Station 2 – Einparkhilfe inkl. Einstieg

„Wenn's knallt, noch'n Meter.. "

Jeder von euch, der demnächst in der Fahrschule seine Runden dreht, wird auch das Rückwärtseinparken kennenlernen! Man dreht und wendet das Auto hin und her, steht letzten Endes doch schief und kann vor allen Dingen den Blick nicht von den Spiegeln lassen – 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 Infrarotlicht, das euch vielleicht von Infrarot-Fernbedienungen bekannt ist. Licht im sogenannten Infrarot-Spektrum ist für das menschliche Auge nicht sichtbar. Die Einparkhilfe benutzt dabei einen Sensor, welcher aus einem Sender und einem Empfänger besteht. Vom Sender wird das Infrarotlicht ausgesandt, dann von einer Oberfläche reflektiert und anschließend vom Empfänger registriert. Wie damit ein Abstand gemessen werden kann und wie aus dem Abstand ein Signalton erzeugt wird, das werdet ihr hier mit dem Arduino-Mikrocontroller nachstellen und damit hoffentlich verhindern, dass unser Testfahrzeug eine verbeulte Stoßstange bekommt.

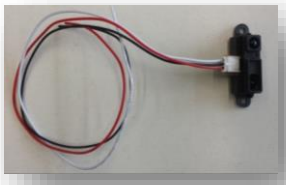


Abb. 2: IR-Distanz-Mess-Sensor

Der Einstieg

Bevor ihr euch an das eigentliche Projekt setzt, werdet ihr zunächst einige Bauteile genauer betrachten und lernen wie ihr richtig damit umgeht.

Für diesen Einstieg wird folgendes benötigt:

- Der Arduino-Mikrocontroller inkl. Steckbrett und USB-Kabel
- Der Piezo-Signalgeber
- Ein 100kΩ-Widerstand (markiert mit **braun-schwarz-gelb-gold**)
- Ein Taster
- Jede Menge Kabel

Abb. 3: Piezo-Signalgeber

Wichtig: *Taster und Widerstand findet ihr in dem Kästchen für Station 0, den Signalgeber in dem Kästchen für Station 2.*

CC BY SA

Der Arduino

Im Mittelpunkt aller heutigen Projekte steht der Arduino-Mikrocontroller, den ihr unten schematisch dargestellt seht:

