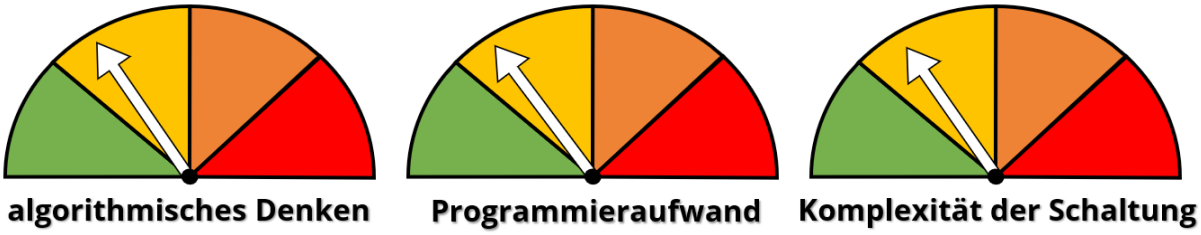
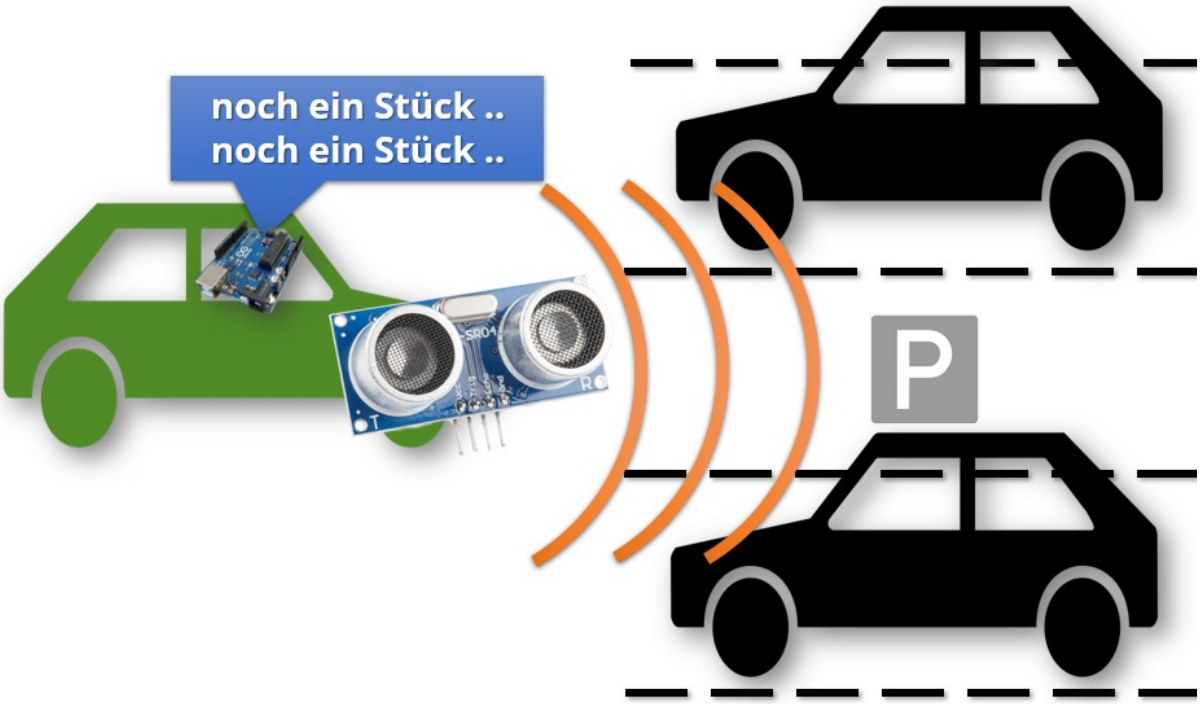
TECHNISCHE UNIVERSITÄT DRESDEN

Arduino Uno

Station 7a | Parksensor - Ultraschallsensor





Professur für Didaktik der Informatik

SYLBER Synergetische Lehrerbildung im exzellenten Rahmen

Das Maßnahmenpaket „TUD-Sylber² – Synergetische Lehrerbildung im exzellenten Rahmen“ wird im Rahmen der gemeinsamen „Qualitätsoffensive Lehrerbildung“ von Bund und Ländern aus Mitteln des Bundesministeriums für Bildung und Forschung gefördert.

