
# TOBB Ekonomi ve Teknoloji Üniversitesi

## ELE362 – Mikrodenetleyiciler ve Uygulamaları

## Proje



| | |
|---|---|
| Adı Soyadı | Erkin Coşkun Ruhi |
| Numara | 211201074 |
| Tarih | 08.04.2024 |

# 1. Projenin Amacı

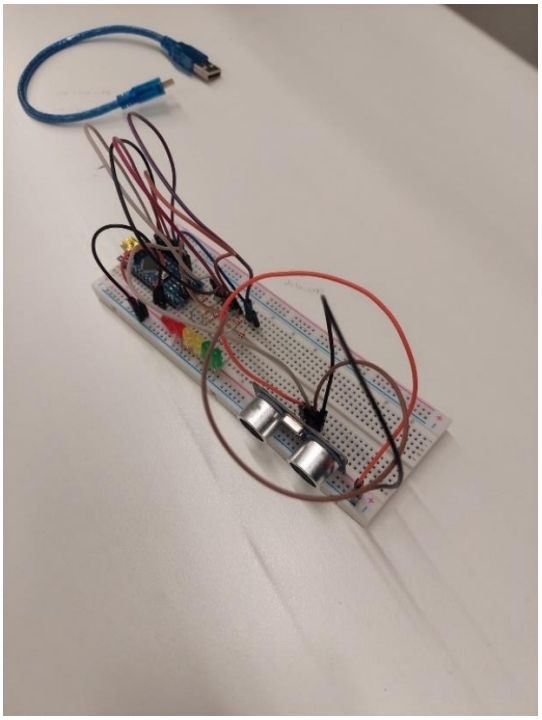
HC-SR04'den çıkan ses dalgasıyla mesafeyi hesaplayıp, uzaklığa göre ledleri yakıyoruz. Sensörün kapsadığı alana giren nesnelilerin sensörden çıkana kadar ki sensöre en yakın mesafesiyle birlikte kaç tane geçtiğini kaydedip USART'la seri port ekranına yazdırıyoruz.

# 2. Giriş Çıkış Pinleri

```
#define TRIG_PIN PD2
#define ECHO_PIN PD3
#define LED_GREEN1 PD4
#define LED_GREEN2 PD5
#define LED_YELLOW1 PD6
#define LED_YELLOW2 PD7
#define LED_RED1 PB0
#define LED_RED2 PB1

#define GREEN_LED_DDR DDRD
#define YELLOW_LED_DDR DDRD
#define RED_LED_DDR DDRB
#define TRIG_DDR DDRD
#define ECHO_DDR DDRD

#define TRIG_PORT PORTD
```

Şekil 1: Pinlerin Tanımı

Devremizde altı tane led kullandık bunların ikisi yeşil, ikisi sarı, ikisi kırmızı. Yeşil ledleri PD4 ve PD5'e, sarı ledleri PD6 ve PD7'ye, kırmızı ledleri ise PB0 ,PB1'e bağladık. HC-SR04'ünüzü ise Vcc kısmından +5V verdik gnd kısmını ise toprağa bağladık. Trig pinini PD2'ye Echo pinini ise PD4'e bağladık.



Şekil 2: Devremiz

```
void setup_io() {
    DDRD |= (1 << TRIG_PIN) | (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
    DDRB |= (1 << LED_RED1) | (1 << LED_RED2);
    DDRD &= ~(1 << ECHO_PIN);
    PORTD &= ~((1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2));
    PORTB &= ~((1 << LED_RED1) | (1 << LED_RED2));
}
```

Şekil 3: setup_io Fonksiyonumuz

Şekil 3'teki fonksiyonumuzda portd de bulunan Trig, yeşil ledler, sarı ledleri ve portb de bulunan kırmızı ledleri çıkış olarak ayarlıyoruz ve ledlerimizi kapalı bir şekilde başlatıyoruz. Portd de bulunan Echo pinimizi ise girdi olarak ayarlıyoruz.

# 3. Kod

```
#define F_CPU 16000000UL
#define SOUND_SPEED 0.0343
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

Şekil 4: Sabitler ve Kütüphaneler

İlk olarak F_CPU ve ses hızı sabitlerini tanımladık kodumuza sonrasında kodumuzda kullandığım kütüphaneleri tanımladım.

```
volatile uint16_t currentDistance = 0;
volatile uint16_t closestDistance = 0xFFFF;
volatile uint16_t objectCount = 0;
volatile uint8_t objectDetected = 0;
```

Şekil 5: Değişkenler

currentDistance: Bu değişken, son ölçülen mesafeyi saklar.

closestDistance: Bu değişken, algılanan nesneler arasında kaydedilen en yakın mesafeyi saklar.

objectCount: Bu değişken, algılanan nesnelerin sayısını tutar.

objectDetected: Bu, bir boolean türündeki değişkendir ve değeri ya sıfır ya da bir olabilir. Bir nesnenin algılandığını veya algılanmadığını gösterir.