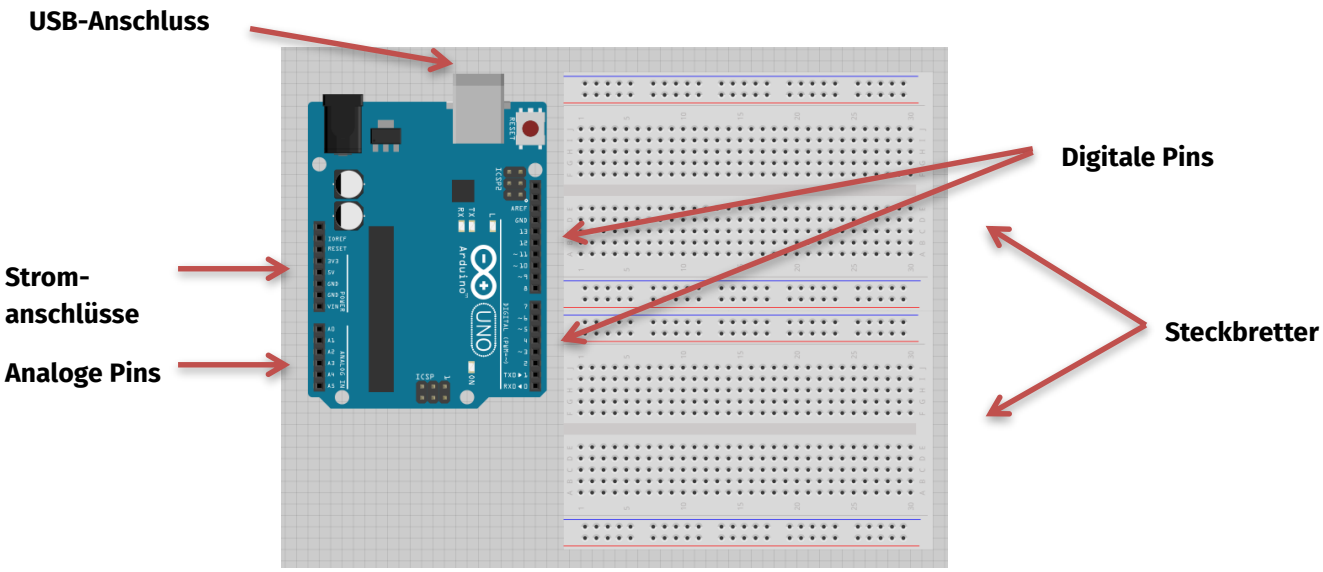


Abb. 4: Arduino-Board und Steckbrett

Zuerst eine kurze Erläuterung der wichtigsten Komponenten:

USB-Anschluss:

Damit verbindet ihr den Arduino später mit dem Computer, um euer Programm zu übertragen. Außerdem dient dieser Anschluss als Stromversorgung.

Stromanschlüsse:

Manche Bauteile müssen mit Strom versorgt werden, auch wenn es nicht explizit drauf steht, merkt euch einfach:

GND = – (Minuspol) 5V = + (Pluspol)

Digitale Pins:

Diese können entweder den Zustand AN (HIGH) oder AUS (LOW) annehmen. Digitale Anschlüsse können als Eingänge (z.B. für Schalter) oder Ausgänge (z.B. für Lampen) benutzt werden.

[**Hinweis:** *Bitte benutzt nur die Pins 2-13. Die anderen beiden haben eine Sonderfunktion.*]

Analoge Pins:

Diese werden im Gegensatz zu den digitalen Pins nur als Eingänge benutzt. Im Gegensatz zu den digitalen Pins können sie nicht nur zwei, sondern 1024 verschiedene Werte unterscheiden. Der Wertebereich geht von 0 (das entspricht 0V) bis 1023 (das entspricht 5V).

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Das Steckbrett

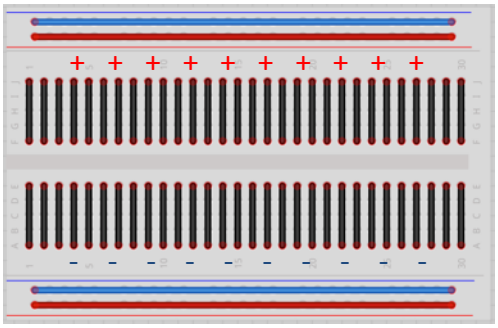


Abb. 5: Steckbrett mit gezeichneten Verbindungen

Auf der obigen Abbildung seht ihr das Innere des Steckbretts dargestellt. Hier könnt ihr sehen wie die Löcher auf der Oberseite des Steckbretts im Inneren miteinander verbunden sind.

Wichtig: *Bevor ihr etwas an der Verkabelung ändert, trennt den Arduino immer vom Strom! Zieht dafür einfach das USB-Kabel aus dem Computer.*

Der Piezo-Signalgeber

Der akustische **Piezo-Signalgeber** ist ein sogenannter „Summer“ oder „Piepser“, ein kleines Bauteil, das elektronisch angesteuert wird und einen **Ton** erzeugt – es ist also unsere akustische Ausgabe. Einsatz finden sie 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

Es „summt“

Genug geredet, jetzt wird gebaut!
Zuerst sollt ihr testen, ob der Signalgeber überhaupt funktioniert.

Eure ToDos

1. Verbindet den Minus-Pol (das **kürzere** Beinchen!) des Signalgebers **direkt über das blaue Verlängerungskabel** mit einem **GND** – Pin des Arduino.
   [**Hinweis:** *Ein Verlängerungskabel hat auf einer Seite ein Loch.*]
2. Verbindet außerdem den Plus-Pol (das **längere** Beinchen!) **über das rote Verlängerungskabel** mit dem **5V** – Anschluss.
3. Versorgt den Arduino nun mit Strom (per USB-Kabel am Computer anschließen).

