Station 2 – Einparkhilfe

„Wenn's knallt, noch'n Meter …"

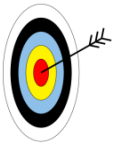
Jeder von euch, der demnächst in der Fahrschule seine Runden drehen wird, wird ein „Schreckensszenario" kennenlernen: rückwärts einparken! Man dreht und wendet das Auto hin und her, steht letzten Endes doch schief und wendet für keine Sekunde den Blick von den Spiegeln ab – aus Angst, das eigene Heck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

[1]

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 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.



In diesem Arbeitsblatt lernt ihr,…

- wie man mit IR-Licht 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 unser Testfahrzeug eine verbeulte Stoßstange bekommt ☺.

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 (Abbildung 2)
- 1 IR-Distanz-Mess-Sensor (Abbildung 3)


[2]


[3]

Station 2 – Einparkhilfe

Der IR-Distanz-Mess-Sensor


[4]

Im Modul benutzt ihr den IR-Distanz-Mess-Sensor *Sharp GP2Y0A21YK*, der ein komplexes Bauteil mit integrierter Verschaltung ist. Entfernungen in einem **Bereich von ca. 10 bis 80 cm** kann er recht genau erkennen.

[5]

Dazu besitzt der Sensor „zwei Augen“: einen Sender und einen Empfänger. Der **Sender** sendet einen IR-Lichtstrahl aus, der von einer Oberfläche reflektiert wird und auf den **Empfänger** trifft – je nach Entfernung in einem unterschiedlichen Winkel. Daraus lässt sich dann mit einem **mathematischen Verfahren** der Abstand berechnen. Das Schaubild links gibt euch einen Überblick hierüber.

Der Piezo-Signalgeber


[6]

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**. 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.

Die analogen Pins

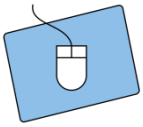
Bisher kennt ihr nur die **digitalen Pins**, über die z. B. eine LED angeschlossen wird. An diesen Pins können nur **binäre Werte** ein- und ausgelesen werden – also ein/aus, 0/1 oder eben die bekannten Werte high und low für eine hohe bzw. niedrige Spannung.
Ein IR-Distanz-Mess-Sensor z. B. kann aber 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.

Jetzt könnt ihr mit dem Bau der Schaltung loslegen, wobei euch die Schritte auf der nächsten Seite helfen werden.

Station 2 – Einparkhilfe

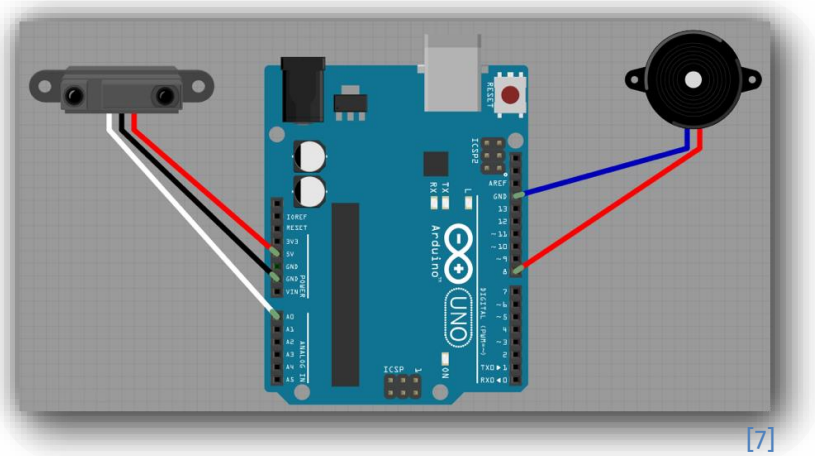
Der Bau der Schaltung

Hinweis: Die Steckbretter benötigt ihr dieses Mal nicht.

