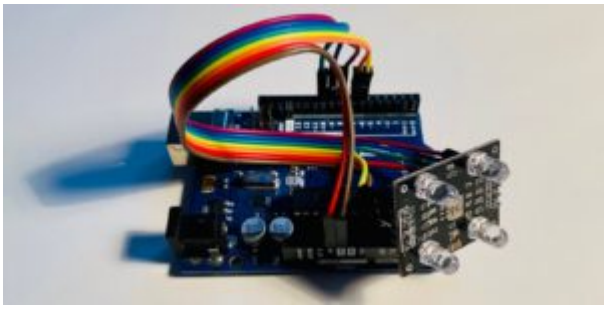
Farben erkennen mit einem Farbsensor und dem Arduino



Mit einem Farbsensor bringst du deinem Arduino bei, verschiedene Farben voneinander unterscheiden zu können. Zwar können die meisten Sensoren keine feinen Unterschiede erkennen – um aber zum Beispiel die Farben von M&Ms oder Skittles erkennen zu können, reichen sie allemal!

Hier lernst du, wie du einen Farbsensor* anschließt, damit Farben erkennst und diese in das bekannte RGB-Modell umrechnest. In diesem Tutorial verwenden wir den gängigen Sensor TC3200.

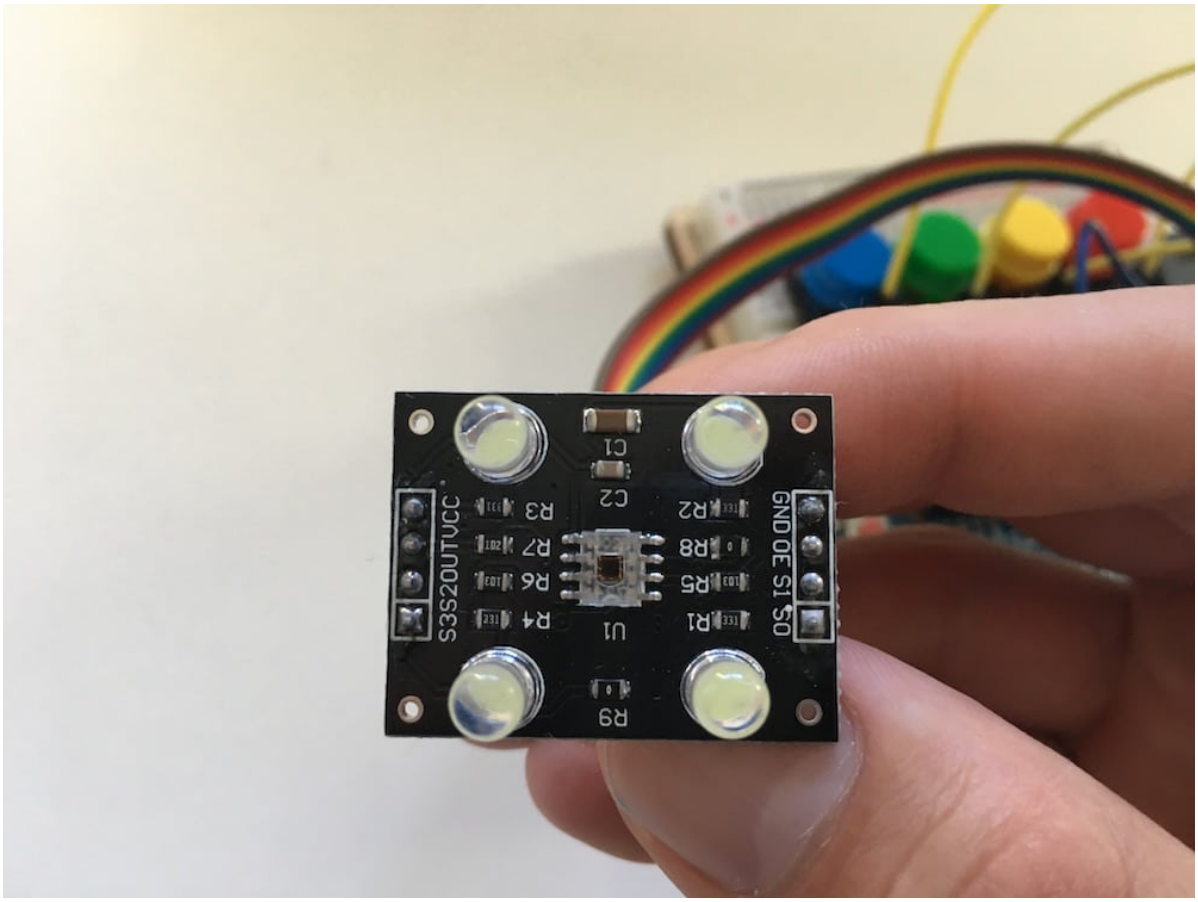
AZDelivery Farbsensor TCS230 TCS3200 Farben Sensor Modul kompatibel mit Arduino inklusive E-Book!
 Chip: TAOS TCS3200 RGB Sensor; Eingangsspannung: DC 3 ~ 5V; Hellweiße LEDs...
8,99 €