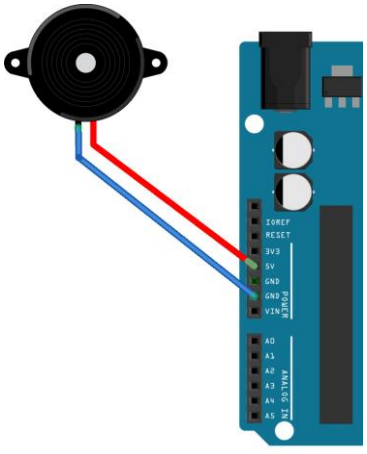


Abb. 7: Schaltung mit Piezo

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Da man einen ständig summenden Signalgeber aber nicht gebrauchen kann, sollten wir diesen nun sinnvoll programmieren! Öffnet hierzu die Arduino-Software:

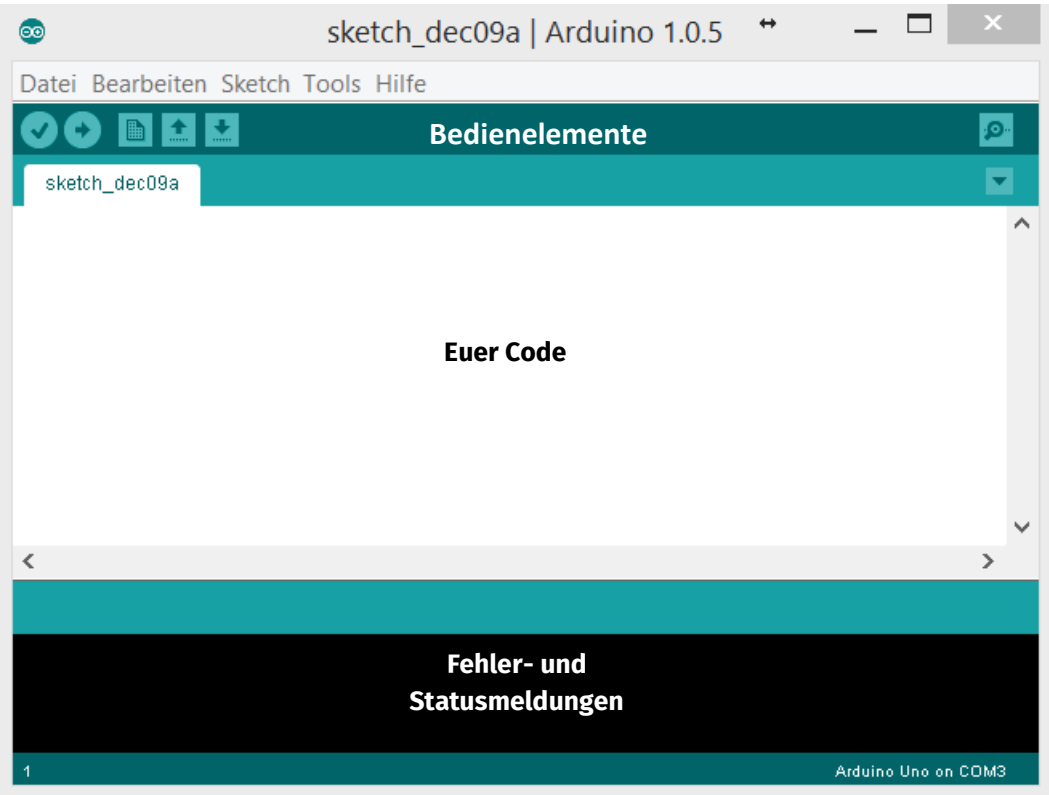


Abb. 8: Screenshot der Arduino-Software

Zunächst eine kurze Erläuterung der Bedienelemente:

Mit diesen Knöpfen überprüft ihr euer Programm. Mit dem linken Knopf prüft und kompiliert ihr euer Programm, und mit dem rechten Knopf wird das Programm sowohl kompiliert als auch an den Arduino gesendet.
[**Hinweis:** *Unter Umständen könnte dies etwas dauern! Achtet dabei auf den Ladebalken rechts unten.*]

Dieser Knopf legt einen neuen Sketch an.

Sketch, das ist der Name eines Programms im Arduino

Diese Knöpfe öffnen bzw. speichern Sketches.

