Eine Arduino Alarmanlage mit Geräuschsensor

In diesem Projekt baust du dir deine eigene Arduino Alarmanlage. Diese besteht aus drei Komponenten: **einem Geräuschsensor, einem aktiven Piezo-Summer und einer RGB-LED.**

Mit dem Geräuschsensor misst du die Umgebungslautstärke. Sobald von dir bestimmte Schwellenwerte unter- bzw. überschritten werden, leuchtet die RGB-LED Grün oder Gelb. Sobald die Lautstärke ein festgelegtes Maß überschreitet, leuchtet die LED in Rot und aus dem Piezo ertönt ein schriller Alarmton.

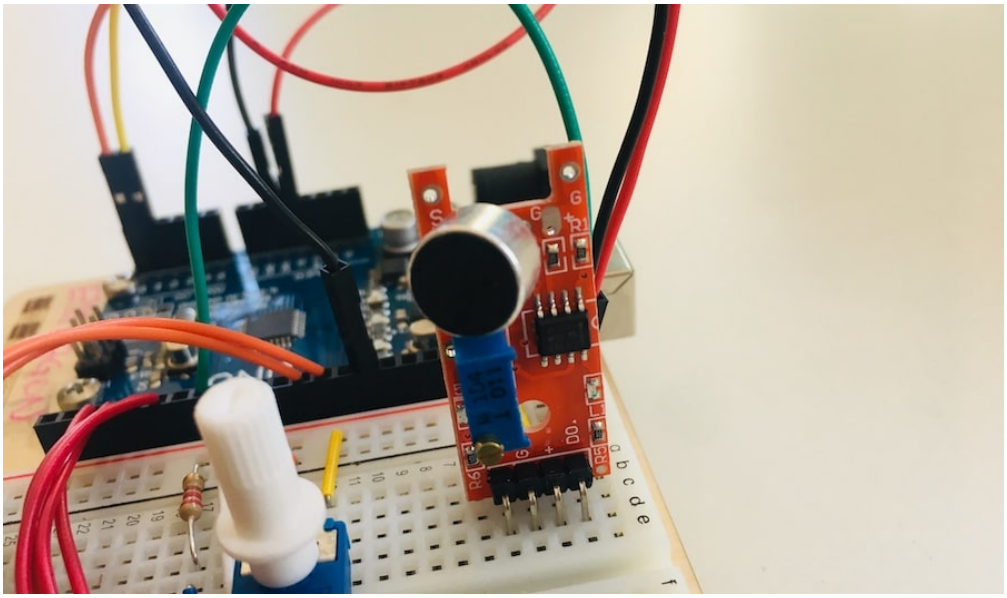
In den folgenden Abschnitten lernst du, wie du den Geräuschsensor, den Piezo-Summer und die RGB-LED anschließt. Danach erfährst du, wie alle drei Bauteile zusammen die Alarmanlage bilden.

DER GERÄUSCHSENSOR

Zunächst erfährst du, wie du den Geräuschsensor (KY-037, auch Sound Sensor genannt) anschließt und damit Geräusche sowohl analog als auch digital in deinem Arduino UNO verarbeitest.