```
unsigned int measureDistance() {
    PORTD |= (1 << TRIG_PIN);
    _delay_us(10);
    PORTD &= ~(1 << TRIG_PIN);

    unsigned int count = 0;
    while (!(PIND & (1 << ECHO_PIN)));

    TCNT1 = 0;
    TCCR1B |= (1 << CS11);

    while (PIND & (1 << ECHO_PIN)) {
        if (TCNT1 > 30000) {
            break;
        }
    }

    TCCR1B = 0;
    count = TCNT1 / 2;

    return (double)count * SOUND_SPEED / 2;
}
```

Şekil 5: measureDistance Fonksiyonu

measureDistance(): Ultrasonik bir mesafe sensörü kullanarak nesnelere olan mesafeyi ölçmemizi sağlar. Normal modda bir zamanlayıcı kullanıyoruz bu işlem için. Önce ultrasonik dalgaları göndermek için TRIG_PIN pinini yüksek duruma getirir ve 10 mikro saniye bekledikten sonra düşük duruma getirir. ECHO_PIN'den yüksek sinyal gelene kadar bekler, bu sinyal dalgaların nesneye çarpıp geri döndüğünü gösterir. Bu süreyi hafızaya kaydeder ve sayacı durdurup, toplam süreyi ikiye bölerek gerçek yansıma süresini hesaplar. Bu süreyi ses hızıyla çarparak mesafeyi santimetre cinsinden bulur. Bu fonksiyon, nesnelerin konumunu belirlemek için hassas mesafe ölçümleri sağlar.

```
void USART_Init() {
    UCSR0A = 0x00;
    UBRR0H = 0x00;
    UBRR0L = 0x67;
    UCSR0C &= ~((1<<UMSEL00)|(1<<UMSEL01));
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);
    UCSR0C = (1<<USBS0)|(1<<UCSZ00)|(1<<UCSZ01);
}
```

Şekil 6: USART_Init Fonksiyonu

USART_Init(): USART için yapılandırma işlemlerini gerçekleştirir. Bu fonksiyon, USART modülünü başlatmak için gerekli olan çeşitli kontrol ve durum kayıtlarını ayarlar. İletişim hızını belirlemek için UBRR0H ve UBRR0L kayıtları kullanılarak baud rate 9600 yapmak için 0x67 olarak ayarladı. UMSEL00 ve UMSEL01 bitleri sıfırlayarak eşzamansız mod ayarlanır. RXEN0 ve TXEN0 bitleri set edilerek alıcı ve verici işlevler etkinleştirilir. USBS0 biti ile durdurma bitinin sayısı ve UCSZ00 ile UCSZ01 ile 8 bitlik karakter boyutu ayarlayarak seri port üzerinden veri alışverişi için gerekli yapılandırmalar tamamlanır. Bu ayarlar sayesinde mikrodenetleyiciden Arduino'nun seri port ekranından veri alırız.

```
void USART_Transmit_Char(char c) {
    while (!(UCSR0A & (1<<UDRE0)));
    UDR0 = c;
}
```

Şekil 7: USART_Transmit_Char Fonksiyonu

USART_Transmit_Char(char c): Veri kayıtçısının boş olup olmadığını kontrol ederek bir karakteri USART üzerinden gönderir.

```
void USART_Send_Distance_and_Count(uint16_t distance, uint16_t count) {

    const char distancePrefix[] = "Yakin Mesafe: ";
    for (uint8_t i = 0; distancePrefix[i] != '\0'; i++) {
        USART_Transmit_Char(distancePrefix[i]);
    }

    if (distance == 0) {
        USART_Transmit_Char('0');
        } else {
        char buffer[6];
        uint8_t i = 0;
        while (distance > 0) {
            buffer[i++] = '0' + distance % 10;
            distance /= 10;
        }
        while (i > 0) {
            USART_Transmit_Char(buffer[--i]);
        }
    }

    const char countPrefix[] = " cm, Nesne Sayisi: ";
    for (uint8_t i = 0; countPrefix[i] != '\0'; i++) {
        USART_Transmit_Char(countPrefix[i]);
    }

    if (count == 0) {
        USART_Transmit_Char('0');
        } else {
        char buffer[6];
        uint8_t i = 0;
        while (count > 0) {
            buffer[i++] = '0' + count % 10;
            count /= 10;
        }
        while (i > 0) {
            USART_Transmit_Char(buffer[--i]);
        }
    }


    USART_Transmit_Char('\r');
    USART_Transmit_Char('\n');
}
```

Şekil 8: USART_Send_Distance_and_Count Fonksiyonu

USART_Send_Distance_and_Count(): USART üzerinden ölçülen mesafe ve nesne sayısını belirli bir formatla gönderir; önce mesafe ve ardından nesne sayısını içeren bilgileri seri port aracılığıyla iletişim kurulan cihaza aktarır.