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

ZIEL DER STATION

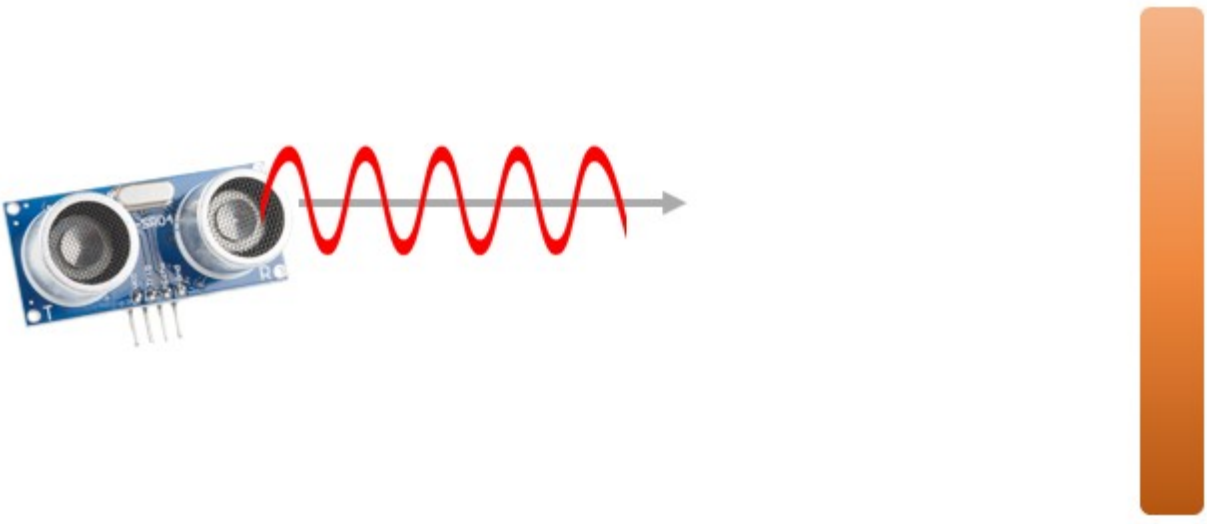
DEINE AUFGABE

Du hast dich für den Ultraschallsensor entschieden! Mit diesem werden wir jetzt Schritt für Schritt einen Rückfahrsensor entwickeln. Dazu schauen wir uns erst einmal an wie dieser funktioniert und angeschlossen wird. Im Anschluss werden wir den dazugehörigen Programmcode schreiben und den Abstand zu einem Gegenstand bestimmen.