CC BY SA

Programmieren

Den meisten von euch ist bestimmt die Programmiersprache Java bekannt. Der Arduino wird auf eine ähnliche Weise programmiert. Im Folgenden seht ihr das Grundgerüst, welches für jedes Programm benötigt wird:

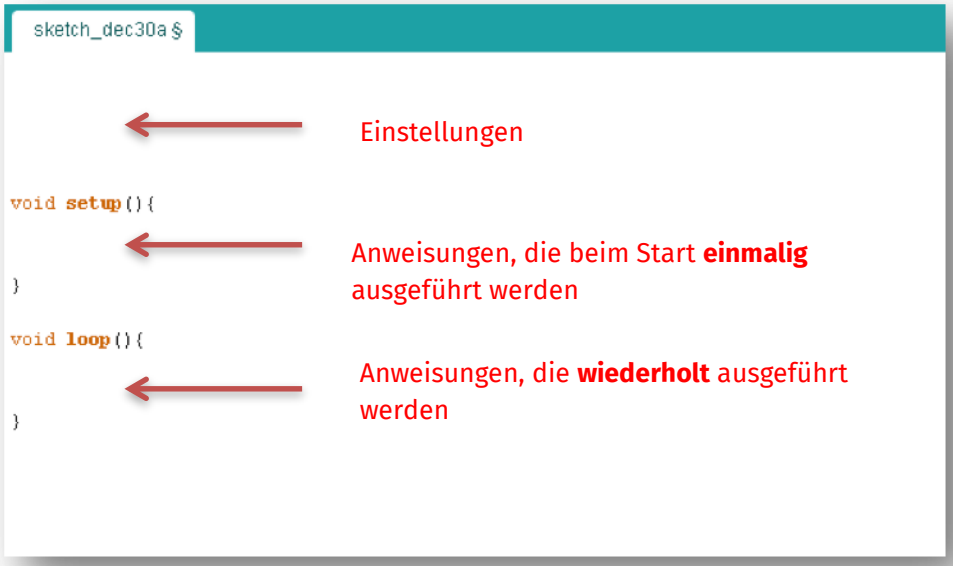


Abb. 9: Aufbau eines Arduino-Sketch

Für eine bessere Lesbarkeit könntet ihr (beispielsweise wie auch in Java) weitere Methoden und Unterprogramme schreiben, die dann in der loop() oder auch in setup() aufgerufen werden.

Die nächste Aufgabe wird sein, den Summer über ein Programm zu steuern.

Wichtig: *Da ihr nun etwas an der Verkabelung ändert, denkt daran vorher den Arduino vom Strom zu trennen.*

Um den Summer nun über ein Programm steuern zu können, verbindet man den Plus-Pol des Signalgebers nicht mit dem 5V – Anschluss, sondern mit einem **digitalen Pin** (denkt daran: nur die Pins 2-13).

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Eure ToDos

1. Verbindet den Plus-Pol des Signalgebers mit einem digitalen Pin.
2. Öffnet einen neuen Arduino-Sketch.
3. Initialisiert am Anfang eures Sketches (Einstellungen) eine int-Variable (<Pin-Name>), in der die Nummer des Pins, an dem der Summer angeschlossen ist, gespeichert wird.
4. Definiert diesen Pin in setup() als Ausgang, benutzt dafür die Methode pinMode(<Pin-Name>, <Pin-Typ>).
   [**Hinweis:** *In <Pin-Typ> wird festgelegt, ob der Pin als Eingang (INPUT) oder Ausgang (OUTPUT) verwendet wird.*]
5. Schaltet mit der Methode digitalWrite(<Pin-Name>, <Zustand>) den Strom an eurem Pin ein.
   [**Hinweis:** *In <Zustand> wird festgelegt, ob der Pin an (HIGH) oder aus ist (LOW).*]
6. Speichert euren Code ab. Kompiliert diesen und schaut was passiert.

Nun sollte der *Summer* wieder zu hören sein.

In eurem nächsten Schritt werdet ihr euer Programm so verändern, dass der Summer beim Summen kurze Pausen macht. Hierfür benötigt ihr eine weitere Methode, die euren Code für kurze Zeit pausieren lässt, nämlich:

delay(<Zeit>).

Die Zahl, die ihr dieser Methode übergeben könnt, entspricht der Zeit in Millisekunden.