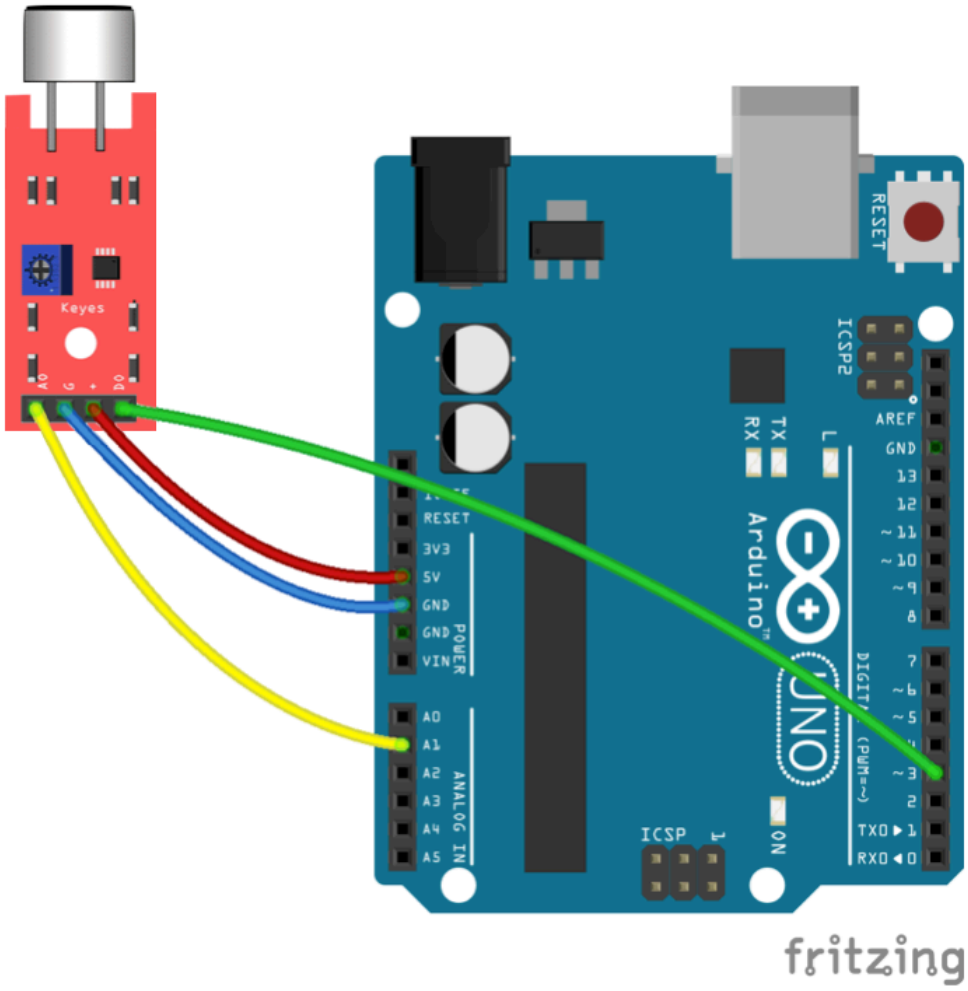
Die wichtigsten Bauteile des Geräuschsensors, den du hier kennenlernst, sind: das Mikrofon, ein Komparator (hier der

LM393, der zwei Spannungen miteinander vergleicht) und ein Potentiometer (um den Schwellenwert einzustellen). Du schließt den Sensor über mindestens 3 der 4 Pins an – Anode, Kathode, Analog- und/oder Digitalausgang.



ANSCHLUSS AM DIGITALEN AUSGANG

Du kannst den Geräuschsensor auf zwei Arten an deinem Arduino anschließen – analog oder digital. **Der digitale Anschluss ist sinnvoll, wenn du etwas in Gang setzen möchtest, sobald ein lautes Geräusch erkannt wird.** Das könnte zum Beispiel ein Klopfen an der Tür oder ein Knall sein. So schließt du deinen Sensor an:



Versorge deinen Sensor mit Strom über die **Pins + und G** (für Ground, also Minus) und schließe den digitalen Ausgang (DO) am Arduino am digitalen Eingang 3 an. Den analogen Ausgang (AO) verbindest du als nächstes am Arduino mit dem Pin A1. Und das war es schon.

