Station 2 – Einparkhilfe

„Wenn's knallt, noch'n Meter …"

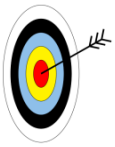
Jeder von euch, der demnächst in der Fahrschule seine Runden dreht, wird ein „Schreckensszenario" kennenlernen: rückwärtseinparken! Man dreht und wendet das Auto hin und her, steht letzten Endes doch schief und wendet für keine Sekunde den Blick von den Spiegeln ab – aus Angst, das eigene Heck küsst 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 ausgesandt, von einer Oberfläche reflektiert und anschließend von einem Empfänger registriert.
Weil Tinkercad keinen IR-Sensor hat, benutzen wir einen Ultraschallsensor. Der funktioniert sehr ähnlich zu einem IR-Sensor und ist für das menschliche Gehör in der Regel nicht hörbar.



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 US-Einparkhilfe auf dem Arduino-Board ist nicht kompliziert, ihr benötigt lediglich – natürlich neben Arduino und Steckbrettern:
- einen 110 Ω Widerstand
- einen Piezo-Signalgeber
- 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, unser Sensor kann so leider nur den Abstand ermitteln.

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.

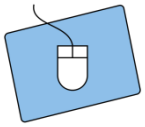
Abb. 6: Piezo-Signalgeber

ANALOGE 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 HIGH und LOW für eine hohe bzw. niedrige Spannung.
Ein Distanz-Sensor kann aber nicht nur zwei Werte annehmen, sondern misst ja 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 den 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 **DIGITALPIN** 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 **DIGITAL PIN.**

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

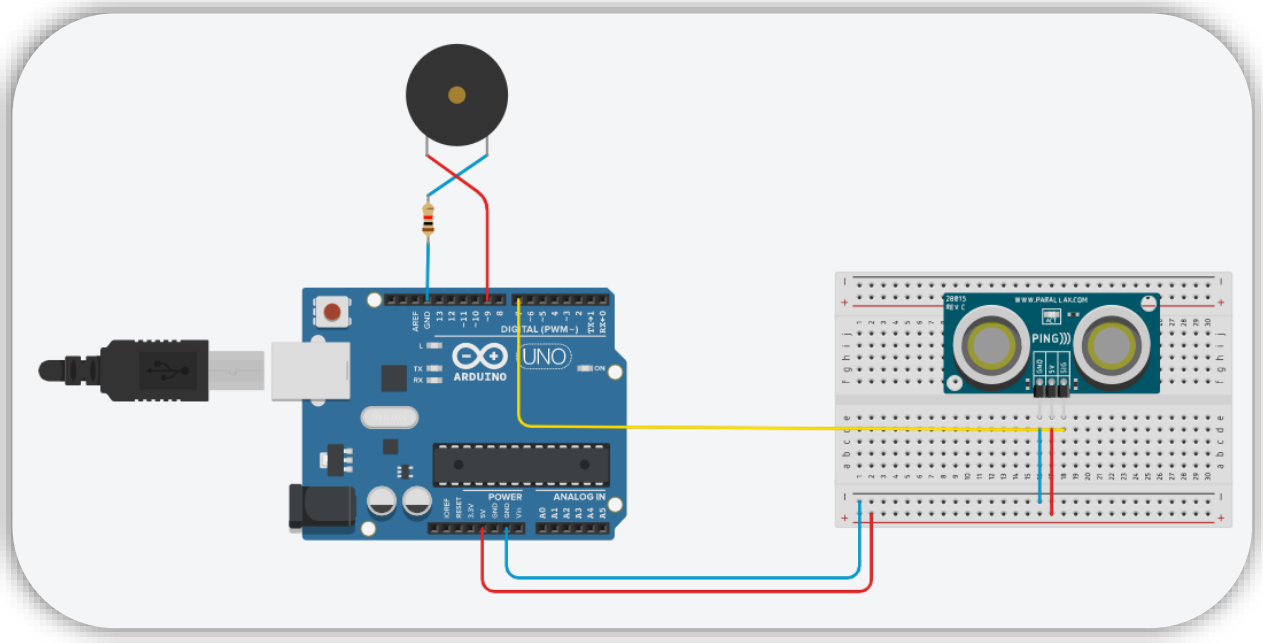


Abb. 7a: Komplette Schaltung (Fritzing)

Station 2 – Einparkhilfe

In diesem Schritt lernt ihr, ...

- **SENSOR-WERTE** über einen analogen Pin einzulesen und
- diese Werte auf dem **SERIEN MONITOR** auszugeben.

