

## Station 2 – Einparkhilfe

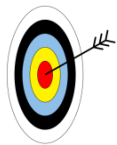
### „Wenn's knallt, noch'n Meter ...“

Ihr kennt das sicherlich von euren Eltern. Sie wollen rückwärts einparken, können aber leider das Auto hinter sich nicht gut sehen. Zum Glück sind in immer mehr Fahrzeugen Einparkhilfen eingebaut, die durch ihr drängendes und immer schneller werdendes Piepen vor dem drohenden Zusammenstoß warnen.



[1]

Doch wie funktioniert so eine Einparkhilfe? Woher weiß das Fahrzeug den Abstand? Und wie wird aus dem Abstand ein Warnton? Die Antwort liegt im Verborgenen: „unsichtbares Licht“! Oder genauer gesagt: Infrarot-Licht, das euch bestimmt von Fernbedienungen, die mit Infrarot (IR) funktionieren, 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. Mit dem Arduino werdet ihr nachstellen, wie daraus eine Einparkhilfe wird.



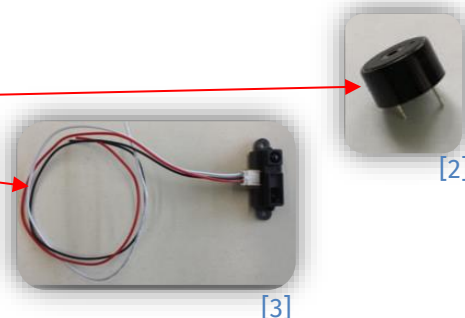
In diesem Arbeitsblatt lernt ihr,...

- ✗ den IR-Sensor zur Distanzmessung zu verwenden,
- ✗ Werte über den Seriellmonitor auszugeben,
- ✗ und den Piezo-Signalgeber zu steuern.

### Benötigte Bauteile

Die Schaltung der IR-Einparkhilfe auf dem Arduino-Board ist nicht kompliziert. Neben dem Arduino benötigt ihr lediglich die folgenden Teile:

- 2 Verlängerungskabel (rot und blau)
- 1 Piezo-Signalgeber
- 1 IR-Sensor



### Der IR-Sensor

Der IR-Sensor kann Entfernungen in einem **Bereich von ca. 10 bis 80 cm** recht genau erkennen.

## Station 2 – Einparkhilfe

**Der Piezo-Signalgeber**

Der akustische Piezo-Signalgeber ist ein sogenannter „Summer“ oder „Pieper“. Es handelt sich um ein kleines Bauteil, das elektronisch angesteuert wird und einen bestimmten Ton erzeugt. Der Piezo ist also eure **akustische Ausgabe**.

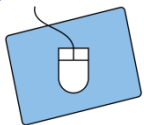
**Die analogen Pins**

Bisher kennt ihr nur die **digitalen 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 IR-Sensor 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).

*Jetzt könnt ihr mit dem Bau der Schaltung loslegen! Die folgenden Schritte helfen euch bei der Verkabelung.*

**Der Bau der Schaltung**

**Hinweis:** Die Steckbretter benötigt ihr dieses Mal nicht.



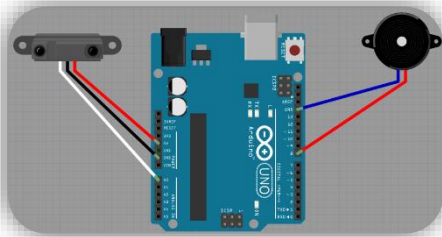
1. Verbindet den **Minus-Pol** (das kürzere Beinchen) des Signalgebers **direkt** über das blaue Verlängerungskabel mit dem **GND-Pin des Arduinos** (bei den digitalen Anschlüssen).
2. Schließt den **Plus-Pol** (das längere Beinchen) des Signalgebers über das rote Verlängerungskabel an einen **digitalen Pin** an, über den ihr die Ausgabe steuern werdet.  
**Hinweis:** Pin 0 und Pin 1 dürfen nicht verwendet werden.
3. Der IR-Sensor wird **direkt am Arduino** angeschlossen, mit dem schwarzen Kabel bei **GND** (bei den analogen Anschlüssen), dem roten Kabel am **5V-Anschluss** und dem weißen Kabel an einem **analogen Pin**, den ihr als Eingangspin für die Abstandswerte benutzt.