Kopiere dir nun den folgenden Sketch und lade ihn auf deinen Arduino UNO.

```
const int sensor = 3;
const int led = LED_BUILTIN;
int noise  = 0;

void setup() {

pinMode(sensor, INPUT);
pinMode(led, OUTPUT);
```

```
Serial.begin(9600);

}

void loop() {

noise = digitalRead(sensor);
Serial.println(noise);

if (noise == 1){
  digitalWrite(led, HIGH);
}else{
  digitalWrite(led, LOW);
  }
}
```

Sollte die interne LED des Arduinos jetzt dauerhaft leuchten, bedeutet das, dass der Sensor durchgehend ein Geräusch erkennt. **Nimm jetzt einen kleinen Schraubendreher zur Hand und drehe die Schraube am Potentiometer nach links – solange bis die LED ausgeht.**

Jetzt erzeuge direkt neben dem Mikrofon ein lautes Geräusch – schnippe zum Beispiel mit den Fingern. Die LED sollte kurz aufleuchten. Falls nicht – drehe die Schraube wieder leicht nach rechts. Mit etwas Fingerspitzengefühl findest du die richtige Feinjustierung. Du kannst auch im Seriellen Monitor nachverfolgen, ob der Sensor ein Geräusch erkennt: Bei “Stille” siehst du dort eine Null, bei einem Geräusch eine 1.

UND JETZT DER ANALOGE AUSGANG

Wenn du den Geräuschsensor analog anschließt, erhältst du in Echtzeit ein Feedback zur Lautstärke. So kannst du zum Beispiel eine LED aufleuchten lassen, sobald **eine von dir bestimmte Lautstärke** überschritten wird.

Du hast den analogen Ausgang (AO) des Sensors ja bereits mit deinem Arduino verbunden. Lade nun folgenden Sketch hoch.

```
const int sensor = A1;
const int led = LED_BUILTIN;
int noise  = 0;

void setup() {

pinMode(sensor, INPUT);
pinMode(led, OUTPUT);
Serial.begin(9600);

}

void loop() {

noise = analogRead(sensor);
Serial.println(noise);

if (noise > 200){
  digitalWrite(led, HIGH);
}else{
  digitalWrite(led, LOW);
  }
}
```

Um die Lautstärke besser verfolgen zu können, starte den Seriellen Monitor und beobachte dort die Sensorwerte. **Auch jetzt ist wieder etwas Feinjustierung nötig.** Stelle das Potentiometer so ein, dass du im seriellen Monitor Werte siehst, du etwas unter 200 liegen. Wenn du jetzt ein Geräusch machst, das laut genug ist, dass der Wert über 200 schnellt, leuchtet die interne LED des Arduinos auf.

Du siehst also, es reicht hier nicht, *dass* ein Geräusch erkannt wird. Dieses Geräusch *muss auch noch* laut genug sein, um die LED einzuschalten.

Die Lautstärke wirst du mit einer RGB-LED darstellen und sobald eine bestimmte Lautstärke überschritten wird, ertönt ein Alarmsignal aus dem Piezo-Summer.

