Station 2 – Einparkhilfe

„Wenn's knallt, noch'n Meter …"

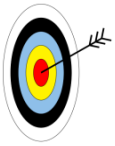
Jeder von euch, der demnächst in der Fahrschule seine Runden dreht, wird ein „Schreckensszenario" kennenlernen: rückwärts einparken! Man dreht und wendet das Auto hin und her, steht letzten Endes doch schief und wendet für keine Sekunde den Blick von den Spiegeln ab – aus Angst, das eigene Heck küsst die Stoßstange des Hintermannes.

Abb. 1: „Parking Assist" von Nozilla

Was für ein Glück, dass in immer mehr Fahrzeugen Einparkhilfen eingebaut sind, die durch ihr drängendes und immer schneller werdendes Piepen vor dem drohenden Zusammenstoß warnen!

Doch wie funktioniert so eine Einparkhilfe? Woher kennt das Fahrzeug den Abstand? Und wie wird aus dem Abstand ein Warnton?

Die Antwort liegt im Verborgenen: „unsichtbares Licht"! Oder genauer gesagt Infrarot-Licht (kurz: IR), das euch bestimmt von Infrarot-Fernbedienungen bekannt ist. Licht im sogenannten Infrarot-Spektrum ist für das menschliche Auge nicht sichtbar. Es wird von einem Sensor ausgesendet, von einer Oberfläche reflektiert und anschließend von einem Empfänger registriert. Da es in Tinkercad keinen IR-Sensor gibt, werdet ihr eine Einparkhilfe mithilfe eines Ultraschallsensors (kurz: US) nachbauen, was im Wesentlichen genauso funktioniert.



In diesem Arbeitsblatt lernt ihr,…

- wie man mit Ultraschall Abstand messen kann.
- wie aus dem Abstand ein Signalton erzeugt wird.
- wie man das mit dem Arduino-Mikrocontroller nachstellt.

Damit verhindert ihr dann hoffentlich, dass Autos eine verbeulte Stoßstange bekommen.

BENÖTIGTE BAUTEILE

Die Schaltung der Einparkhilfe auf dem Arduino-Board ist nicht kompliziert, ihr benötigt lediglich – natürlich neben Arduino und Steckbrettern – …

- einen 110 Ω Widerstand,
- einen Piezo-Signalgeber und
- einen Ultraschall-Abstandssensor.

Abb. 2: Widerstand

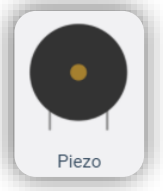


Abb. 3: Piezo-Signalgeber

Abb. 4: Ultraschallabstandssensor

Station 2 – Einparkhilfe

DER US-ABSTANDSSENSOR

Der **US-Abstandssensor** ist ein komplexes Bauteil mit integrierter Verschaltung, das **Entfernungen in einem Bereich von ca. 2 - 300 cm** recht genau erkennen kann.

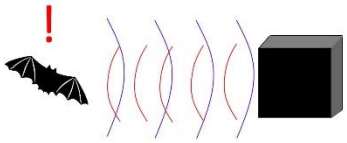


Abb. 5: Schaubild der Funktionsweise

Der **Sensor** orientiert sich so, wie ihr es von Fledermäusen kennt. Er **sendet** ein Ultraschallsignal in eine Richtung und wartet darauf, das gesendete Signal wieder selbst zu **empfangen**. Jedes Hindernis **reflektiert** die Ultraschallwellen wieder zurück. Eine Fledermaus kann sich so ein sehr detailliertes Bild von ihrer Umgebung machen, aber unser Sensor kann so leider nur den Abstand ermitteln.