```
void processMeasurement(uint16_t distance) {
    if (distance <= 200 && objectDetected == 0) {
        objectDetected = 1;
        closestDistance = distance;
        } else if (distance > 200 && objectDetected == 1) {

        objectDetected = 0;
        objectCount++;
        USART_Send_Distance_and_Count(closestDistance, objectCount);
        closestDistance = 0xFFFF;
        } else if (distance <= closestDistance) {

        closestDistance = distance;
    }
}
```

Şekil 9: processMeasurement Fonksiyonu

processMeasurement: Ölçülen mesafeye göre nesne algılama durumunu günceller, en yakın mesafeyi kaydeder ve belirli bir mesafe eşiğinin ötesine geçildiğinde nesne sayısını artırıp mesafe ve sayı bilgisini USART üzerinden gönderir.

```
ISR(TIMER1_COMPA_vect) {
    uint16_t distance = measureDistance();

    PORTD &= ~((1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2));
    PORTB &= ~((1 << LED_RED1) | (1 << LED_RED2));


    if (distance > 200) {

        } else if (distance > 100) {
        PORTD |= (1 << LED_GREEN1);
        } else if (distance > 60) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
        } else if (distance > 40) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1);
        } else if (distance > 20) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
        } else if (distance > 10) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
        PORTB |= (1 << LED_RED1);
        } else {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
        PORTB |= (1 << LED_RED1) | (1 << LED_RED2);
    }
}
```

Şekil 10: ISR(TIMER1_COMPA_vect) Fonksiyonu

ISR(TIMER1_COMPA_vect): Timer1 kullanarak kesme servis rutini, belirlenen zaman aralıklarında mesafeyi ölçer ve ölçülen mesafe değerine göre çeşitli LED'leri aktive ederek görsel bir uyarı sağlar; mesafe azaldıkça daha fazla LED yanar, böylece nesnenin yakınlığına göre bir görsel geri bildirim sunar.

```
void setLEDsBasedOnDistance(uint16_t distance) {

    PORTD &= ~((1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 << LED_YELLOW2));
    PORTB &= ~((1 << LED_RED1) | (1 << LED_RED2));

    if (distance <= 10) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
        PORTD |= (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
        PORTB |= (1 << LED_RED1) | (1 << LED_RED2);
        } else if (distance <= 20) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
        PORTD |= (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
        PORTB |= (1 << LED_RED1);
        } else if (distance <= 40) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
        PORTD |= (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
        } else if (distance <= 60) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
        PORTD |= (1 << LED_YELLOW1);
        } else if (distance <= 100) {
        PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
        } else if (distance <= 200) {
        PORTD |= (1 << LED_GREEN1);
    }
}
```

Şekil 11: setLEDsBasedOnDistance Fonksiyonu

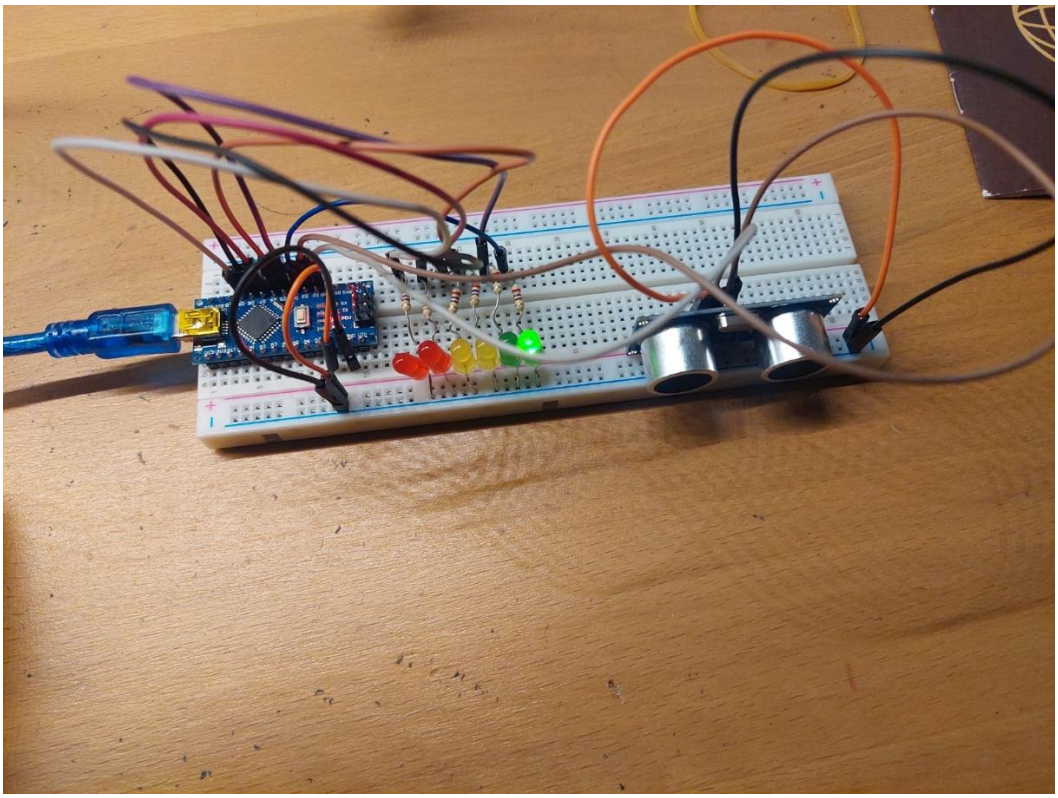
setLEDsBasedOnDistance: Ölçülen mesafeye göre farklı LED'leri aktive eder; mesafe azaldıkça daha fazla LED yanar ve böylece nesnenin yakınlığını görsel olarak gösterir.