Der *Summer* sollte nun also immer kurz pausieren.

Eure ToDos

1. Erweitert euren Code, indem ihr die oben beschriebene Methode delay(<Zeit>) einbaut.
2. Kompiliert und testet euer Programm.

Wichtig: *Falls es nicht geklappt hat, überlegt euch, wie oft* delay(<Zeit>) *benutzt werden muss.*

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Der Taster

Abb. 10: Taster

Wollt ihr das Summen vielleicht auch manuell ein- und ausschalten? Dafür benötigt ihr einen Taster, der im Folgenden kurz vorgestellt wird.

Ein Taster, das ist ein Bauelement, das Strom leitet, solange der Knopf runtergedrückt ist. Nimmt man den Finger weg, fließt auch kein Strom mehr. Ihr findet den Taster in eurer Box im Kästchen der Station 0.

Ein paar Kleinigkeiten sind beim Einbau eines Tasters jedoch zu beachten. Die eine Seite des Tasters kommt an den Plus-Pol des Steckbretts, die andere Seite wird mit einem der digitalen Pins des Arduino verbunden. Zusätzlich muss diese Seite aber noch mit einem großen Widerstand (**100 kΩ, braun-schwarz-gelb-gold**) mit dem Minus-Pol verbunden werden. Wenn euch interessiert warum, schaut auf dem Plakat Elektrotechnik unter Pull-down nach!

Wir wollen also den *Summer* nun ein- und ausschaltbar machen.

Eure ToDos

1. Schließt den Taster wie oben beschrieben an euer Steckbrett an.
2. Legt eine neue Variable für den digitalen Pin an.
3. Macht diesen Pin zu einem Eingang (Hinweis: INPUT).
4. Verwendet für die Abhängigkeit (ob der Pin ein- oder ausgeschaltet ist) if- Anweisungen. Um zu überprüfen, ob der Pin ein- oder ausgeschaltet ist, braucht ihr die Methode digitalRead().
   [**Hinweis:** Überprüft, ob der Rückgabewert von digitalRead() HIGH *oder* LOW *ist.*]

Wichtig: *Wie der Großteil der Arduino-Sprache sehen auch* die if-Anweisungen *wie in Java aus.*

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Baut also den Taster ein, denkt dabei an den Pull-Down-Widerstand und vergesst nicht dem zugehörigen Pin den richtigen Typ zuzuweisen

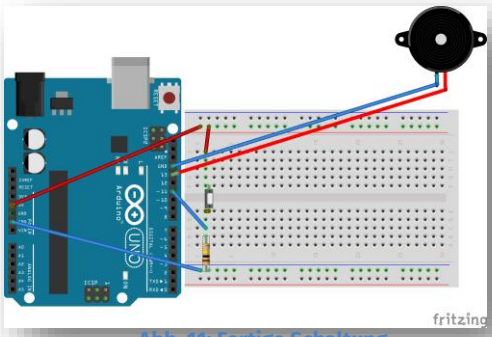


Abb. 11: Fertige Schaltung

Hervorragend, der Einstieg ist geschafft! Ihr wisst nun wie man digitale Pins auslesen und ansteuern kann. Außerdem seid ihr jetzt mit dem Piezo-Signalgeber vertraut und könnt nun eine Einparkhilfe programmieren! Eine Sache fehlt euch aber noch: ein **Infrarot-Distanz-Mess-Sensor** (IR-Distanz-Mess-Sensor), damit der Signalgeber erst dann summt, wenn man sich zu nah an einem Hindernis befindet.

