli bir Li-Po batarya üzerinden sağlayacağız. Burada belirtmemiz gereken önemli bir pratik nokta var: Li-Po batarya ESC'ye bağlanınca ESC'nin sinyal kablolarından orta kablo olan kırmızı kablo otomatik olarak +5V olduğundan dolayı bu orta kabloya Arduino'dan ekstra olarak +5V bağlantısı çekmeye gerek yok.<sup>6</sup>. Son olarak da XBee shield bağlanmış Arduino Uno'muzu Vin bacağından Li-Po batarya'nın dengeleyici (balancer) kısmındaki gerilimden beslemek zorundayız. Eğer Li-Po bataryamız 2-3-4 hücreli ise bunda bir sorun olmayacağı ama daha az veya yüksek hücreli Li-Po'larda Arduino

<sup>6</sup>Servo motor kontrolü deneyinde orta kırmızı bacağa +5V bağlamıştık.



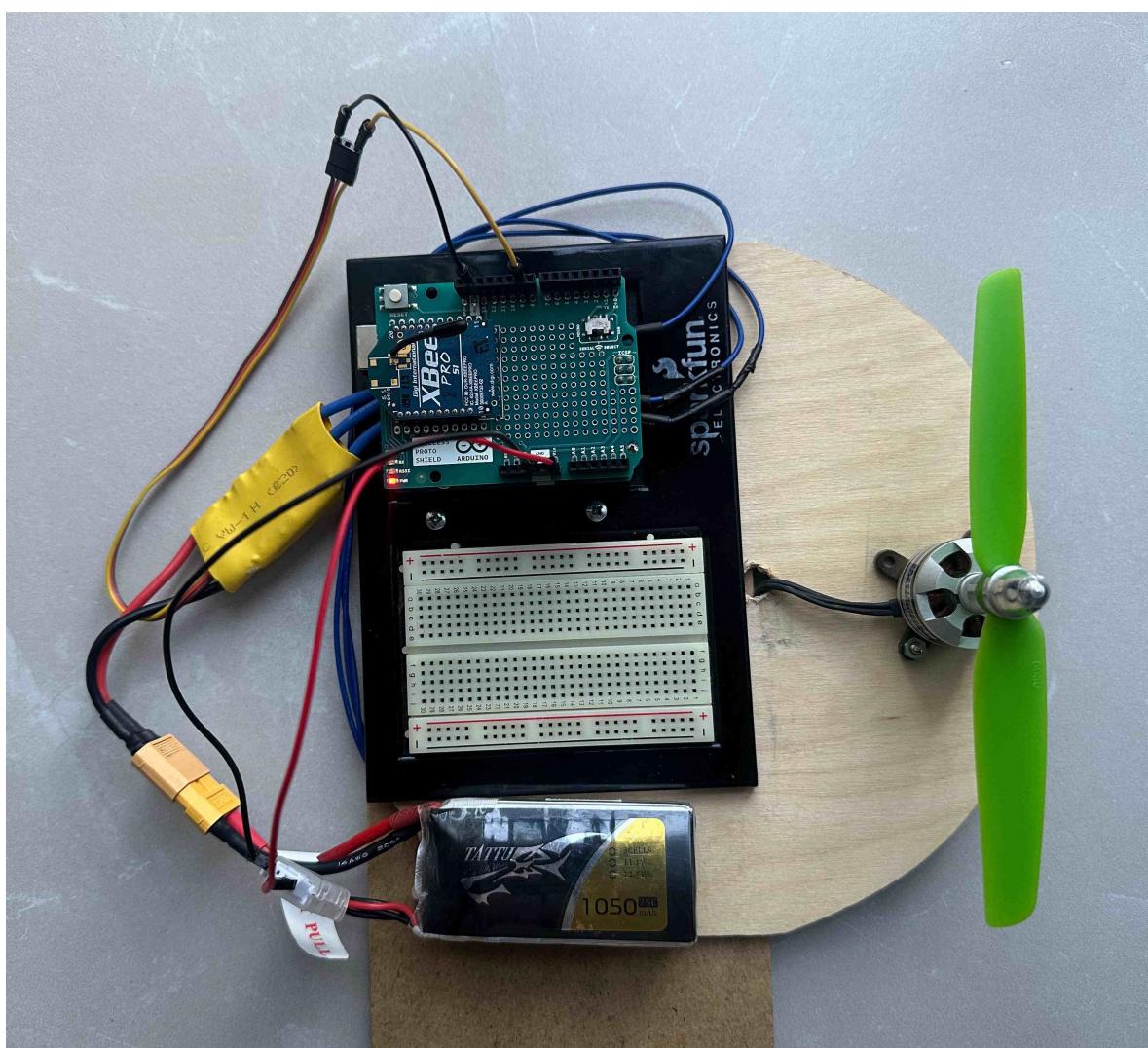
Sekil 15. Verici (transmitter) devresi.

veya batarya zarar görebilir. Şekil 16'te Li-Po batarya ile güç verilerek çalıştırılan bir Arduino Uno, XBee kablosuz modülü (shield ile bağlanmış hâlde) ve ESC tarafından sürülen firçasız motor - pervane ikilisinin fotoğrafını görebilirsiniz. Ayrıca aşağıda alıcı kodunu da bulabilirsiniz.

```
1 #include <Servo.h>
2
3 Servo motor;
4 const int motorPin = 9;
5 uint16_t kanalSinyali;
6
7 void setup() {
8     Serial.begin(9600);
9     motor.attach(motorPin);
```

```

10  motor.writeMicroseconds(900); // sinyal yoksa bunu uygula
11 }
12
13 void loop() {
14   if (Serial.available()) {
15     Serial.readBytes((byte*)(&kanalSinyali)), sizeof(
16       kanalSinyali));
17   motor.writeMicroseconds(kanalSinyali);
18 }
```



Şekil 16. Alıcı (receiver) devresi ve firçasız motor.