```
int main(void) {
    setup_io();
    USART_Init();
    sei();

    while (1) {
        uint16_t distance = measureDistance();
        processMeasurement(distance);
        setLEDsBasedOnDistance(distance);

        _delay_ms(100);
    }

    return 0;
}
```

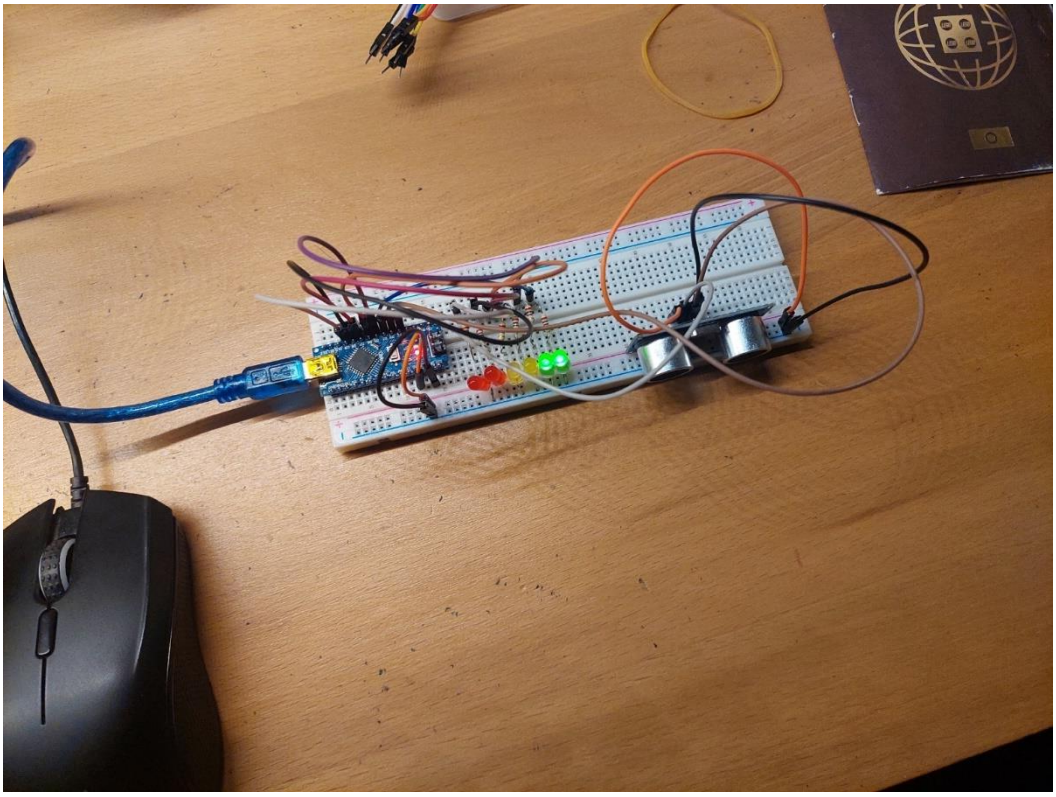
Şekil 12: main Fonksiyonu