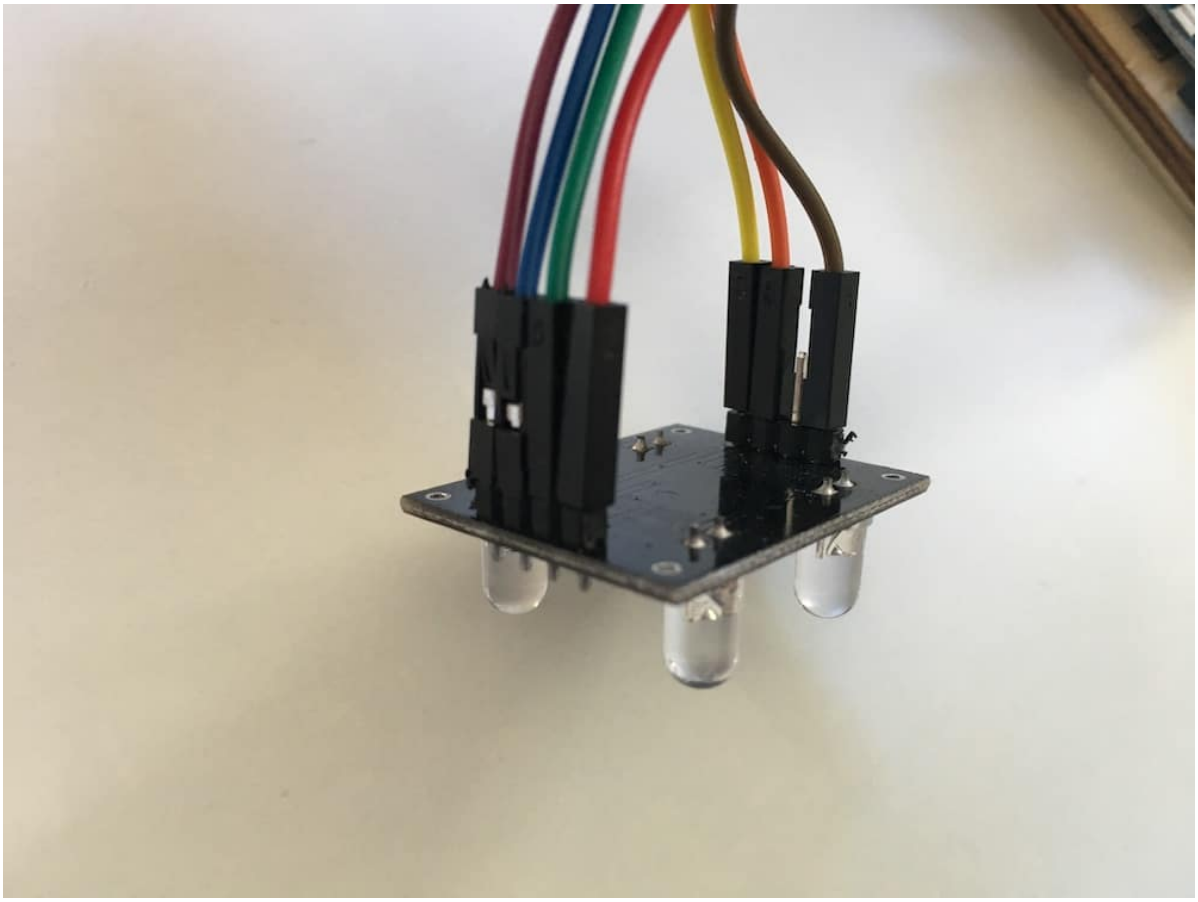
Den Farbsensor am Arduino anschließen

Am Sensor findest du insgesamt 8 Pins, von denen du 7 mit deinem Arduino wie folgt verbinden kannst. Die Wahl der Digitalpins ist natürlich dir selbst überlassen.