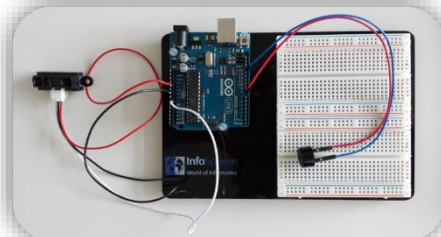
## Station 2 – Einparkhilfe



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



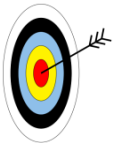
[5]



[6]

### Manuelle Abstandsmessung

Zuallererst werdet ihr jetzt die Daten, die der IR-Sensor misst, über ein kleines Programm einlesen und euch auf dem **Seriellmonitor** ausgeben lassen. Daher ist eure erste Aufgabe, mit dem IR-Sensor Werte zu messen und zu beobachten, wie diese sich ändern, wenn man die Abstände variiert.

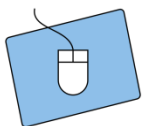


In diesem Schritt lernt ihr,...

- ✗ Sensor-Werte über einen analogen Pin einzulesen
- ✗ und diese Werte auf dem Seriellmonitor auszugeben.



**Hinweis:** Auch wenn die analogen Pins mit A0, A1 usw. bezeichnet sind, benutzt man für die Zuweisung im Sketch nur den Zahlenwert.



1. Öffnet einen neuen Arduino-Sketch und speichert ihn unter einem sinnvollen Namen.
2. Als Erstes müsst ihr die Ausgabe über den Seriellmonitor starten. Den entsprechenden **serial-println**- und **message**-Block findet ihr unter **Kommunikation**. Durch Anklicken könnt ihr den Text „**message**“ in „**entfernung**“ ändern. Euren IR-Sensor findet ihr unter **Input** (Sharp GP2Y0A21 Distanz-Sensor). Damit ihr ihn anhängen könnt, müsst ihr zunächst an den **entfernung**-Block den geeigneten **verbinde**-Block mit **spitzer** Einkerbung anfügen. Nun müsst ihr noch die Pin-Nummer des IR-Sensors an den Block anhängen.

## Station 2 – Einparkhilfe



3. Öffnet einen neuen Arduino-Sketch und speichert ihn unter einem sinnvollen Namen.
4. Als Erstes müsst ihr die Ausgabe über den Seriellmonitor starten. Den entsprechenden **serial-println**- und **message**-Block findet ihr unter **Kommunikation**. Durch Anklicken könnt ihr den Text „**message**“ in „**entfernung**“ ändern. Euren IR-Sensor findet ihr unter **Input** (Sharp GP2Y0A21 Distanz-Sensor). Damit ihr ihn anhängen könnt, müsst ihr zunächst an den **entfernung**-Block den geeigneten **verbinde**-Block mit **spitzer** Einkerbung anfügen. Nun müsst ihr noch die Pin-Nummer des IR-Sensors an den Block anhängen.
5. Jetzt müsst ihr hinter den **serial-println**-Block ein **Warte Millisekunden** mit 1000 ms anfügen. Das kennt ihr schon aus dem Einstiegsprojekt.
6. Es ist an der Zeit, den Sketch zu testen. Führt das Programm aus und beobachtet die Sensor-Werte. Um den Seriellmonitor zu starten, klickt einfach auf den Button **Seriellmonitor**.

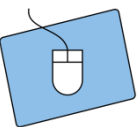


**Hinweis:** Alle Befehle müssen immer in den Wiederhole-fortlaufend-Block eingefügt werden.

Eine Lösung sollte so ähnlich aussehen:



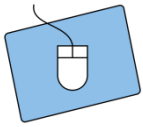
[7]



Misst einmal verschiedene Abstände und beobachtet die Werte auf dem Seriellmonitor.

Wie ihr bald feststellen werdet, bewegen sich die gemessenen **Distanz-Werte** des Sensors nicht linear (1 zu 1) in dem angegebenen Zentimeter-Bereich von ca. **10 bis 80 cm**.

## Station 2 – Einparkhilfe



Welche minimalen und maximalen Werte liefert der Sensor im angegebenen Zentimeter-Bereich?

Minimum = \_\_\_\_\_

Maximum = \_\_\_\_\_

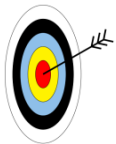
An einem analogen Eingangs-Pin bekommt ihr auf dem Arduino immer Werte im Bereich 0 bis 1023. Diese Werte müsst ihr zuerst in Zentimeter umrechnen. Das ist allerdings nicht so einfach, d. h. es gibt keine einfache mathematische Formel zum Umrechnen.



**Hinweis:** Der Sensor gibt erst ab ca. 10 cm Entfernung einen korrekten Wert aus. Später erhaltet ihr eine Tabelle mit Werten, die ihr verwenden könnt.

### Ein einfacher Warnton

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



Im nächsten Schritt lernt ihr,...