main(void): Başlangıçta giriş, çıkış ayarlarını ve USART yapılandırmasını kurar, ardından kesmeleri etkinleştirir ve sürekli bir döngüde mesafe ölçer, bu mesafeyi işler ve ölçülen mesafeye bağlı olarak LED'leri ayarlar, processMeasurement ve setLEDsBasedOnDistance fonksiyonlarını çağırarak bir döngü kurar ve bu döngü her 100 milisaniyede bir tekrarlanır.

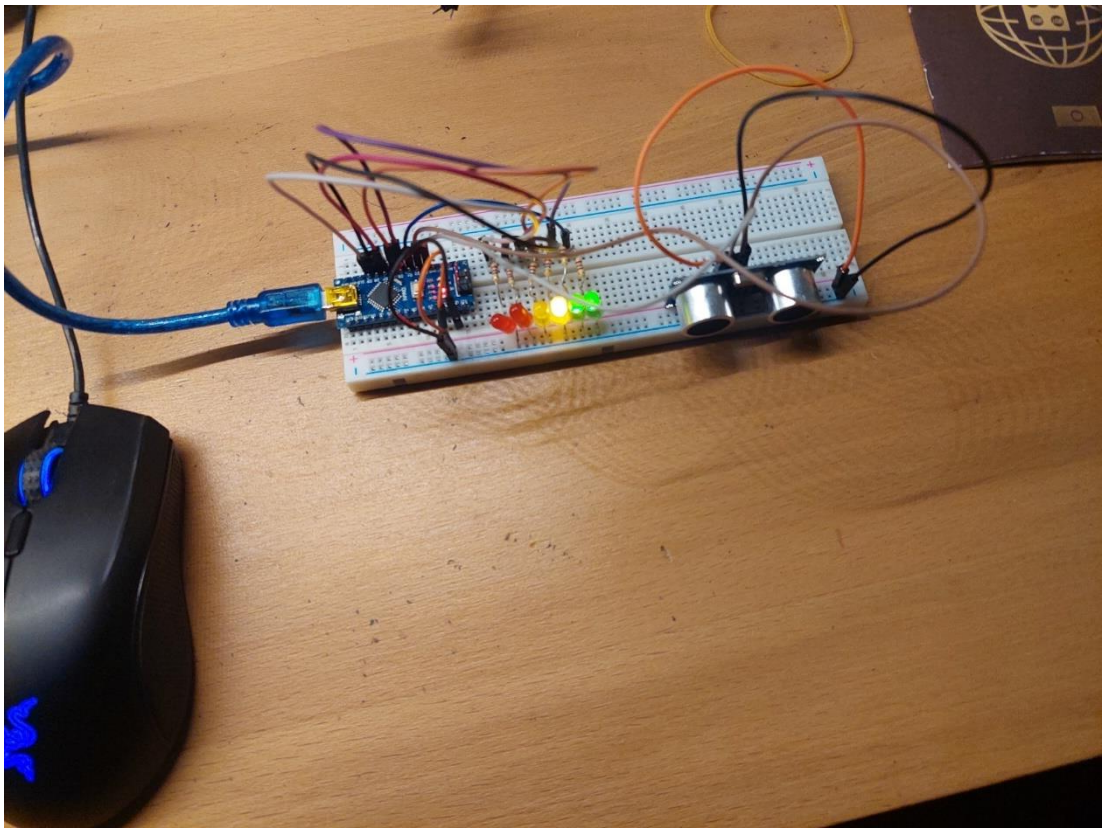
# 4. Çıktılar



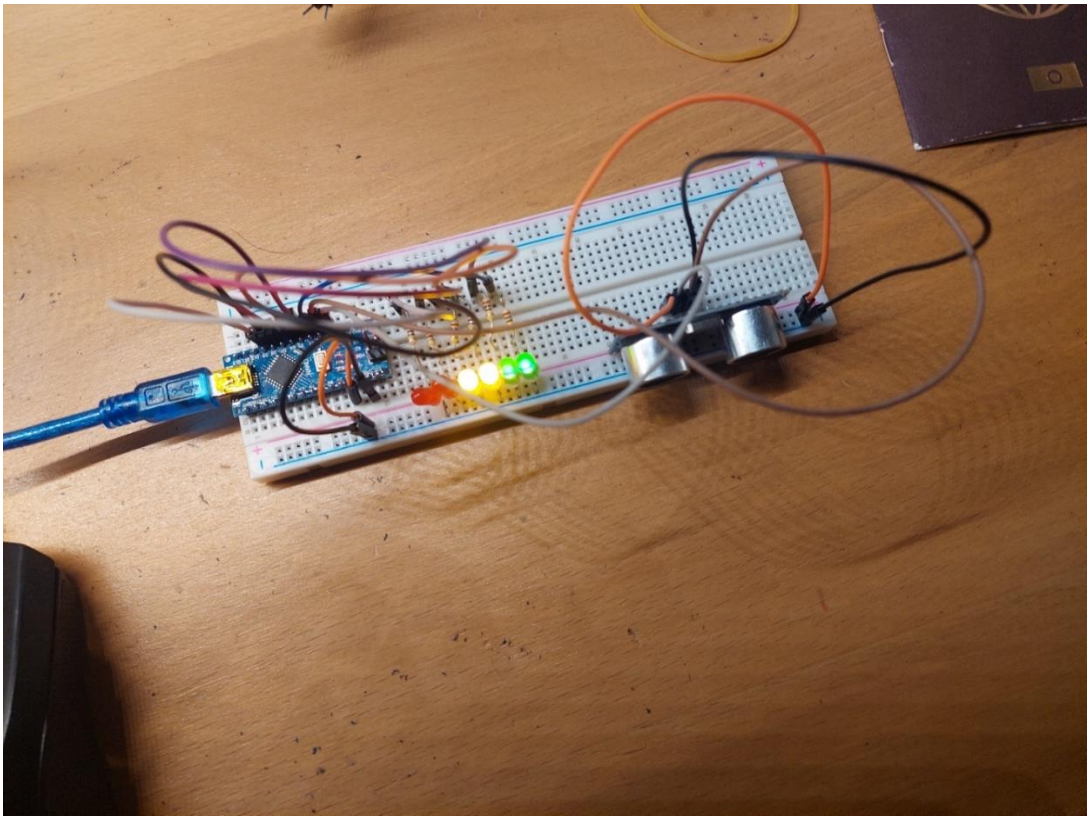