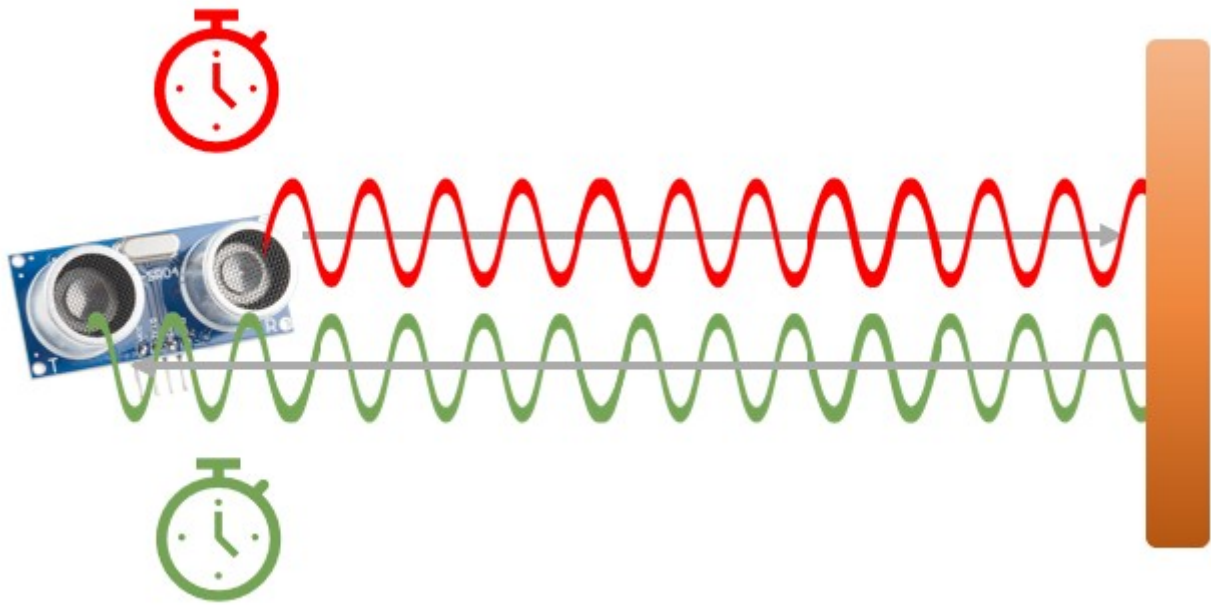
DIE FUNKTIONSWEISE DES SENSORS

Als Menschen hören wir Tonfrequenzen im Bereich von 16 Hz bis 20000 Hz. Als Ultraschall bezeichnen wir alles, was über diesem Intervall liegt, somit ist dies für uns nicht wahrnehmbar. Aber woher kennen wir das bereits? Fledermäuse zum Beispiel orientieren sich mit Hilfe von Ultraschall am nächtlichen Himmel. Doch wie funktioniert das eigentlich?

Schall breitet sich mit einer Geschwindigkeit von $342{,}2\frac{m}{s}$ beziehungsweise $1236\frac{km}{h}$ aus. Weiterhin hat eine Schallwelle die Eigenschaft an Oberflächen und Gegenständen reflektiert zu werden. Wir beginnen also zunächst damit eine Ultraschallwelle auszusenden.



Die Schallwelle erreicht das Hindernis natürlich nicht sofort. Das wollen wir uns zu nutze machen, wir stoppen die Zeit vom Aussenden der Schallwelle, bis zum wieder Auftreffen auf dem Sensor, nach der Reflektion an der Wand.



Die Schallwelle hat jetzt einmal den Weg vom Sensor zur Wand und wieder zurück hinter sich gebracht. Gestoppt haben wir die Zeiten und können damit bestimmen, wie lange die Welle für die Strecke gebraucht hat. Aus dem Physikunterricht kennst du bereits das Weg-Zeit-Gesetz:

$$v = \frac{s}{t} \qquad v \ldots Geschwindigkeit \quad s \ldots Strecke\,[Meter] \quad t \ldots Zeit\,[Sekunden]$$

Die Geschwindigkeit der Schallwelle kennen wir ja bereits mit $v = 342{,}2\frac{m}{s}$. Ebenso die Zeit ist mit unserer Messung bekannt, wie müssen lediglich darauf achten, was genau wir gemessen haben. Wir haben die Zeitdauer für das Doppelte des Weges zum Objekt bestimmt (hin und zurück), teilen wir diese Zeit durch 2, erhalten wir unsere gesuchte Zeitdauer. Nehmen wir beispielhaft eine Gesamtzeit von 1 Sekunde an, damit ergibt sich folgende Rechnung zur Bestimmung der Strecke.

$$s = v * t = 342{,}2\frac{m}{s} * (\frac{1}{2}) s = 171{,}1 m$$

AUFGABE 1 – BEISPIELRECHNUNG

Berechne die zurückgelegte Strecke, wenn die Zeitdifferenz 10 Millisekunden beträgt. Eine Millisekunde entspricht 0,001 Sekunden.