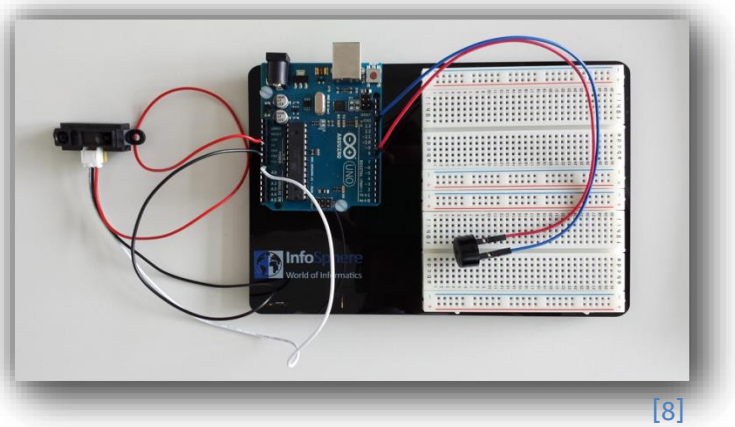


1. Verbindet den Minuspol (das kürzere Beinchen) des Signalgebers **direkt** über das blaue Verlängerungskabel mit dem **GND-Pin des Arduinos** (bei den digitalen Anschlüssen).
2. Schließt den Pluspol (das längere Beinchen) des Signalgebers über das rote Verlängerungskabel an einen **digitalen Pin** an, über den ihr die Ausgabe steuern werdet.
   Hinweis: Pin 0 und Pin 1 dürfen nicht verwendet werden.
3. Der IR-Distanz-Mess-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.

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!



[7]



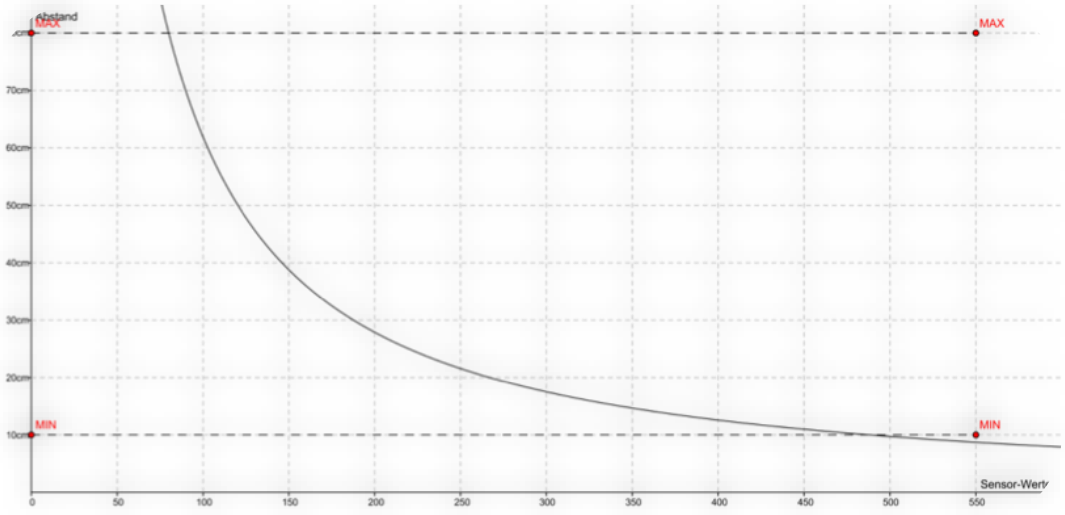
[8]

Station 2 – Einparkhilfe

Manuelle Abstandsmessung

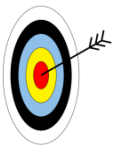
Zuallererst werdet ihr jetzt die Daten, die der IR-Distanz-Mess-Sensor misst, über ein kleines Programm einlesen und euch auf dem **Serial Monitor** ausgeben lassen. Daher ist eure erste Aufgabe, mit dem IR-Distanz-Mess-Sensor Werte zu messen und diese entsprechend der Tabelle am Ende der Aufgaben verschiedenen Zentimeter-Angaben per Hand zuzuordnen. Ihr müsst ja irgendwie feststellen können, wie diese Werte zu den Zentimeter-Angaben passen, die ihr eigentlich für den Abstand benötigt.

In der folgenden Abbildung seht ihr die **Ausgabewerte** des IR-Distanz-Mess-Sensors für die **Abstände** zwischen 10 und 80 cm.

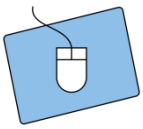


[9]

An einem analogen Eingangspin bekommt ihr auf dem Arduino immer Werte im Bereich **von 0 bis 1023**, die ihr zuerst in Zentimeter umrechnen müsst. Das ist allerdings gar nicht so einfach, d. h. es gibt keine einfache mathematische Formel zum Umrechnen.
Hinweis: Später erhaltet ihr eine Formel zur Umrechnung, die ihr ganz einfach benutzen und mit euren von Hand gemessenen Werten vergleichen könnt.