Şekil 13: 100-200 cm Arası 1 Yeşil Led



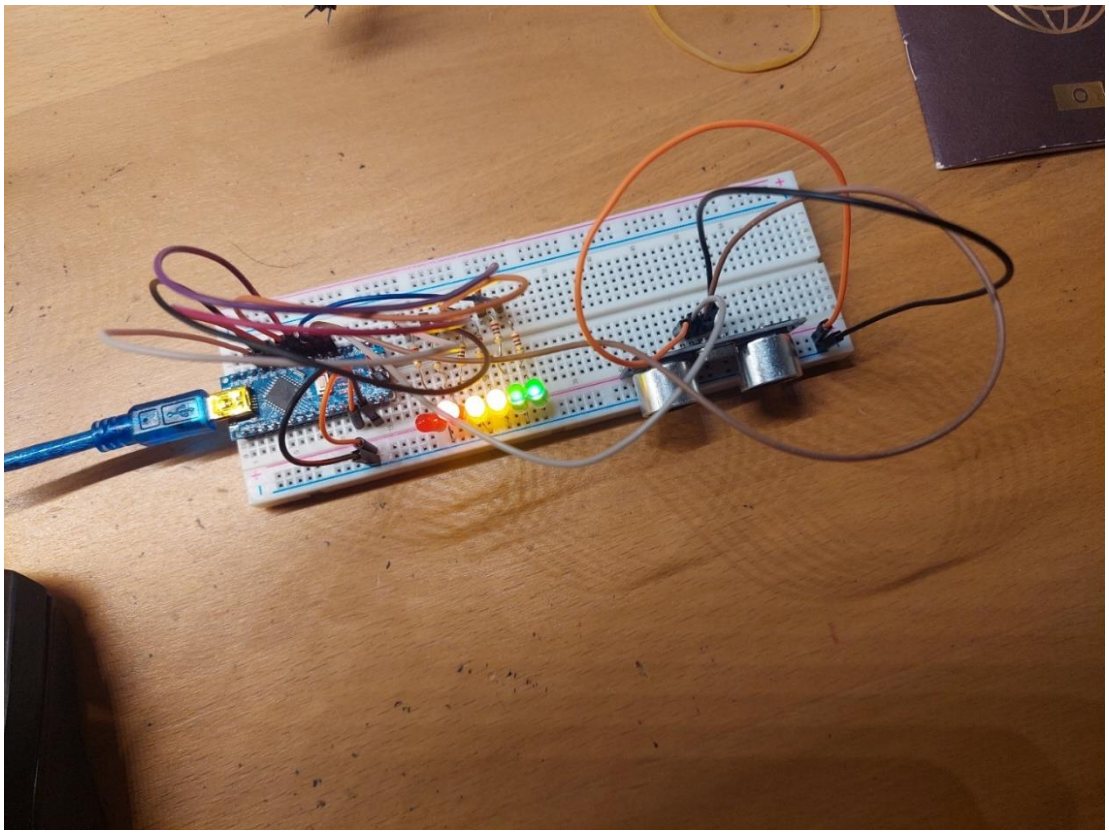
Şekil 14: 60-100 cm Arası 2 Yeşil Led



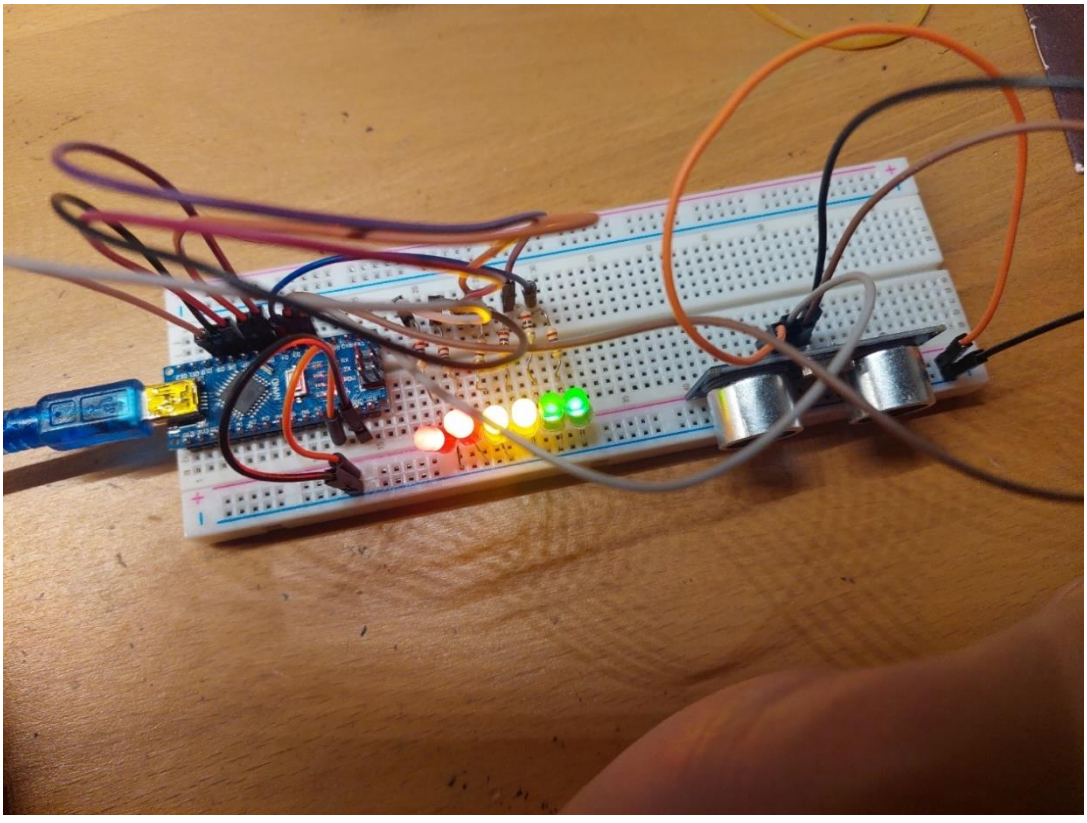
Şekil 14: 40-60 cm Arası 1 Sarı, 2 Yeşil Led



Şekil 15: 20-40 cm Arası 2 Sarı, 2 Yeşil Led



Şekil 15: 10-20 cm Arası 2 Kırmızı, 2 Sarı, 2 Yeşil Led



Şekil 16: 10 cm ve Altı 2 Kırmızı, 2 Sarı, 2 Yeşil Led