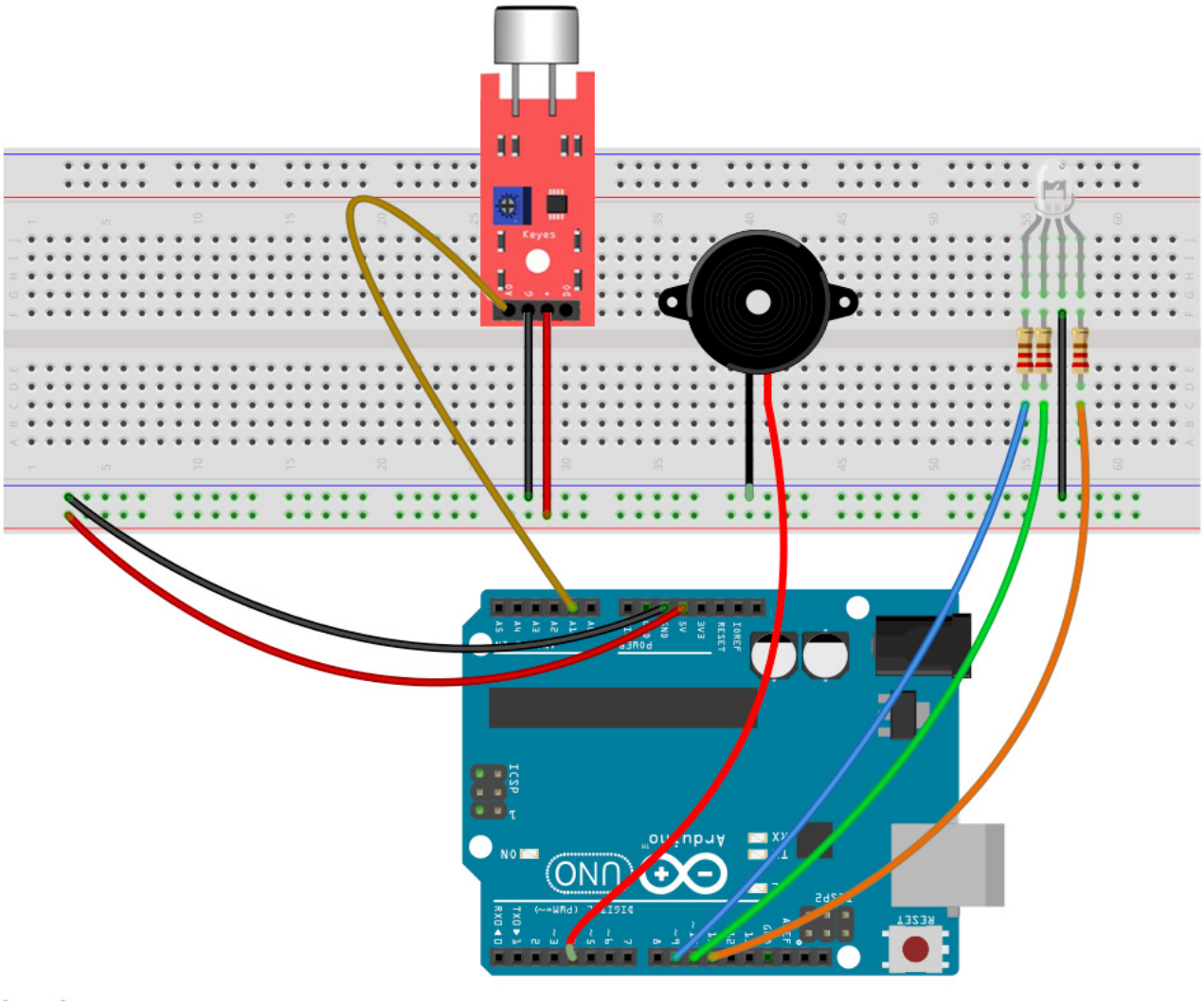
DEN AKTIVEN PIEZO UND DIE RGB-LED ANSCHLIESSEN

Den Geräuschsensor hast du ja bereits installiert. Allerdings benötigst du für die Arduino Alarmanlage nur den Analogausgang (AO) des Sensors. Das Kabel am Digitalausgang kannst du weglassen bzw. entfernen.

Mit der Spieluhr und dem Theremin hast du ja bereits den passiven Piezo-Summer kennengelernt. Wie du weißt, kannst du mit diesem unterschiedliche Töne erzeugen. Nicht so mit dem aktiven Piezo: Dieses Bauteil kann lediglich einen einzigen Ton erzeugen – und macht das, sobald es mit Strom versorgt wird. Für eine Alarmanlage ist das jedoch völlig ausreichend.

Wenn du dir nicht sicher bist, ob dein Piezo aktiv oder passiv ist, lege einfach 5V Spannung direkt von deinem Arduino an. Erklingt ein Ton? Dann ist es der aktive Piezo, den du für dieses Projekt brauchst.

Der Anschluss ist ganz einfach. Verbinde das kurze Bein des Piezo-Summers mit Minus und das lange mit dem Digitalpin 4 an deinem Arduino. Wenn später der Geräuschsensor einen Alarm auslöst, leitest du über diesen Pin Strom an den Piezo, der daraufhin zu piepsen anfängt.