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



1. Speichert euren bestehenden Arduino-Sketch unter einem anderen Namen (Speichern unter). Ihr werdet eure Programme von nun an immer erweitern, verändern und anpassen.
2. Ihr benötigt jetzt den bekannten **falls/sonst-Block** aus dem Einstiegsprojekt.
  - a) Ihr wollt, dass der Piezo-Signalgeber nur einen **dauernden Ton** erzeugt, also piept, wenn der Abstand zum IR-Sensor zu gering wird (hier **unter 40 cm, d. h. Sensorwert > 200**). Das >-Zeichen bedeutet „größer als“ und zählt zu den **logischen Operatoren**.
  - b) Falls diese Bedingung wahr ist, dann soll ein Dauerton erklingen. Den **Ton-Block** findet ihr unter **Output**. Denkt daran, die richtige Pin-Nummer anzugeben.
  - c) Falls die Bedingung nicht erfüllt ist, soll kein Ton erklingen.

## Station 2 – Einparkhilfe

So oder ähnlich sollte eure Lösung aussehen:



[8]

### „Bei euch piepst's wohl!“

Oder auch nicht ... Denn: Bisher habt ihr das Projekt zur 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** noch bleibt, 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**.

Im Folgenden findet ihr eine Beispiel-Tabelle zum Vergleich: Zentimeter und Sensor-Werte:

	Bereich 1 (keine Gefahr)	Bereich 2 (Hindernis nähert sich)	Bereich 3 (Hindernis gefährlich nah)
Zentimeter	> 40	40 – 10	< 10
Sensorwerte	< 200	200 – 500	> 500

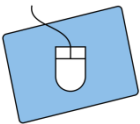
Dieses Mal ist eure einzige „Baustelle“ der Teil mit dem **falls/sonst**-Block: Dazu werden aus dem einen **falls/sonst**-Block (der ja nur zwei Bereiche unterscheiden kann) drei einzelne **falls**-Blöcke.

Den keine-Gefahr-Bereich müsst ihr im Programm nicht betrachten, da der Signalgeber keinen Ton spielen soll. Den Hindernis-nähert-sich-Bereich teilt ihr in zwei kleinere Bereiche auf, damit der Fahrer einschätzen kann, wie nah oder weit weg das Hindernis ist. Wählt dazu z. B. die Sensor-Werte 200 bis 300, wenn das Hindernis noch weiter weg ist und 300 bis 500, wenn das Hindernis näherkommt.

## Station 2 – Einparkhilfe



**Hinweis:** Nutzt die **mathematischen Vergleiche** und das **logische Und** sinnvoll und verändert die Geschwindigkeit des Piepsens über **Warte**-Blöcke mit unterschiedlichen Werten.



1. Der erste **falls**-Block ( nähert sich Hindernis): Hier müsst ihr den Bereich bei der Bedingung durch zwei Werte eingrenzen. Wenn die Bedingung erfüllt ist, dann soll 1000 ms lang ein Ton erklingen und dann für 1000 ms kein Ton erklingen.
2. Der zweite Bereich: Auch hier braucht ihr zwei Grenzen: die 300 und die obere Grenze zur „Gefahr“. Wählt die Wartezeit kürzer, damit das Piepsen schneller wird.
3. Zum Schluss braucht ihr noch die **falls**-Anweisung für den Gefahr-Bereich. Hier braucht ihr nur eine Grenze. Der Piep-Ton soll hier durchgängig sein.
4. Damit seid ihr auch schon bereit für die nächste **Testphase**. Passt die gewählten Bereiche oder die Verzögerungen so an, dass ihr mit dem Ergebnis zufrieden seid.

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



### Quellenverzeichnis:





Abb. 1, , , ,  – Quelle: commons.wikimedia.org  
([https://commons.wikimedia.org/wiki/File:Parking\\_Assist.jpg](https://commons.wikimedia.org/wiki/File:Parking_Assist.jpg)), Autor: Nozilla, CC BY-SA 3.0  
(<https://creativecommons.org/licenses/by-sa/3.0/deed.en>), abgerufen am: 27.06.2022.

Abb. 2 bis 4 – Quelle: InfoSphere, CC BY-SA 4.0 Attribution-ShareAlike 4.0 International  
(<https://creativecommons.org/licenses/by-sa/4.0/>).

Abb. 5 – Quelle: Screenshot der Fritzing-Software (<http://fritzing.org>), CC BY-SA 3.0 Attribution-ShareAlike 3.0 Unported  
(<https://creativecommons.org/licenses/by-sa/3.0/>), abgerufen am: 14.06.2022.

Abb. 6 bis 9 – Quelle: Screenshot der Software Ardublock (<https://arduino-basics.com/ardublock/>), GNU General Public License (<https://github.com/taweili/ardublock/blob/master/LICENSE.txt>), erstellt am 31.01.2023.