DER PIEZO-SIGNALGEBER

Der akustische **Piezo-Signalgeber** ist ein sogenannter „Summer“ oder „Pieper“, ein kleines Bauteil, das elektronisch angesteuert wird und einen bestimmten **Ton** erzeugt – es ist also unsere akustische Ausgabe. Einsatz findet er überall dort, wo zur **Warnung** oder **Benachrichtigung** schnell laute Töne erzeugt werden müssen – z. B. beim Rauchmelder, in der Mikrowelle oder eben als Piepton einer Einparkhilfe am PKW.

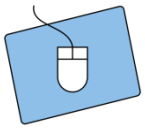
Abb. 6: Piezo-Signalgeber

DIE ANALOGEN PINS

Bisher kennt ihr nur **digitale Pins**, über die z. B. eine LED angeschlossen wird. An diesen Pins können nur **binäre Werte** ein- und ausgelesen werden – also ein/aus, 0/1 oder eben die bekannten Werte HOCH und NIEDRIG für eine hohe bzw. niedrige Spannung.
Ein Distanz-Sensor z. B. kann aber nicht nur zwei Werte annehmen, sondern misst im Bereich zwischen dem minimalen und maximalen Wert ganz viele **Zwischenwerte**. Das realisieren am Arduino **analoge Pins** (*A0 bis A5*), an denen ein ganzer **Bereich** gemessen werden kann.

Station 2 – Einparkhilfe

JETZT KÖNNT IHR EUCH ANS ZUSAMMENBAUEN BEGEBEN!
Die folgenden Schritte helfen euch:



1. Zieht zunächst ein **Steckbrett** auf die Fläche, und schließt den **Plus-** und **Minuspol** entsprechend an dem Arduino an (**Minus** zu **GND**, **Plus** zu **5V**). Nutzt dazu ein blaues Kabel zur Verbindung zum Minuspol und ein rotes zur Verbindung zum Pluspol.
2. Zieht nun einen **Ultraschall-Abstandssensor** auf das Steckbrett, und verbindet **GND** mit dem **Minuspol** und **5V** mit dem **Pluspol**. **SIG** soll mit einem **Digitalen Pin** verbunden werden.
   [***Hinweis:*** *Pin 0 und Pin 1 dürfen nicht verwendet werden.]*
3. Zieht nun einen **Piezo** auf die Fläche, und verbindet „negativ“ mit **GND** und **positiv** mit einem **digitalen Pin.**

[Hinweis: *Eure Lösung kann anders aussehen (z. B. die Belegung der Pins) als auf den folgenden Abbildungen. Das ist aber nicht schlimm, da es keine eindeutige Lösung gibt!]*

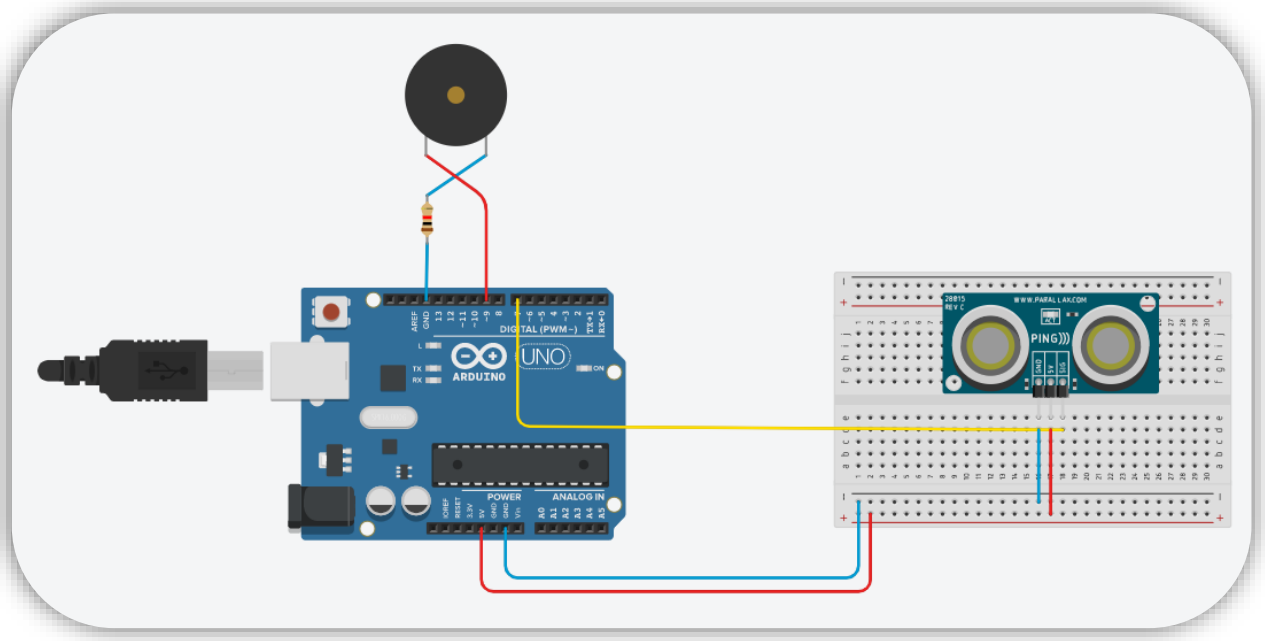
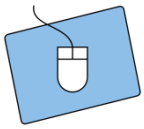