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

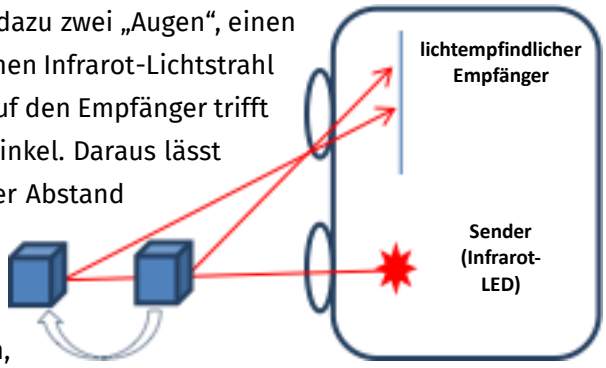
Analoge Pins

Bisher kennt ihr nur **digitale Pins**, über die z.B. der Piezo-Signalgeber angeschlossen wurde. An diesen Pins können nur **binäre Werte** ein- und ausgelesen werden – also ein/aus, d.h. 1/0, oder eben die bekannten Werte HIGH und LOW für eine hohe bzw. niedrige Spannung. Ein IR-Distanz-Mess-Sensor zum Beispiel kann aber nicht nur zwei Werte annehmen, sondern misst im **Bereich** zwischen dem minimalen und maximalen Wert 1024 **Zwischenwerte**. Das realisieren am Arduino **analoge Pins** (*A0* bis *A5*), an denen ein Bereich gemessen werden kann.

Der IR-Distanz-Mess-Sensor

Der **Infrarot-Distanz-Mess-Sensor** *„Sharp GP2Y0A21YK“* ist ein komplexes Bauteil mit integrierter Verschaltung, das **Entfernungen in einem Bereich von ca. 10-80 cm** recht genau erkennen kann.

Wie am Anfang bereits erläutert besitzt der Sensor dazu zwei „Augen“, einen **Sender** und einen **Empfänger**. Der Sender sendet einen Infrarot-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. Die Skizze gibt euch dazu einen Überblick.



Für das Zusammenbauen solltet ihr darauf achten, dass der IR-Distanz-Mess-Sensor **direkt am Arduino** angeschlossen wird, daher wird das Steckbrett auch ab sofort nicht mehr benötigt.
[**Hinweis:** *Den IR-Distanz-Mess-Sensor findet ihr im Kästchen zur Station 2.*]

Eure ToDos

1. Löst alle Kabel, den Pull-Down-Widerstand und den Taster vom Steckbrett.
2. Schließt den IR-Distanz-Mess-Sensor an den Arduino an, indem ihr das schwarze Kabel mit GND, das rote Kabel mit 5V und das weiße mit einem analogen Pin verbindet.

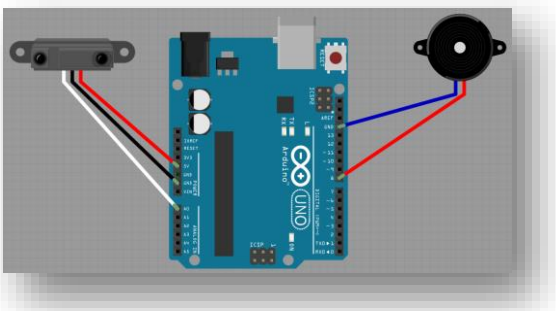


Abb. 12: Fertiger Aufbau

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Manuelle Abstandsmessung

Zuallererst werdet ihr die Daten, die der IR-Distanz-Mess-Sensor misst, über ein kleines Programm einlesen und euch auf der seriellen Konsole ausgeben lassen.
Die serielle Konsole schickt Daten über das Kabel vom Arduino zum Computer. Ihr startet diese Übertragung indem ihr in setup() die Methode

Serial.begin(9600);

aufruft. Die übergebene Zahl 9600 gibt die Geschwindigkeit an mit der die Daten über USB gesendet werden. Um euch die Daten auch ausgeben zu lassen, braucht ihr in der loop() eine weitere Methode, nämlich

Serial.println(<Eure Daten>);.

Ihr übergebt dieser Methode einfach eine Variable oder natürlich auch einen String (die Anführungszeichen dabei nicht vergessen). Mit diesem Button (oben rechts in der Bedienleiste) öffnet ihr die serielle Konsole.
Nun ist eure nächste Aufgabe, mit dem IR-Sensor Werte zu messen und sie entsprechend der Tabelle am Ende der Aufgaben verschiedenen Zentimeter-Angaben per Hand zuzuordnen. Hierfür sollt ihr feststellen können, wie diese Werte zu den **Zentimeter-Angaben** passen.
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 einfach zu bestimmende mathematische Formel zum Umrechnen.