Dein Ergebnis für die zurückgelegte Strecke: _____

ANSCHLIESSEN DES ULTRASCHALLSENSORS

Angeschlossen werden müssen 4 Pins. Dabei sind `+5 V` und `GND` zur Stromversorgung des Sensors. Die beschrifteten Pins für „`Trig`“ und „`Echo`“ dienen dem Senden und Empfangen von Ultraschallwellen. Wenn der „`Trigger`“-Pin auf `HIGH` gesetzt wird, sendet der Sensor eine Schallwelle aus. Sobald die Welle wieder den Sensor erreicht. Sendet der Sensor ein `HIGH` am Echo Pin.

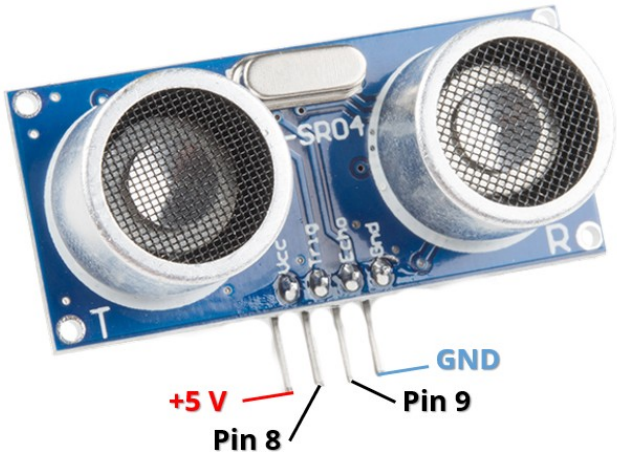
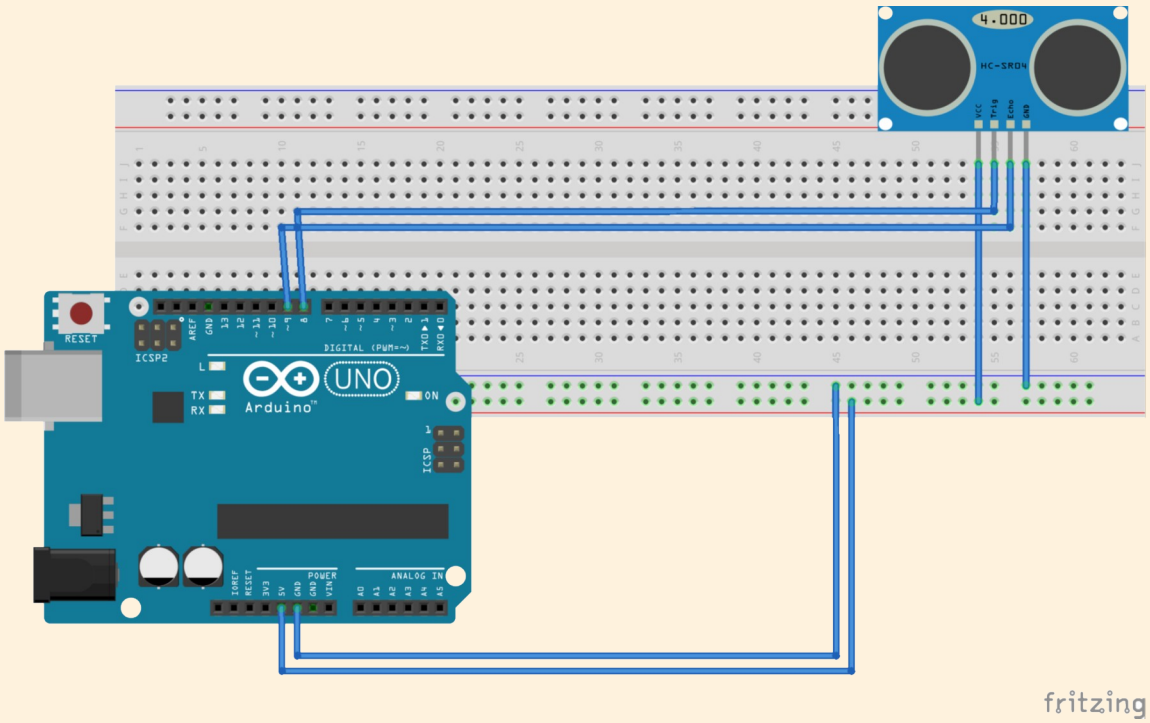