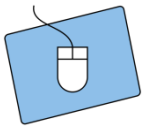


In diesem Schritt lernt ihr,...
- Sensor-Werte über einen analogen Pin einzulesen
- und diese Werte auf dem Serial Monitor auszugeben.



1. Öffnet einen neuen Arduino-Sketch und speichert ihn unter einem sinnvollen Namen.
2. Erstellt das Grundgerüst aus dem Einstiegsprojekt.
3. Bevor ihr `setup()` und `loop()` mit Leben füllt, definiert (wie im Einführungsprojekt) eine **Variable** vom Typ `int` für den analogen **Eingangspin** und weist dieser 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!

Station 2 – Einparkhilfe



4. Jetzt müsst ihr nur noch eine **weitere Variable** deklarieren, in der ihr später die Sensordaten speichern werdet, um schnell und direkt darauf zugreifen zu können. Denkt auch hier an aussagekräftige Variablennamen.
5. Nun zu `setup()`: Wie wird die Übertragung zum Serial Monitor gestartet?
6. Füllt jetzt `loop()` mit Leben.
   a. Der wichtigste Schritt ist das **Einlesen der Sensorwerte**. Statt wie an einem digitalen Pin die Spannung auf high oder low zu setzen, müsst ihr die analogen Werte des Sensors durch den Befehl
   `sensorDatenVariable = analogRead(sensorPinVariable);`
   einlesen. Weist also jetzt der Variablen aus Schritt 4 über „=“ die Werte zu und wählt dabei den richtigen Pin aus.
   _____ = _____(_____);
   b. Nun folgt noch die **Ausgabe** auf dem Serial 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 wollt ihr aber die Sensorwerte anzeigen lassen, die in der Variablen gespeichert sind.
   d. Damit die Werte später nicht viel zu schnell auf dem Serial Monitor angezeigt werden, setzt ihr zum Schluss noch ein `delay()` hinter die Ausgabebefehle, sodass die Werte nur alle ein oder zwei Sekunden aktualisiert werden.
7. Testet euren Sketch! Schließt dazu den Arduino wieder an, führt das Programm aus und beobachtet die Sensorwerte.

Wie ihr bald feststellen werdet, bewegen sich die gemessenen **Distanzwerte** des Sensors nicht in dem angegebenen Bereich von ca. 10 bis 80 cm. In welchem Bereich liegen die Werte?

Minimum = _____ Maximum = _____

Um nicht nur Minimum und Maximum zu kennen, sondern eine aussagekräftige Einteilung der Abstände vornehmen zu können, ist jetzt Maßarbeit gefragt. Nutzt einen langen **Papierstreifen** und einen **Zollstock**, um herauszufinden, welche Sensorwerte bei 10, 20, 30 cm usw. ausgegeben werden. Tragt diese Werte in die **Tabelle** ein!
Hinweis: Der IR-Distanz-Mess-Sensor misst am besten und genausten, wenn man z. B. ein weißes Blatt Papier als reflektierende Oberfläche verwendet!

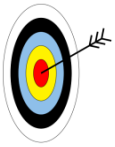
Abstand	10 cm	20 cm	30 cm	40 cm	50 cm	60 cm	70 cm	80 cm
Sensorwert								

Station 2 – Einparkhilfe

Lasst euch beim Messen nicht frustrieren! Auch wenn der Sensor ganz ruhig gehalten wird, kann es passieren, dass er unterschiedliche Werte für ein- und denselben Abstand misst. Entscheidet euch hier einfach für einen **sinnvollen Durchschnittswert**.

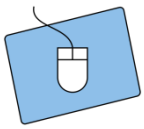
Ein einfacher Warnton

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

Weiter geht's auf der nächsten Seite...

Station 2 – Einparkhilfe

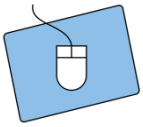
if-else-Anweisungen

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 –, verwendet man die sogenannten **if-else-Anweisungen**. So werden sie benutzt:

```
if(bedingung){
  //Anweisungen, die ausgeführt werden sollen, wenn Bedingung erfüllt;
}
else {
  //Anweisungen, wenn Bedingung nicht erfüllt;
}
```

Dabei sind mit `bedingung` meist mathematische Vergleiche (>, <, >=, <=, ==) o. ä. gemeint. Es gibt aber auch den **logischen Operator** `&&` (logisches Und), um zu überprüfen, ob zwei Bedingungen gleichzeitig wahr sind:

```
if(bedingung1 && bedingung2) ){
  //Anweisungen, die nur ausgeführt werden, wenn beide Bedingungen wahr sind
}
```



