Station 2 - Einparkhilfe

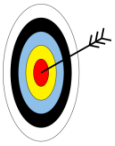
„Wenn's knallt, noch'n Meter.. "

Ihr kennt das sicher von euren Eltern. Sie wollen rückwärts einparken, können aber leider das Auto hinter sich nicht gut sehen. 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.

Abb. 1:" Parking assist' von Nozilla

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

Die Antwort liegt in „unsichtbarem Licht"! Licht im sogenannten Infrarot-Spektrum ist für das menschliche Auge nicht sichtbar, ihr kennt es vielleicht von Infrarot-Fernbedienungen. Es wird von einem Sensor ausgesandt, von einer Oberfläche zurückgestrahlt und anschließend wieder vom Sensor registriert. Wie daraus eine Einparkhilfe wird, werdet ihr hier mit dem **Arduino** nachstellen..



Dafür werdet ihr lernen..

- den Distanzsensor zu verwenden,
- Werte über den seriellen Monitor auszugeben,
- und den Piezo-Signalgeber zu steuern.

Der Aufbau der Schaltung

Benötigte Bauteile

Die Schaltung der Infrarot-Einparkhilfe mit dem Arduino ist nicht kompliziert, ihr benötigt lediglich – natürlich zusätzlich zum Arduino...

- **2 Verlängerungskabel (rot und blau),**
- **einen Piezo-Signalgeber und**
- **einen Infrarot-Distanz-Mess-Sensor.**

Der IR-Distanz-Mess-Sensor

Der **Infrarot-Distanz-Mess-Sensor** kann **Entfernungen in einem Bereich von ca. 10-80 cm** recht genau erkennen.

Station 2 - Einparkhilfe

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.

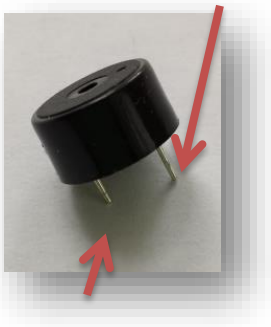
Analoge Pins

Bisher habt ihr nur **digitale Pins** benutzt, 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 ***HIGH*** und ***LOW***.
Ein Distanz-Sensor kann 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 das Zusammenbauen beginnen!

Die folgenden Schritte helfen euch bei der Verkabelung:

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



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

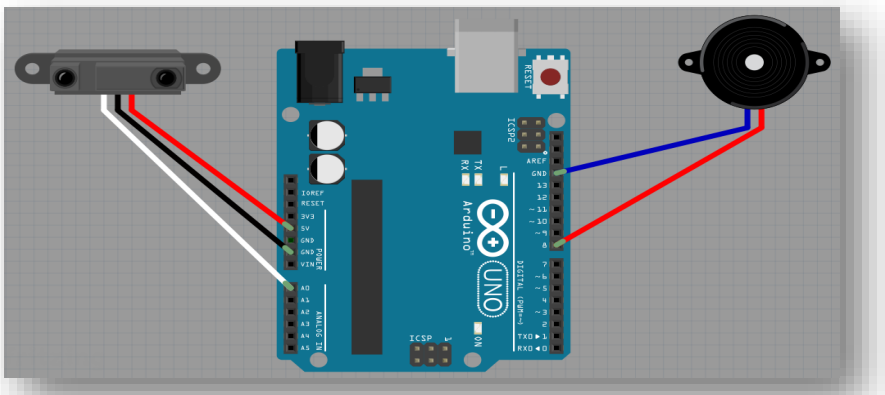


Abb. 3: Komplette Schaltung (Fritzing Schaltplan)

Station 2 - Einparkhilfe

Manuelle Abstandsmessung

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

In diesem ersten Schritt lernt ihr...