Farbsensor	Arduino (Digitalpin)
VCC	5V
GND	GND
S0	6

S1	7
S2	8
S3	9
OUT	10

- 

- 

Eine kurze Erklärung zu den einzelnen Pins am Farbsensor: Mit den Pins S0 und S1 kannst du die Ausgangsfrequenz des Sensors steuern. So kannst du deine Ergebnisse optimieren, was wir in diesem Tutorials allerdings erst einmal beiseite lassen.

Die Pins S2 und S3 steuern die einzelnen Photodioden des Sensors. Hiervon gibt es je 16 Stück für die Farben Rot, Grün, Blau und Clear (also ohne Farbfilter). Die beiden Pins kannst du von deinem Arduino entweder auf HIGH oder LOW setzen – die Kombination bestimmt dann, welche Dioden angesprochen und welche Farbe also „gelesen“ werden soll:

S2	S3	Farbe
LOW	LOW	Rot
LOW	HIGH	Blau
HIGH	LOW	Clear (kein Filter)
HIGH	HIGH	Grün

Gleich im Sketch schauen wir uns diese Kombinationen genauer an. Es fehlt allerdings noch der Pin OUT – hierüber liest dein Arduino die Messergebnisse ein.

Der Sketch zum Farben erkennen

Kommen wir also zum Code. Du benötigst für den Farbsensor keine eigene Bibliothek, sondern kannst die Messungen mit ein paar einfachen Funktionen selbst erledigen. Definiere zunächst die verwendeten Anschlüsse zu Beginn des Sketchs:

```
#define S0 6
#define S1 7
#define S2 8
#define S3 9
#define sensorOut 10
```

Anschließend benötigst du noch je drei Variablen für die Roh- und RGB-Werte der einzelnen Farben Rot, Grün und Blau:

```
int frequencyR = 0;
int frequencyG = 0;
int frequencyB = 0;

int red = 0;
int green = 0;
int blue = 0;
```

Die Setup-Funktion

Hier startest du deinen Seriellen Monitor und definierst die einzelnen **pinModes**:

```
Serial.begin(115200);

pinMode(S0, OUTPUT);
pinMode(S1, OUTPUT);
pinMode(S2, OUTPUT);
pinMode(S3, OUTPUT);
```

```
pinMode(sensorOut, INPUT);
```

Wie du siehst, sprichst du die Pins S1 bis S3 von deinem Arduino aus an (OUTPUT) und möchtest vom Pin **sensorOut** Messdaten erhalten (INPUT). Fehlt noch eine Sache: Wie oben kurz erwähnt, steuerst du mit den Pins S1 und S2 die Ausgangsfrequenz. Diese setzt du mit den folgenden zwei Zeilen auf 20%:

```
digitalWrite(S0, HIGH);
digitalWrite(S1, LOW);
```

Für die Ausgangsfrequenz gibt es festgelegte Werte und Kombinationen. Wenn du etwas tiefer einsteigen und die Ergebnisse deines Sensors optimieren möchtest, orientiere dich an dieser Tabelle:

S0	S1	Ausgangsfrequenz
LOW	LOW	Aus
LOW	HIGH	2%
HIGH	LOW	20%
HIGH	HIGH	100%

Der Loop

Kommen wir also zu den Messungen. Im Loop deines Sketchs führst du für jede der drei Farben Rot, Grün und Blau eine separate Messung durch und rechnest deine Ergebnisse in den RGB-Farbraum um. Zunächst der Code für Rot:

```
digitalWrite(S2, LOW);
digitalWrite(S3, LOW);

frequencyR = pulseIn(sensorOut, LOW);
Serial.print("Red = ");
Serial.print(frequencyR);
```

Als erstes setzt du die beiden Pins S2 und S3 auf LOW, um nur die Photodioden mit rotem Farbfilter anzusprechen. Anschließend liest du den Wert mit der Funktion **pulseIn()** ein und speicherst den Wert in der Variablen **frequencyR**. Zuletzt gibst du ihn im Seriellen Monitor aus. In der Arduino-Referenz lernst du mehr über die Funktion pulseIn().