Abbildung 1: Ultraschallsensor

AUFGABE 2 – ANSCHLIESSEN DES SENSORS

Verbinde zunächst den Sensor mit deinem Arduino. Du benötigst keine weiteren Widerstände. Achte darauf die „`Trigger`“- und „`Echo`“-Pins so anzuschließen, wie oben in der Abbildung gezeigt.



PROGRAMMIERUNG DER ABSTANDSMESSUNG

Zunächst beginnen wir mit der Initialisierung von Variablen und legen die jeweiligen Pin-Modi fest. Was benötigen wir alles? Wir wollen die Zeitdauer bestimmen und daraus die Entfernung ableiten. Dabei reicht uns in diesem Fall der normale Datentyp `integer` nicht aus, denn die Zahlen werden größer als der Wertebereich.

Mit dem normalen Integer-Datentyp können wir 16-Bit Zahlen abbilden. Das entspricht 65536 Werten. Unsere Zahlen werden allerdings größer werden, daher nutzen wir den `long` Integer-Datentyp. Dieser unterstützt 32 Bit und damit 4 Milliarden mögliche Werte. Die Initialisierung sieht dann wie folgt aus:

```
long zeitdauer = 0;
long entfernung = 0;
```

Außerdem brauchen wir noch Variablen für die „`Trigger`“- und „`Echo`“-Pins.

```
int trigger = 9;
int echo = 8;
```

Schauen wir uns nun die Festlegung der Pin-Modi an. Mit Hilfe des „`Trigger`“-Pins lösen wir das Aussenden der Schallwelle aus, dieser muss also als `OUTPUT` festgelegt werden. Der „`Echo`“-Pin liefert `HIGH`, wenn die Schallwelle wieder auf dem Server auftrifft, dies wollen wir auslesen. Daher muss dieser Pin auf `INPUT` festgelegt werden.

```
pinMode(trigger, OUTPUT);
pinMode(echo, INPUT);
```

AUFGABE 3 – VARIABLEN UND PIN-MODI

Öffne einen neuen Sketch und initialisiere die oben benannten Variablen, lege den Modus der Pins fest und starte bereits innerhalb der `setup()`-Methode den seriellen Monitor mit `Serial.begin (9600)` um später die Entfernungswerte ausgeben zu lassen.

Die Grundlagen haben wir somit gelegt, beschäftigen wir uns mit der Funktionalität des Programms. Um die Schallwelle auszusenden, legen wir zunächst für 10 Millisekunden `HIGH` an den „`Trigger`“-Pin an.

```
digitalWrite(trigger, HIGH);
delay(10);
digitalWrite(trigger, LOW);
```

Um im Anschluss die Zeitdauer für die Ausbreitung der Welle zu bestimmen, nutzen wir die `pulseIn()`-Funktion.