- **Sensor-Werte** über einen analogen Pin einzulesen und
- diese Werte auf dem **Seriellen Monitor** 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 seriellen Monitor starten. Den entsprechenden Block 

findet ihr unter 

. Durch Anklicken könnt ihr den Text *message* in *entfernung* ändern. Euren Abstandssensor findet ihr unter 

(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 Distanzsensors an den Block anhängen.
3. Jetzt müsst ihr hinter den serial println-Befehl ein *Warte 1000ms* anfügen, das ihr aus dem Einstiegsblatt schon kennt.
4. Es ist an der Zeit, den Sketch zu testen. Führt das Programm aus und beobachtet die Sensor-Werte. Um den seriellen Monitor zu starten, klickt einfach auf den Button 

.

Hinweis: *Alle Befehle müsse immer in den* 

-Block eingefügt werden.!

Eure Lösung sollte so ähnlich aussehen:

Messt einmal verschiedene Abstände und beobachtet die Werte auf dem seriellen Monitor.

Station 2 - Einparkhilfe

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. **10-80 cm**.

min = _____ **max = _____**

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

An einem analogen **Eingangspin** bekommt ihr auf dem Arduino immer Werte im Bereich **0 bis 1023**, also müsst ihr diese zuerst in Zentimeter umrechnen. Das ist allerdings gar 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. Wir stellen euch später eine Tabelle mit Werten zur Benutzung zur Verfügung, die ihr verwenden könnt.*

Ein einfacher Warnton

Nach dem kurzen Abstecher 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 ein Programm einzubinden und anzusteuern und in Abhängigkeit eines Sensor-Wertes ein **akustisches Signal** zu erzeugen.

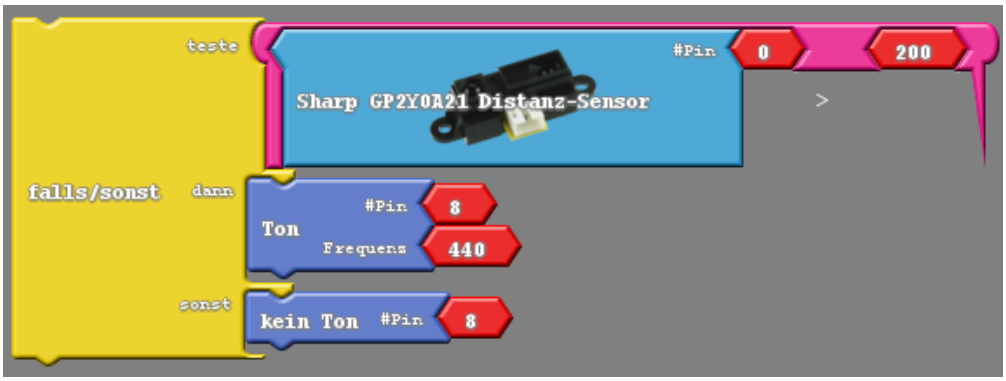
1. Ihr werdet euer Programm von nun an immer erweitern, verändern und anpassen. Damit eure bisherigen Programme nicht verloren gehen, solltet ihr eurer bestehendes ArduBlocks-Programm unter einem anderen Namen speichern („**Speichern unter**“).
2. Ihr benötigt jetzt den bekannten **falls/sonst-Block** aus dem Einstiegsprojekt.

   a. Ihr wollt, dass der Piezo-Signalgeber nur einen **dauernden Ton** erzeugt, also piept, wenn der Abstand zum Infrarot-Sensor zu gering wird (hier **unter 40 cm, d. h. Sensorwert > 200**). Das *„>“-Zeichen* bedeutet „größer als“ und zählt zu den Log. Operatoren.

   b. Falls diese Bedingung wahr ist, dann soll ein Dauerton erklingen. Dazu findet ihr unter Output den *Ton*-Block. Denkt daran, die richtige Pin-Nummer anzugeben.

   c. Falls die Bedingung nicht erfüllt ist, soll kein Ton erklingen.

So oder ähnlich sollte die Lösung aussehen:



Auf zum Endspurt..

„Bei euch piept‘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-Signalgeber realisiert werden, indem dieser einfach unterschiedlich schnell **piept**.