Abb. 7a: Komplette Schaltung (Fritzing)

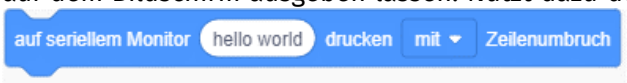
Station 2 – Einparkhilfe

In diesem Schritt lernt ihr,...

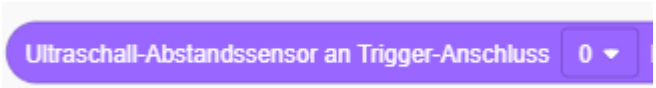
- **Sensorwerte** über einen analogen Pin einzulesen und
- diese Werte auf dem **seriellen Monitor** auszugeben.



1. Begebt euch nun wieder in den Codebereich, und wählt den Blöcke-Modus.
2. Achtet darauf, an welche Pins euer Ultraschallsensor und der Piezo angeschlossen sind.
3. Jetzt füllt ihr eure Bauteile mit Leben.
   a. Zunächst werdet ihr euch die Distanz, die der Ultraschallsensor misst, auf dem Bildschirm ausgeben lassen. Nutzt dazu den folgenden Block:

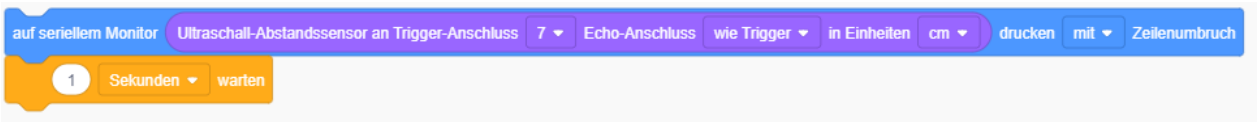


   b. Zieht nun diesen Block ins Feld:



   c. Stellt nun hinter „Trigger-Anschluss“ den Anschluss ein, an dem euer Ultraschallsensor angeschlossen ist.
   d. Damit die Werte später nicht viel zu schnell auf dem seriellen Monitor angezeigt werden, müsst ihr zum Schluss noch eine Pause hinter dem Ausgabebefehl einfügen, sodass die Werte nur **alle ein oder zwei Sekunden** aktualisiert werden.
4. Testet euren Code! Schließt den Arduino wieder an, führt das Programm aus, und beobachtet die Sensor-Werte.

Insgesamt sollte euer Programm jetzt so aussehen:



Wie ihr bald feststellen werdet, bewegen sich die gemessenen **Distanz-Werte** des Sensors nicht in dem angegebenen Zentimeter-Bereich von ca. **2-300 cm**. Überprüft einmal, welche Werte ihr annehmen könnt, und tragt euer Ergebnis unten ein.