Mit diesen Messergebnissen kannst du noch nicht viel anfangen – besser ist es, wenn du einen bekannten Wert wie den Rotanteil im RGB-Modus erhältst. So kannst du nämlich zusammen mit den Ergebnissen der Farben Grün und Blau die Farbe vor deinem Sensor mehr oder weniger genau bestimmen und sie z.B. zur Kontrolle auf einem TFT-Display* anzeigen.

Um das zu erreichen, bietet sich die Funktion **map()** an:

```
map(frequencyR, 20, 120, 255, 0)
```

In dieser Funktion benötigst du fünf Argumente: Zunächst den Messwert **frequencyR** und den Messbereich des Sensors. Hierfür machst du am besten ein paar Testmessungen mit verschiedenfarbigen Objekten vor deinem Farbsensor und schaust dir die Minimal- und Maximalwerte im Seriellen Monitor an. **Bei uns lagen die Werte ungefähr im Bereich von 20 bis 120 – das kann bei dir jedoch anders sein.**

Wenn du deinen passenden Wertebereich ermittelt hast, kommen die letzten beiden Argumente ins Spiel. Diese geben den maximalen (255) und minimalen Rotwert (0) im RGB-Modus an. Je „mehr“ Rot dein Farbsensor erkennt, desto geringer sein Messwert (bei uns also in Richtung 20). Das entspricht jedoch einem hohen Wert im RGB-Farbraum – deshalb „mappst“ du die 20 auf die 255 und andersherum die 120 auf die 0.

Alle Farben zusammen

Und das war es im Prinzip. Für die Farben Blau und Grün wiederholst du einfach das Prozedere der Farbe Rot. In deinem

Seriellen Monitor siehst du nun die RGB-Werte jeder einzelnen Farbe, die zusammen **in etwa** die Farbe des Objekts vor dem Sensor ergeben sollten.

Hier nun der gesamte Sketch zum Rauskopieren:

```
//pins for color sensor
#define S0 6
#define S1 7
#define S2 8
#define S3 9
#define sensorOut 10

//raw and RGB values for each color
int frequencyR = 0;
int frequencyG = 0;
int frequencyB = 0;

int red = 0;
int green = 0;
int blue = 0;

void setup() {
  Serial.begin(115200);

  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);
  digitalWrite(S0, HIGH);
  digitalWrite(S1, LOW);
}

void loop() {
  digitalWrite(S2, LOW);
  digitalWrite(S3, LOW);

  frequencyR = pulseIn(sensorOut, LOW);
  Serial.print("Red = ");
  //Serial.print(frequencyR);
```

```
  red = map(frequencyR, 20, 150, 255, 0);
  Serial.print(red);


  digitalWrite(S2, LOW);
  digitalWrite(S3, HIGH);

  frequencyB = pulseIn(sensorOut, LOW);
  Serial.print(" Blue = ");
  //Serial.print(frequencyB);
  blue = map(frequencyB, 20, 150, 255, 0);
  Serial.print(blue);


  digitalWrite(S2, HIGH);
  digitalWrite(S3, HIGH);

  frequencyG = pulseIn(sensorOut, LOW);
  Serial.print(" Green = ");
  //Serial.println(frequencyG);
  green = map(frequencyG, 20, 150, 255, 0);
  Serial.println(green);

  delay(100);
}
```

Wie geht es weiter?

Du kannst nun Farben mit einem Farbsensor erkennen und sie in den RGB-Farbraum übertragen. Wie wäre es, wenn du dir einen eigenen **Color Picker** baust, mit dem du die RGB-Werte einzelner Objekte ermittelst? Hierfür bietet sich ein TFT-Display an, auf dem du die ermittelte Farbe auch gleich darstellen kannst.