```
21:43:46.696 -> Yakin Mesafe: 16 cm, Nesne Sayisi: 1
21:43:46.933 -> Yakin Mesafe: 48 cm, Nesne Sayisi: 2
21:43:48.948 -> Yakin Mesafe: 58 cm, Nesne Sayisi: 3
21:43:49.724 -> Yakin Mesafe: 74 cm, Nesne Sayisi: 4
21:43:53.558 -> Yakin Mesafe: 90 cm, Nesne Sayisi: 5
21:43:54.409 -> Yakin Mesafe: 99 cm, Nesne Sayisi: 6
21:43:56.240 -> Yakin Mesafe: 51 cm, Nesne Sayisi: 7
21:43:57.763 -> Yakin Mesafe: 40 cm, Nesne Sayisi: 8
21:43:58.032 -> Yakin Mesafe: 59 cm, Nesne Sayisi: 9
21:43:59.758 -> Yakin Mesafe: 104 cm, Nesne Sayisi: 10
21:44:00.721 -> Yakin Mesafe: 102 cm, Nesne Sayisi: 11
21:44:02.381 -> Yakin Mesafe: 98 cm, Nesne Sayisi: 12
21:44:02.856 -> Yakin Mesafe: 27 cm, Nesne Sayisi: 13
21:44:05.209 -> Yakin Mesafe: 92 cm, Nesne Sayisi: 14
21:44:10.277 -> Yakin Mesafe: 40 cm, Nesne Sayisi: 15
21:44:11.096 -> Yakin Mesafe: 48 cm, Nesne Sayisi: 16
21:44:12.182 -> Yakin Mesafe: 101 cm, Nesne Sayisi: 17
21:44:13.230 -> Yakin Mesafe: 29 cm, Nesne Sayisi: 18
21:44:13.505 -> Yakin Mesafe: 42 cm, Nesne Sayisi: 19
21:44:14.183 -> Yakin Mesafe: 6 cm, Nesne Sayisi: 20
21:44:14.455 -> Yakin Mesafe: 48 cm, Nesne Sayisi: 21
21:44:15.641 -> Yakin Mesafe: 8 cm, Nesne Sayisi: 22
21:44:16.792 -> Yakin Mesafe: 14 cm, Nesne Sayisi: 23
21:44:17.060 -> Yakin Mesafe: 51 cm, Nesne Sayisi: 24
```

Şekil 16: Seri Port Ekranı Çıktısı

# 5. Ekler