Eure ToDos

1. Beginnt einen neuen Arduino-Sketch.
2. Legt eine neue Variable vom Typ int für den analogen Eingangspin an.
3. Legt eine weitere Variable an, in die ihr später die Sensor-Daten speichern werdet, um schnell und direkt darauf zugreifen zu können.
4. Nun zu setup(): Startet die Übertragung zur seriellen Konsole.
5. Die loop():
   a. Die Sensor-Werte sollten nun eingelesen werden. Da ihr analoge Werte auslesen möchtet, wird der Befehl
      analogRead(<Pin-Name>);
      benötigt. Dieser liefert einen int-Wert zurück, den ihr der oben angelegten Variable (aus Schritt 2) direkt zuweisen könnt.
   b. Lasst euch diese Werte auf dem seriellen Monitor ausgeben und versucht dabei in der Ausgabe auch eine Beschreibung dessen anzuzeigen, was ausgegeben wird.
   c. Damit die Werte nicht zu schnell auf dem seriellen Monitor angezeigt werden, setzt zum Schluss noch die bereits kennengelernten Methode delay(<Zeit>) hinter den Ausgabebefehl, sodass die Werte nur **jede ein oder zwei Sekunden** aktualisiert werden.
6. Es ist an der Zeit, den Sketch zu testen! Schließt den Arduino wieder an, führt das Programm aus und beobachtet die Sensor-Werte.

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

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

min = _____ **max = _____**

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

Um hierbei aber nicht nur Minimum und Maximum zu kennen, sondern eine **vernünftige Einteilung** der Abstände vornehmen zu können, ist jetzt „Maßarbeit“ gefragt. Nutzt den **Zollstock** vor euch, um herauszufinden, welche Sensor-Werte bei 10, 20 30 Zentimetern usw. ausgegeben werden, und tragt diese in die Tabelle ein!

Hinweis: *Der IR-Sensor misst am besten und genauesten, 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
Sensor-Wert								

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

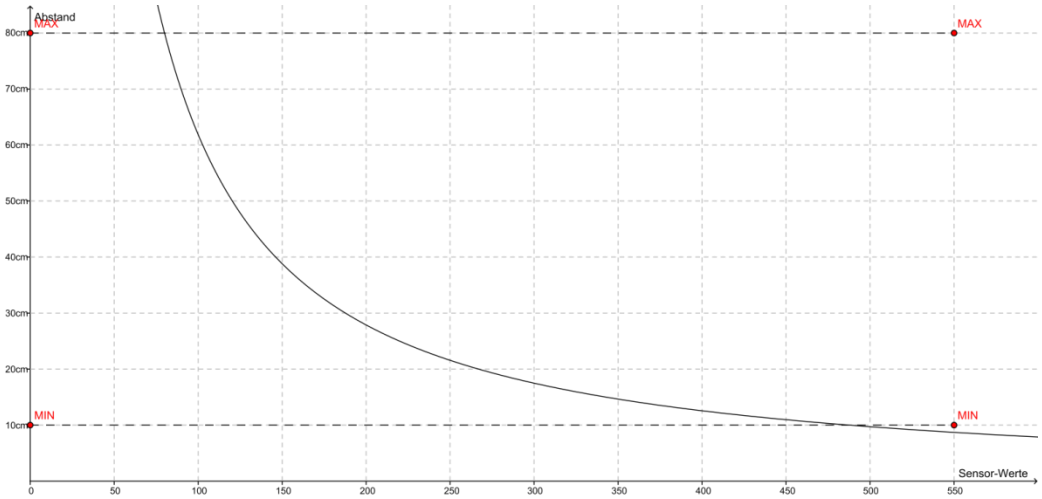


Abb. 13: Ausgabewerte des IR-Distanz-Mess-Sensors für Abstände von 10-80cm

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

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 gibt es noch nicht. Dazu benutzt ihr den bereits kennengelernten **Piezo-Signalgeber**.