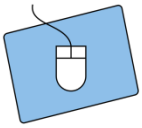
MIN = _____ **MAX = _____**

Ein einfacher Warnton

Nach diesem kurzen Test geht es jetzt wieder mit dem **Programmieren der Schaltung** weiter, denn: Der Sensor misst zwar die Abstände, aber einen Warnton hört ihr noch nicht. Dazu benutzt ihr das zweite neue Bauteil, den **Piezo-Signalgeber**.

Im nächsten Schritt lernt ihr,...

- den **Piezo-Signalgeber** in einen Sketch einzubinden und anzusteuern.
- in Abhängigkeit eines Sensor-Wertes ein **akustisches Signal** zu erzeugen.



1. Überprüft, an welchen Pin euer Signalgeber angeschlossen ist, und tragt ihn hier ein: _____
2. Ihr benötigt jetzt die bekannten Wenn-dann-Anweisungen aus dem Einstiegsprojekt. Unten ist noch einmal erklärt, wie ihr sie genau verwendet.

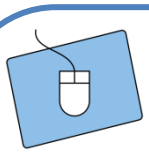
WENN-DANN-ANWEISUNGEN

Möchte man in einem Sketch definieren, dass etwas nur unter einer bestimmten **Bedingung** passieren soll – z. B. dass ein Piepton erst bei zu geringem Abstand ertönt –, bedient man sich der sogenannten **Wenn-dann-Anweisungen**. So werden sie benutzt:

In die rautenförmige Lücke könnt ihr eine Bedingung einfügen. Wenn diese Bedingung wahr ist, wird der erste Programmblock ausgeführt. Wenn die Bedingung falsch, wird das ausgeführt, was hinter „sonst“ steht. Dabei sind mit Bedingungen meist **mathematische Vergleiche (>, <, >=, <=, ==)** o. Ä. gemeint. Es gibt aber auch den **logischen Operator && [und],** um zu überprüfen, ob zwei Bedingungen gleichzeitig wahr sind:

Um die Bedingungen klar zu definieren, müsst ihr euch aber vorab überlegen, welche Grenzen sinnvoll sind. Ab wann sollte der Einparksensor einen Ton von sich geben?

Station 2 – Einparkhilfe



1. Jetzt gilt es, eine Wenn-dann-Anweisung zu benutzen. Legt dazu die Grundstruktur wie oben beschrieben an.
2. Ihr wollt, dass der Piezo nur piepst (einen **DAUERNDEN TON** erzeugt), wenn der Abstand zum Ultraschallsensor zu gering wird (hier **UNTER 100 CM**). Welche Variable braucht ihr also, um diese Bedingung zu formulieren?
3. Falls eure Bedingung wahr ist, der Abstand also zu gering, soll ein Dauerton erklingen. Dazu muss der **AUSGANGSPIN,** an dem der Signalgeber angeschlossen ist, auf **HIGH** gesetzt werden (was genauso funktioniert wie bei einer LED).
4. Im anderen Fall soll der Ton natürlich nicht zu hören sein. Überlegt euch, welcher Befehl den Piezo wieder „ausschaltet“, und vollendet so die Wenn-dann-Anweisung. Nutzt das Schema, um eure Überlegungen aufzuschreiben:

```
Wenn( _____ _____ _____), dann {
        _____;
}
sonst {
        _____;
}
```

5. Fertig! Testet euer Programm jetzt aus, indem ihr, während es läuft, auf den Sensor klickt und den Kreis verschiebt.

UND JETZT?

Habt ihr weitere Ideen oder Verbesserungsvorschläge? Notiert eure Ideen. Eurer Kreativität sind keine Grenzen gesetzt.
In den nächsten beiden Teilen werdet ihr eure Einparkhilfe immer weiter optimieren.