PULSEIN()-FUNKTION

Diese Funktion liest einen Wert von einem Pin des Arduino ein (`LOW` bzw. `HIGH`). Wenn der Wert z.B. `HIGH` ist, wartet `pulseIn()` darauf, dass der Pin auf den Wert `HIGH` wechselt, startet einen Timer und wartet anschließend darauf, dass der Pin wieder auf `LOW` wechselt. Daraufhin stoppt `pulseIn()` den Timer und gibt die Länge des Pulses in Mikrosekunden zurück.

```
pulseIn(Pin, Wert)
```

Somit ergibt sich für unsere Zeitdauer:

```
zeitdauer = pulseIn(trigger, HIGH);
```

Die Berechnung ist dir ja bereits bekannt, was es jetzt noch zu beachten gilt, ist die richtige Einheit. Wir werden eine Zeitdauer von einigen Mikrosekunden erhalten, damit dies bezüglich der Dimensionen der Einheiten mit unserer Berechnungsformel konform ist, müssen wir die Schallgeschwindigkeit umrechnen:

$$s = 342{,}2 \frac{m}{s} = 0{,}03432 \frac{cm}{\mu s}$$

Die Mikrosekunden werden sich dann wegkürzen und wir erhalten unsere Entfernung in der Einheiten Zentimeter. Im Quellcode sieht dies wie folgt aus:

```
entfernung = (dauer/2) * 0.03432;
```

ACHTUNG

Zunächst musst du darauf achten, dass die Programmiersprache keine Kommas kennt bei gebrochenen Zahlen. Dies wird mit Hilfe eines Punktes repräsentiert. Auch das dir bekannte Zeichen für „geteilt-durch“ suchst du vergebens auf deiner Taster, im Code wird dies einfach mit Hilfe eines Schrägstriches (im engl. slash) „/“ ausgedrückt.

Jetzt wollen wir natürlich die Funktionalität testen. Gib dazu die Entfernung im seriellen Monitor aus.

```
Serial.println(entfernung);
```

AUFGABE 4 – ENTFERNUNG BESTIMMEN

Implementiere den oben beschriebenen Code zur Bestimmung der Entfernung. Ergänze im Programmcode die Berechnung für die Entfernung und lass dir diese im seriellen Monitor ausgeben. Teste dein Programm in dem du Entfernungen misst und deine Ergebnisse mit dem Lineal kontrollierst.

BREAKPOINTS FÜR WARNHINWEISE

Das Bestimmen der Entfernung funktioniert bereits. Jetzt wollen wir ja aber einen Rückfahrwarner konstruieren und müssen sogenannte Breakpoints festlegen. Diese versehen wir dann mit spezifischen Warnungen, in Abhängigkeit der aktuellen Entfernung zum Sensor.

Um dies zu realisieren nutzen wir die **einfache Verzweigung** und die logischen Vergleichsoperatoren **UND** „**&&**“, sowie **ODER** „**||**“. Zu Beginn wollen wir erstmal fehlerhafte Werte abfangen, dazu zählt alles was über oder unter dem Wertebereich des Sensors liegt. Sprich alle Entfernung unter 0 Zentimetern und über 300 cm. Im Code sieht dies dann wie folgt aus:

```
if (entfernung <= 0 || entfernung >=300){
    Serial.println("fehlerhafte Messung – kein Ergebnis");
}
```

Weitere Regeln kannst du mit weiteren Verzweigungen und Bedingungen festlegen mit Hilfe von:

```
else if(entfernung <= 300 && entfernung >= XY){
     Serial.println( … deine Warnung … );
}
```

AUFGABE 5 – BREAKPOINTS FESTLEGEN

Ergänze deinen Code um die oben benannte Verzweigung. Ergänze danach weitere Punkte mit spezifischen Warnungen. Diese sollten ungefähr der Realität entsprechen – je näher man dem Hindernis kommt, desto ausdrucksvoller muss die Warnung sein.

Teste auch hier wieder mit einem Hindernis in verschiedenen Entfernungen zum Sensor und überprüfe mit einem Lineal.

DU HAST NOCH NICHT GENUG?

Es gibt noch eine Zusatzaufgabe zur Station. Dort wird es darum gehen, dass textuelle Feedback zu ersetzen. Denn für einen Einsatz im Auto sind natürlich nur optische und akustische Warnungen geeignet! - Interesse? Dann frage die Betreuer nach dem Zusatzblatt!