5. Jetzt gilt es, eine `if-else`-Anweisung zu benutzen. Legt dazu die Grundstruktur wie oben beschrieben an.
6. Ihr wollt, dass der Piezo nur piepst (einen dauerhaften Ton erzeugt), wenn der Abstand zum IR-Distanz-Mess-Sensor zu gering wird (hier: unter 40 cm). Welche **Variable** braucht ihr also, um diese Bedingung zu formulieren? Seht euch die Tabelle nochmal an: Mit welchem Grenzwert müsst ihr vergleichen?
7. Falls eure Bedingung wahr ist, der Abstand also zu gering, soll ein Dauerton erklingen. Dazu muss der **digitale Ausgangspin** auf **high** gesetzt werden. (Das funktioniert genauso wie bei einer LED.)
8. 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 {
_____;
}
```

9. Fertig! Übertragt den Sketch auf den Arduino und testet ihn. Seid ihr zufrieden mit dem gewählten Abstand? Ertönt der Ton zu spät/zu früh? Nehmt ggf. Anpassungen vor.

Und jetzt?

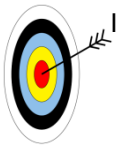
Habt ihr weitere Ideen oder Verbesserungsvorschläge? Diskutiert sie in der Gruppe und notiert sie.
In den nächsten beiden Teilen werdet ihr eure Einparkhilfe aber immer weiter optimieren.

Station 2 – Einparkhilfe

„Bei euch piepst's wohl!"

Oder auch nicht … Denn: Bisher habt ihr das Projekt 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**.



In der Optimierung lernt ihr,…

- anhand der Sensorwerte Abstandsbereiche festzulegen.
- durch `digitalWrite( pinName, zustand);` und `delay()` einen Piepton zu erzeugen.

1. Zusammengebaut ist schon alles. Also geht es los, indem ihr euren alten Sketch unter einem neuen Namen abspeichert.
2. Eure einzige „Baustelle" ist dieses Mal der `loop()`, genauer gesagt der Teil mit der **`if-else`-Anweisung:**
   a. Seht euch noch einmal eure Tabelle auf Seite 5 an und überlegt euch mit Hilfe der folgenden Tabelle eine passende Einteilung der Sensorwerte und Zentimeter. Die folgenden drei Bereiche sollen abgedeckt werden:
   Bereich 1: Die Distanz ist groß. → kein Ton
   Bereich 2: Das Hindernis kommt näher. → Piepen
   Bereich 3: Das Hindernis ist sehr nah. → schnelleres Piepen

	Bereich 1 (große Distanz)	**Bereich 2 (Hindernis nähert sich)**	**Bereich 3 (Hindernis gefährlich nah)**
Zentimeter	>	–	<
Sensorwerte	<	–	>

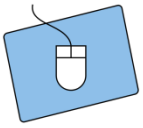
   b. Nun müsst ihr noch eure `if-else`-Anweisung aus dem ersten Teil ein wenig anpassen und die eben bestimmten Bereiche einbauen. Dazu werden aus der einen `if-else`-Anweisung (die ja nur zwei Bereiche unterscheiden kann) **drei einzelne** `if-else`-Anweisungen.

   Hinweis: Nutzt die mathematischen Vergleiche und vor allem das **logische Und** (`&&`) sinnvoll und verändert die Geschwindigkeit des Piepsens über `delays` mit unterschiedlichen Werten. Auf der nächsten Seite findet ihr ein Schema der drei `if-else`-Anweisungen.

Station 2 – Einparkhilfe

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

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



Testet nun euer Programm! Überspielt euren Sketch 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! Dank eurer Hilfe kann das Testfahrzeug jetzt ganz einfach rückwärts einparken. 😊

Wenn ihr wollt, gibt es noch eine kleine Herausforderung. Fragt die Betreuerinnen und Betreuer einfach nach dem Bonus-Blatt. 😉

Quellenverzeichnis:

Abb. 1 – *Quelle: commons.wikimedia.org (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 6, 8, 9 – *Quelle: InfoSphere, CC BY-SA 4.0 Attribution-ShareAlike 4.0 International (https://creativecommons.org/licenses/by-sa/4.0/)*

Abb. 7 – *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*

 , *– Quelle: InfoSphere, CC BY-SA 4.0 Attribution-ShareAlike 4.0 International (https://creativecommons.org/licenses/by-sa/4.0/)*