Station 2 – Einparkhilfe

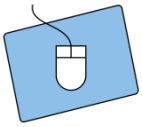
„Bei euch piepst's wohl!"

Oder auch nicht... Denn bisher habt ihr das Projekt „Einparkhilfe" auf den Stand gebracht, dass bei zu geringer Entfernung ein **Dauerton** ertönt. Ein Fahrer sollte aber auch durch die Einparkhilfe ungefähr **abschätzen können, wie viel Platz** er noch hat, ob also noch keine Gefahr besteht, er sich dem Hindernis nähert oder schon ganz kurz davor ist.

Das alles soll durch den Piezo realisiert werden, indem dieser einfach unterschiedlich schnell **piepst**.

In der Optimierung lernt ihr...,

- anhand der Sensor-Werte **Abstandsbereiche** festzulegen und
- mit Hilfe von Pausen einen **Piepton** zu erzeugen.



1. Eure einzige „Baustelle" dieses Mal ist das Programm, genauer gesagt der Teil mit der Wenn-dann-Anweisung:
   a. Überlegt euch eine passende **Einteilung der Sensor-Werte und Zentimeter** in einem Bereich, in dem kein Ton erklingt, und zwei Bereichen mit unterschiedlichen Pieps-Geschwindigkeiten.

	Bereich 1 - keine Gefahr -	Bereich 2 - Näherung -	Bereich 3 - Gefahr -
Zentimeter	>_____	_____-_____	<_____
Sensor-Werte	<_____	_____-_____	>_____

   b. Nun müsst ihr noch eure Wenn-dann-Anweisung aus dem ersten Teil ein wenig anpassen und die eben bestimmten Bereiche einbauen. Dazu werden aus der einen Wenn-dann-Anweisung (die ja nur zwei Bereiche unterscheiden kann) drei einzelne Wenn-dann-Anweisungen.

*[**Hinweis:** Nutzt die **mathematischen Vergleiche** und vor allem das **logische „und"** sinnvoll, und verändert die **Geschwindigkeit** des Piepsens über Pausen mit unterschiedlichen Werten!]*

Für jeden der drei Bereiche benötigt ihr eine eigene Wenn-dann-Anweisung nach dem folgenden Schema:

```
Wenn(_____ ____ _____), dann {
        Anschluss _ Auf ___ Einstellen
}
Wenn(_____ ____ _____ && _____ ____ _____), dann {
        Anschluss _ Auf ___ Einstellen
        __ Millisekunden warten
        Anschluss _ Auf ___ Einstellen
}
Wenn(_____ ____ _____), dann {
        Anschluss _ Auf ___ Einstellen
        __ Millisekunden warten
        Anschluss _ Auf ___ Einstellen
}
```

2. Testet nun euer Programm und überprüft eure verbesserte Einparkhilfe. Passt ggf. die gewählten Bereiche oder die delays so an, sodass ihr mit dem Ergebnis zufrieden seid.

Station 2 – Einparkhilfe

Geschafft?!? Herzlichen Glückwunsch! Dank eurer Hilfe kann das Testfahrzeug jetzt ganz einfach rückwärts einparken.

Wenn ihr wollt, gibt es noch eine **KLEINE HERAUSFORDERUNG** für euch!
FRAGT EINFACH EINEN BETREUENDEN NACH DEM BONUS-BLATT!

Quellenverzeichnis:
Abb. 1 – *Quelle: Nozilla „Parking Assist“ CC BY-SA 2.5/ verändertes Original*
Abb. 2, 3, 4, 5, 6, 7b, 8 – *Quelle: InfoSphere*
Abb. 7a – *Quelle: Screenshots der Fritzing-Software (http://fritzing.org)*
Alle Codeblock-Screenshots – *Quelle: Screenshots von Tinkercad https://www.tinkercad.com/*
, , , angefertigt vom InfoSphere-Team