Etwas aufwändiger ist der Anschluss der RGB-LED. Insgesamt besitzt sie 4 Beinchen: eine Kathode, die du mit Minus verbindest, und drei Anoden – je eine für die Farben Rot, Grün und Blau. Wie du es von “regulären” LEDs gewohnt bist, musst du auch hier einen 220Ω Vorwiderstand einbauen – hier für jede Anode einen.

Die drei Anoden verbindest du mit den Digitalpins 9, 10 und 11. Diese drei Pins verfügen über die Pulsweitenmodulation (PWM), die du bereits kennengelernt hast, also du die Helligkeit einer einfachen LED gesteuert hast.

Möglicherweise passen die oben gezeigten Farbkanäle und Arduino-Pins **je nach Bauart der LED** nicht zusammen. Im Beispiel unten ist der rote Farbkanal rechts von der Kathode,

was bei deiner LED jedoch anders sein kann. Das kannst du jedoch leicht herausfinden und beheben, wenn du später die Farbe des Lichts umschaltest: Passe einfach die Variablen im Sketch an schreibe hinter den Farbkanal den richtigen Pin:

```
int ledRed = 10;
int ledGreen = 9;
int ledBlue = 8;
```

Wenn du alles so wie auf der Skizze oben aufgebaut und verkabelt hast, kann es mit dem Code weitergehen.

DER SKETCH FÜR DIE Arduino ALARMANLAGE

Jetzt, wo du alles auf deinem Breadboard aufgebaut hast, wird es Zeit für das Programm. Im Folgenden erfährst du mehr über die einzelnen Teile des Programms und deren Funktion.

Gleich zu Beginn des Sketchs legst du fest, welche Hardware du an welchen Pins angeschlossen hast. Diese kannst du mit **const** natürlich auch als Konstanten festlegen. Außerdem benötigst du ein paar Variablen für die Helligkeit der einzelnen Farbkanäle der RGB-LED (z.B. **brightnessRed = 150)** und die Lautstärke, die der Geräuschsensor misst. Diese legst du zu Beginn auf Null fest.

```
int ledRed = 11;
int ledGreen = 10;
int ledBlue = 9;

int brightnessRed = 150;
int brightnessGreen = 150;
int brightnessBlue = 150;

int noise = 0;
int sensor = A1;
```

```
int piezo = 4;
```

DIE SETUP-FUNKTION

Hier gibt es vermutlich auch nichts, das du nicht bereits schon kennst. Du legst die Pins der LED und des Piezos als **OUTPUT** fest und startest den Seriellen Monitor.

```
void setup() {

  pinMode(ledRed, OUTPUT);
  pinMode(ledGreen, OUTPUT);
  pinMode(ledRed, OUTPUT);

  pinMode(piezo, OUTPUT);

  Serial.begin(9600);
}
```

DER LOOP

Hier wird es nun spannend. Als erstes misst du die Umgebungslautstärke, denn hierauf basieren später alle weiteren Aktionen im Sketch – also ob deine Arduino Alarmanlage anspringt oder nicht.

```
noise = analogRead(sensor);
Serial.println(noise);
```

Anschließend folgt die erste bedingte Anweisung. Wenn nämlich der Geräuschpegel unter einem bestimmten Wert liegt, soll die LED grün leuchten – was so viel bedeutet wie “Die Luft ist rein”. Den Wert von 200 (und die folgenden in den weiteren Anweisungen) kannst du natürlich anpassen. Achte auch darauf, dass du beim ersten Start der Alarmanlage den Geräuschsensor so kalibrierst, dass er bei Ruhe unter dem von dir

festgelegten Wert liegt.

```
if(noise <= 200){
  analogWrite(ledGreen, brightnessGreen);
  analogWrite(ledRed, 0);
  analogWrite(ledBlue, 0);
  digitalWrite(piezo, LOW);
  }
```

Wie erwähnt, soll in diesem Zustand die LED grün leuchten. Das erreichst du, indem du nur den grünen Farbkanal mit der von dir festgelegten Helligkeit **brightnessGreen** aufleuchten lässt. Die beiden anderen Kanäle für Rot (Red) und Blau (Blue) erhalten die Helligkeit **Null**, sind also aus. Du kannst natürlich auch Farben mischen (auch wenn die LED diese nicht sehr differenziert darstellen kann) – hierfür eignet sich z.B. dieses RGB Color Wheel.