Eure ToDos

1. Speichert euren bestehenden Arduino-Sketch unter einem anderen Namen ab.
2. Erstellt auch für den Piezo-Signalgeber eine int-Variable und weist ihr einen **digitalen Ausgangspin** zu.
3. setup(): Definiert den Pin des Piezo-Signalgebers als Ausgang.
4. In der loop() benötigt ihr wieder if-else-Anweisungen:
    a. Lasst den Piezo-Signalgeber piepsen, wenn der Abstand zum Infrarotsensor zu gering wird (hier **unter 40 cm**). Schaut dafür auf eure Tabelle und überlegt euch mit welchem Grenzwert ihr vergleichen müsst.
    b. Falls eure Bedingung wahr ist, der Abstand also zu gering, soll ein Dauerton erklingen. Dazu muss der **digitale Ausgangspin** auf HIGH gesetzt werden.
    c. Im anderen Fall soll kein Ton zu hören sein. Überlegt euch, welcher Befehl den Piezo-Signalgeber wieder ausschaltet und vollendet so eure if-else-Anweisung.
5. **Fertig!** Übertragt den Sketch auf den Arduino und testet ihn.

Und jetzt?!

Habt ihr weitere Ideen und Verbesserungsvorschläge? Diskutiert diese in der Gruppe! In den nächsten Teilen werdet ihr eure Einparkhilfe immer weiter optimieren.

„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!

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

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

Habt ihr jetzt schon ein paar Vorschläge, wie ihr das umsetzen könnt?

Eure ToDos

1. Zusammengebaut ist schon alles, also geht es damit los, indem ihr euren Sketch unter einem neuen Namen abspeichert!
2. Eure einzige „Baustelle" dieses Mal ist loop(), genauer gesagt der Teil mit der if-else-Anweisung:
   a. Seht euch nochmal eure Tabelle auf Seite 10 an und überlegt euch eine passende **Einteilung der Sensor-Werte und Zentimeter** in einen Bereich, in dem kein Ton erklingt, und zwei Bereiche mit unterschiedlichen Pieps-Geschwindigkeiten.
   [**Hinweis:** *Ihr könnt die Tabelle nach diesem ToDo nutzen.*]
   b. Nun müsst ihr nur noch eure if-else-Anweisung aus dem ersten Teil ein wenig anpassen und die eben bestimmten Bereiche einbauen. Dazu werden aus der einen Anweisung drei einzelne if-Anweisungen.
   [**Hinweis:** *Nutzt die mathematischen Vergleiche und das logische* ***&& [und]*** *sinnvoll und verändert die Geschwindigkeit des Piepsen über* delay() *mit unterschiedlichen Werten!*]
3. Damit seid ihr auch schon bereit für die letzte **Test-Phase**! Überspielt euren Sketch und überprüft eure verbesserte Einparkhilfe. Passt ggf. die gewählten Bereiche oder die Verzögerungen so an, dass ihr mit dem Ergebnis zufrieden seid.

	Bereich 1 - keine Gefahr -	**Bereich 2 - nährt sich Hindernis -**	**Bereich 3 - Gefahr -**
Zentimeter	>_____	_____-_____	<_____
Sensor-Werte	<_____	_____-_____	>_____

Geschafft?!?

Herzlichen Glückwunsch! Ihr habt es geschafft! Dank eurer Hilfe kann das Testfahrzeug jetzt ganz entspannt einparken ☺

Wenn ihr wollt, gibt es noch eine kleine Herausforderung für euch...

Fragt einfach nach dem Bonus-Blatt!

CC BY SA

Station 2 – Einparkhilfe inkl. Einstieg

Es lebe das Chaos... oder auch nicht!

Damit auch die nächste Gruppe alle Bauteile wiederfindet, würden wir uns freuen, wenn ihr so nett wärt, die Bauteile in die zugehörigen Kästchen einzusortieren.

Im Kästchen zu **Station 0** gehören:

- der 100kΩ-Widerstand,
- der Taster.

Im Kästchen zu **Station 2** gehören:

- der Piezo-Signalgeber,
- der IR-Distanz-Mess-Sensor.

Vielen Dank!!! ☺

Quellenverzeichnis:

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

Abb. 2, 3, 6, 10, 13 - *Quelle: InfoSphere*

Abb. 4, 5, 7, 11, 12 - *Quelle: Screenshots der Fritzing-Software (http://fritzing.org)*

Abb. 8, 9 - *Quelle: Screenshots der Arduino-Software (http://www.arduino.cc)*

Alle weiteren Grafiken/Icons - Quelle: InfoSphere

CC BY SA