```
#define F_CPU 16000000UL
#define SOUND_SPEED 0.0343
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define TRIG_PIN PD2
#define ECHO_PIN PD3
#define LED_GREEN1 PD4
#define LED_GREEN2 PD5
#define LED_YELLOW1 PD6
#define LED_YELLOW2 PD7
#define LED_RED1 PB0
#define LED_RED2 PB1

#define GREEN_LED_DDR DDRD
#define YELLOW_LED_DDR DDRD
#define RED_LED_DDR DDRB
#define TRIG_DDR DDRD
#define ECHO_DDR DDRD

#define TRIG_PORT PORTD

volatile uint16_t currentDistance = 0;
volatile uint16_t closestDistance = 0xFFFF;
volatile uint16_t objectCount = 0;
volatile uint8_t objectDetected = 0;
```

```
unsigned int measureDistance() {
PORTD |= (1 << TRIG_PIN);
_delay_us(10);
PORTD &= ~(1 << TRIG_PIN);

unsigned int count = 0;
while (!(PIND & (1 << ECHO_PIN)));

TCNT1 = 0;
TCCR1B |= (1 << CS11);

while (PIND & (1 << ECHO_PIN)) {
if (TCNT1 > 30000) {
break;
}
}

TCCR1B = 0;
count = TCNT1 / 2;

return (double)count * SOUND_SPEED / 2;
}



void USART_Init() {
UCSR0A = 0x00;
UBRR0H = 0x00;
UBRR0L = 0x67;
UCSR0C &= ~((1<<UMSEL00)|(1<<UMSEL01));
UCSR0B = (1<<RXEN0)|(1<<TXEN0);
UCSR0C = (1<<USBS0)|(1<<UCSZ00)|(1<<UCSZ01);
}

void USART_Transmit_Char(char c) {
while (!(UCSR0A & (1<<UDRE0)));
UDR0 = c;
}

void USART_Send_Distance_and_Count(uint16_t distance, uint16_t count) {

const char distancePrefix[] = "Yakin Mesafe: ";
for (uint8_t i = 0; distancePrefix[i] != '\0'; i++) {
USART_Transmit_Char(distancePrefix[i]);
}

if (distance == 0) {
USART_Transmit_Char('0');
} else {
char buffer[6];
uint8_t i = 0;
while (distance > 0) {
buffer[i++] = '0' + distance % 10;
distance /= 10;
}
while (i > 0) {
USART_Transmit_Char(buffer[--i]);
}
}

const char countPrefix[] = " cm, Nesne Sayisi: ";
for (uint8_t i = 0; countPrefix[i] != '\0'; i++) {
```

```
USART_Transmit_Char(countPrefix[i]);
}

if (count == 0) {
USART_Transmit_Char('0');
} else {
char buffer[6];
uint8_t i = 0;
while (count > 0) {
buffer[i++] = '0' + count % 10;
count /= 10;
}
while (i > 0) {
USART_Transmit_Char(buffer[--i]);
}
}


USART_Transmit_Char('\r');
USART_Transmit_Char('\n');
}


void setup_io() {
DDRD |= (1 << TRIG_PIN) | (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 <<
LED_YELLOW1) | (1 << LED_YELLOW2);
DDRB |= (1 << LED_RED1) | (1 << LED_RED2);
DDRD &= ~(1 << ECHO_PIN);
PORTD &= ~((1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 <<
LED_YELLOW2));
PORTB &= ~((1 << LED_RED1) | (1 << LED_RED2));
}


void processMeasurement(uint16_t distance) {
if (distance <= 200 && objectDetected == 0) {
objectDetected = 1;
closestDistance = distance;
} else if (distance > 200 && objectDetected == 1) {

objectDetected = 0;
objectCount++;
USART_Send_Distance_and_Count(closestDistance, objectCount);
closestDistance = 0xFFFF;
} else if (distance <= closestDistance) {

closestDistance = distance;
}
}


ISR(TIMER1_COMPA_vect) {
uint16_t distance = measureDistance();

PORTD &= ~((1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 <<
LED_YELLOW2));
PORTB &= ~((1 << LED_RED1) | (1 << LED_RED2));


if (distance > 200) {

} else if (distance > 100) {
```

```
PORTD |= (1 << LED_GREEN1);
} else if (distance > 60) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
} else if (distance > 40) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1);
} else if (distance > 20) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 <<
LED_YELLOW2);
} else if (distance > 10) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 <<
LED_YELLOW2);
PORTB |= (1 << LED_RED1);
} else {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 <<
LED_YELLOW2);
PORTB |= (1 << LED_RED1) | (1 << LED_RED2);
}
}


void setLEDsBasedOnDistance(uint16_t distance) {

PORTD &= ~((1 << LED_GREEN1) | (1 << LED_GREEN2) | (1 << LED_YELLOW1) | (1 <<
LED_YELLOW2));
PORTB &= ~((1 << LED_RED1) | (1 << LED_RED2));

if (distance <= 10) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
PORTD |= (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
PORTB |= (1 << LED_RED1) | (1 << LED_RED2);
} else if (distance <= 20) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
PORTD |= (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
PORTB |= (1 << LED_RED1);
} else if (distance <= 40) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
PORTD |= (1 << LED_YELLOW1) | (1 << LED_YELLOW2);
} else if (distance <= 60) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
PORTD |= (1 << LED_YELLOW1);
} else if (distance <= 100) {
PORTD |= (1 << LED_GREEN1) | (1 << LED_GREEN2);
} else if (distance <= 200) {
PORTD |= (1 << LED_GREEN1);
}
}

int main(void) {
setup_io();
USART_Init();
sei();

while (1) {
uint16_t distance = measureDistance();
processMeasurement(distance);
setLEDsBasedOnDistance(distance);

_delay_ms(100);
}

return 0;
}
```