Auch der Piezo soll hier nicht ertönen, weswegen du keinen Strom an ihn leitest. Das erreichst du mit dem Parameter **LOW** in der Funktion **digitalWrite()**.

EINE WEITERE ANWEISUNG

Wenn nun der Geräuschpegel etwas ansteigt, aber immer noch nicht hoch genug für einen Alarm ist, kommt eine zweite bedingte Anweisung ins Spiel – mit **else if{}**.

```
else if(noise > 200 && noise <= 350){
  analogWrite(ledRed, brightnessRed);
  analogWrite(ledGreen, brightnessGreen);
  analogWrite(ledBlue, 0);
  digitalWrite(piezo, LOW);
  }
```

Diese Befehle werden ausgeführt, wenn die Lautstärke zwischen 201 und 350 liegt. Wie gesagt, experimentiere mit diesen Werten, um sie an deine Gegebenheiten anzupassen.

Befindet sich also die Lautstärke in diesem Bereich, wechselt das Licht der LED von Grün zu Gelb. Für diese Farbe gibt es keinen eigenen Farbkanal, weswegen du sie kurzerhand mischst. Gelb ist eine Mischung aus Rot und Grün. Deshalb schaltest du diese Farbkanäle der LED und lässt den blauen Kanal erlöschen.

ALARM!

Wenn es nun noch lauter wird, soll der Alarm ertönen. Die LED strahlt in einem satten Rot und der Piezo-Summer fängt an zu piepsen.

```
else if(noise > 350){
  analogWrite(ledRed, brightnessRed);
  analogWrite(ledGreen, 0);
  analogWrite(ledBlue, 0);
  digitalWrite(piezo, HIGH);
  delay(10000);
  }
```

Diesmal schaltest du also nur den roten Kanal der LED ein. Außerdem leitest du nun Strom vom Arduino an den Piezo – mit der Funktion **digitalWrite(piezo, HIGH);**

Sicherlich sollte eine Alarmanlage so lange Krach machen, bis sie ausgeschaltet wird. Für ein erstes Experiment reicht jedoch vielleicht auch erst einmal eine Sekunde. Deshalb befindet sich am Ende noch ein **delay()** von 1.000 Millisekunden.

Und das war es! Lade den folgenden Sketch auf deinen Arduino, kalibriere deinen Geräuschsensor und probiere deine Arduino Alarmanlage aus.

```
int ledRed = 11;
int ledGreen = 10;
int ledBlue = 9;

int brightnessRed = 150;
```

```
int brightnessGreen = 150;
int brightnessBlue = 150;

int noise = 0;
int sensor = A1;

int piezo = 4;

void setup() {

  pinMode(ledRed, OUTPUT);
  pinMode(ledGreen, OUTPUT);
  pinMode(ledRed, OUTPUT);

  pinMode(piezo, OUTPUT);

  Serial.begin(9600);

}

void loop() {

  noise = analogRead(sensor);
  Serial.println(noise);

  if(noise <= 200){
    analogWrite(ledGreen, brightnessGreen);
    analogWrite(ledRed, 0);
    analogWrite(ledBlue, 0);
    digitalWrite(piezo, LOW);
    }
    else if(noise > 200 && noise <= 350){
      analogWrite(ledRed, brightnessRed);
      analogWrite(ledGreen, brightnessGreen);
      analogWrite(ledBlue, 0);
      digitalWrite(piezo, LOW);
      }
      else if(noise > 350){
        analogWrite(ledRed, brightnessRed);
        analogWrite(ledGreen, 0);
        analogWrite(ledBlue, 0);
```

```
        digitalWrite(piezo, HIGH);
        delay(10000);
        }
}
```

Apropos Piezo: Im verlinkten Tutorial erfährst du, wie du einen passiven Piezo-Summer verwenden kannst.