1. Begebt euch nun wieder in den Code-Bereich (zur Erinnerung: den findet ihr oben rechts). Das Grundgerüst sollte wieder stehen.
2. Bevor ihr setup() und loop() mit Leben füllt, definiert (wie im Einführungsprojekt) eine Variable vom Typ int für den analogen **ULTRASCHALLPIN**, und weist ihr die entsprechende Nummer des Pins zu.
   *[**Hinweis:** Auch wenn die analogen Pins mit A0, A1 usw. bezeichnet sind, benutzt man für die Zuweisung im Sketch nur den Zahlenwert!]*
3. Jetzt müsst ihr nur noch zwei weitere Variablen deklarieren, in die ihr die Sensor-Daten speichern werdet. Eine Variable soll den gemessenen Wert enthalten, die andere die Distanz in cm. Denkt hier, wie bei Schritt 2, an aussagekräftige **VARIABLEN-NAMEN**.
4. Nun zu setup()
   Wie wird die Übertragung zum Serien Monitor gestartet? Erinnert ihr euch?
5. Füllt jetzt loop() mit Leben.
   a. Der wichtigste Schritt ist das Senden des Signals. Dazu müssen wir den Pinmode auf **OUTPUT** setzen, für zwei Millisekunden ein **LOW**, DANN FÜR 5 MILLISEKUNDEN EIN **HIGH** schreiben. Jetzt müsst ihr nur noch den Pinmode wieder auf input setzen und den Sensor auslesen:
   sensorDatenVariable = pulseIn(sensorPinVariable, HIGH);
   b. Nun folgt noch die Ausgabe auf dem Seriellen Monitor, die wieder durch Serial eingeleitet wird. Setzt in die Anführungszeichen eine kurze Beschreibung dessen, was angezeigt werden soll.
   c. Fügt eine zweite Ausgabe hinzu, die aber jetzt in den Klammern die Variable enthält, die die Sensorwerte gespeichert hat. Dieses Mal ohne Anführungszeichen, da diese nur bei Text verwendet werden! In diesem Fall wollen wir aber die Sensorwerte anzeigen lassen, die in der Variable gespeichert sind.
   d. Damit die Werte später nicht viel zu schnell auf dem Seriellen Monitor angezeigt werden, solltet ihr noch ein delay() hinter die Ausgabebefehle setzen, sodass die Werte nur **ALLE EIN ODER ZWEI SEKUNDEN** aktualisiert werden.
6. Testet euren Code! Schließt den Arduino wieder an, führt das Programm aus, und beobachtet die Sensorwerte.

1 Sek. = 1000 Millisek.

Wie ihr bald feststellen werdet, bewegen sich die gemessenen **DISTANZWERTE** des Sensors nicht in dem angegebenen Zentimeter-Bereich von ca. **2-300 CM**. Welche Minimal- und Maximalwerte könnt ihr registrieren?

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 Sensorwertes ein **AKUSTISCHES SIGNAL** zu erzeugen.

1. Erstellt für den Piezo eine int–Variable und weist ihr den **DIGITALEN AUSGANGSPIN** zu, über den das Bauteil mit dem Arduino verbunden ist.
2. In setup() müsst ihr den Pin des Piezos als **AUSGANG** definieren – genauso wie ihr das mit der LED im Einstiegsprojekt gemacht habt:
   pinMode(_____, _____);
3. In loop() benötigt ihr jetzt die bekannten if-else-Anweisungen aus dem Einstiegsprojekt.

if-else -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 if-else-**ANWEISUNGEN**. So werden sie benutzt:

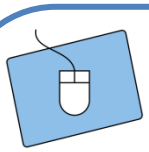
```
if(bedingung){
        ANWEISUNGEN;
}
else {
        ANDERE ANWEISUNGEN;
}
```

Falls die Bedingung wahr ist, wird der erste Programmblock ausgeführt, und wenn die Bedingung falsch ist, der zweite. Dabei sind mit bedingung meist **MATHEMATISCHE VERGLEICHE (>, <, >=, <=, ==)** o. Ä. gemeint. Es gibt aber auch den **LOGISCHEN OPERATOR && [UND],** um zu überprüfen, ob zwei Bedingungen gleichzeitig wahr sind:

```
 if(bedingung1 && bedingung2){
        ANWEISUNGEN;
}...
```

Um die Bedingungen klar zu definieren, müssten wir aber wissen, was genau die gemessenen Werte bedeuten. Dazu brauchen wir ein bisschen einfache Physik. PulseIn() gibt uns die Anzahl der Mikrosekunden an, die vergehen, bis wir ein Signal messen. Schall braucht 29 Mikrosekunden pro Zentimeter und muss hin und zurückreisen.
Diese Angaben sollten dir reichen, um den gemessenen Sensorwert in Zentimeter umzurechnen.

Station 2 – Einparkhilfe



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

```
if( _________________ ______ _________________) {
        ____________________________________;
} else {
        ____________________________________;
}
```

8. 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? Diskutiert sie in der Gruppe, und notiert sie.
In den nächsten beiden Teilen werdet ihr eure Einparkhilfe aber 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 Sensorwerte **ABSTANDSBEREICHE** festzulegen.
- durch digitalWrite(pinName, **ZUSTAND**); und delay() einen **PIEPTON** zu erzeugen.

1. Eure einzige „Baustelle" dieses Mal ist in loop(), genauer gesagt in dem Teil mit der if-else-Anweisung:
   a. Überlegt euch eine passende **EINTEILUNG DER SENSORWERTE UND ZENTIMETER** in einem Bereich, in dem kein Ton erklingt, und zwei Bereiche mit unterschiedlichen Pieps-Geschwindigkeiten. Nutzt dazu einen Taschenrechner.

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

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

*[**Hinweis:** Nutzt die **MATHEMATISCHEN VERGLEICHE** und vor allem das **LOGISCHE &&** sinnvoll, und verändert die **GESCHWINDIGKEIT** des Piepsens über delays mit unterschiedlichen Werten!]*

Für jeden der drei Bereiche benötigt ihr eine eigene if-Anweisung nach dem Schema:

```
if(_____ ____ _____) {
        digitalWrite(_____,_____);
        delay(_____);
        digitalWrite(_____,_____);
}
if(_____ ____ _____ && _____ ____ _____) {
        digitalWrite(_____,_____);
        delay(_____);
        digitalWrite(_____,_____);
}
if(_____ ____ _____) {
        digitalWrite(_____,_____);
        delay(_____);
        digitalWrite(_____,_____);
}
```

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

GESCHAFFT?!?

Herzlichen Glückwunsch! Ihr habt es geschafft! Dank eurer Hilfe kann das Testfahrzeug jetzt ganz einfach rückwärtseinparken.

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. 5, 6 – *Quelle: InfoSphere*
Abb. 2, 3, 4, 7a – *Quelle: Screenshots von Tinkercad https://www.tinkercad.com/*
, , , angefertigt vom InfoSphere-Team