Tabelle zum Vergleich: Zentimeter - Sensorwerte

	Bereich 1 - keine Gefahr -	Bereich 2 - nähert sich Hindernis -	Bereich 3 - Gefahr -
Zentimeter	>40	40-10	<10
Sensor-Werte	<200	200-500	>500

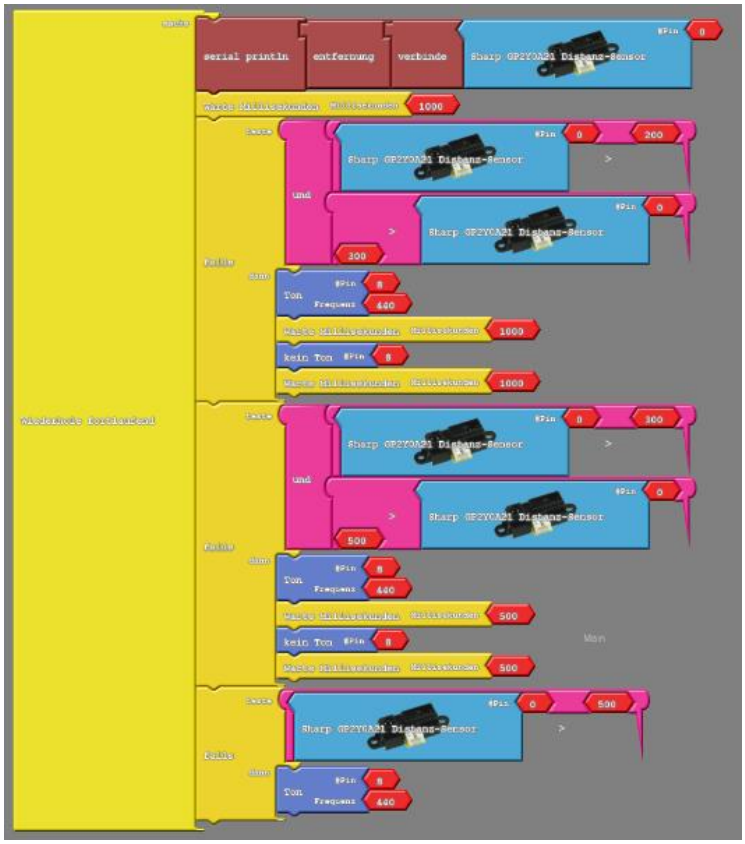
Eure einzige „Baustelle“ dieses Mal ist der Teil mit dem ***falls/sonst***-Block:
Dazu werden aus dem einen ***falls/sonst***-Block (der ja nur zwei Bereiche unterscheiden kann) drei einzelne ***falls***-Blöcke.

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

Hinweis: *Nutzt die **mathematischen Vergleiche** und das **logische „und“** sinnvoll und verändert die **Geschwindigkeit** des Piepens über **Warte**-Blöcke mit unterschiedlichen Werten!*

Station 2 - Einparkhilfe

1. Der erste *falls*-Block „Nähert sich Hindernis“: Dazu 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 dann die obere Grenze zur „Gefahr“. Wählt die Wartezeit kürzer, damit das Piepen schneller wird.
3. Zum Schluss braucht ihr noch die Falls-Anweisung für den Bereich „Gefahr“. Hier braucht ihr nur diese eine Grenze. Der Piepton sollte hier durchgängig sein.
4. Damit seid ihr auch schon bereit für die nächste **Test-Phase**! 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 sicher rückwärts einparken. ☺

Quellenverzeichnis:

Abb. 1 – *Quelle: Nozilla „Parking assist“ unter Lizenz CC BY-SA 3.0/ verändertes Original*

Abb. 2 – *Quelle: InfoSphere*

Abb. 3 – *Quelle: Screenshots der Fritzing-Software (http://fritzing.org)*

Screenshots der Programmelemente aus ArduBlock

Alle weiteren Grafiken/Icons – Quelle: InfoSphere