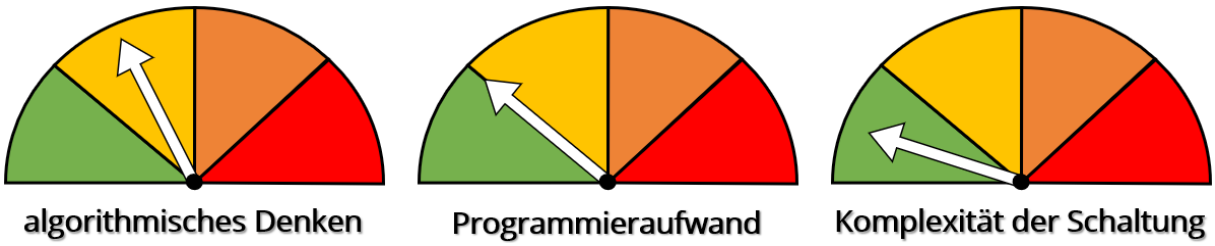
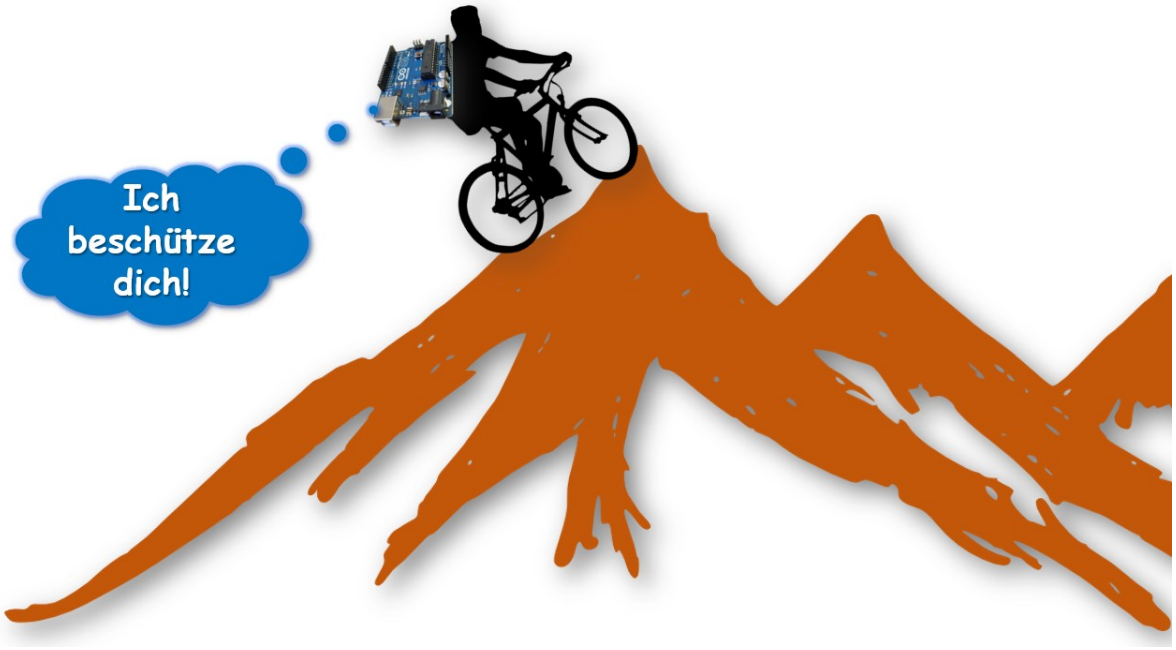
TECHNISCHE UNIVERSITÄT DRESDEN

Arduino Uno

Station 2 | Auf die schiefe Bahn geraten!





DDI Professur für Didaktik der Informatik

SYLBER Synergetische Lehrerbildung im exzellenten Rahmen

Das Maßnahmenpaket „TUD-Sylber² – Synergetische Lehrerbildung im exzellenten Rahmen“ wird im Rahmen der gemeinsamen „Qualitätsoffensive Lehrerbildung“ von Bund und Ländern aus Mitteln des Bundesministeriums für Bildung und Forschung gefördert.

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

DIE DIGITAL GESTÜTZTE FORTBEWEGUNG

Die Fortbewegung wird in jeglicher Form immer digitaler und vernetzter. Egal ob es sich hierbei um hochmoderne Autos handelt, welche mit vielen Sensoren ausgestattet sind und teils sogar ganz ohne menschliches Eingreifen fahren. Aber auch Fahrräder, Elektroscooter, Hoverboards und viele mehr sind mit viel Technik ausgestattet. Ziel dieser Digitalisierung ist es den Menschen ein bequemeres Leben zu bieten und die Sicherheit bestmöglich zu erhöhen, um Unfälle zu vermeiden.

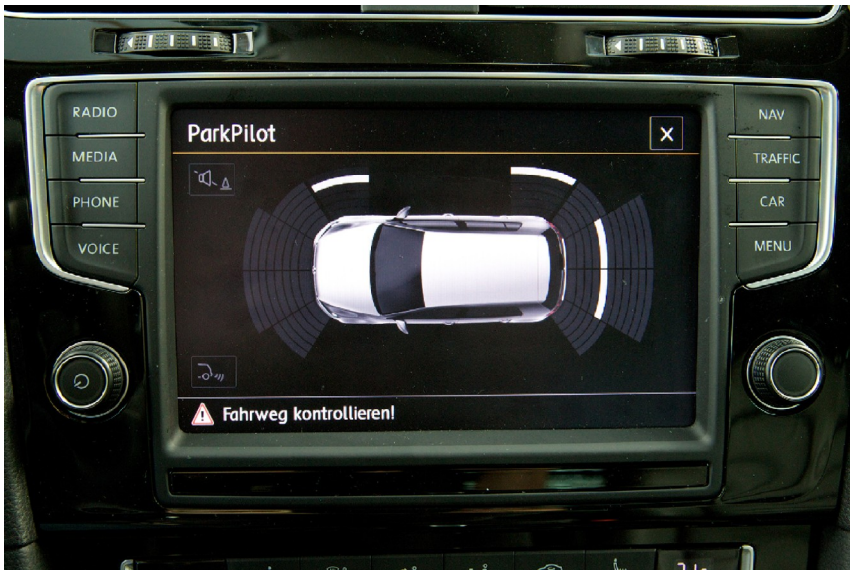


Abbildung 1: Darstellung eines Parkassistenten

AUF DIE SCHIEFE BAHN GERATEN

Stell dir vor du fährst mit deinem Fahrrad einen ziemlich steilen Berg hinauf. Ab einem Winkel von `45°` besteht die Gefahr, dass du nach hinten mit dem Fahrrad umfällst. Das gilt es zu verhindern und du musst dafür das notwendige Projekt umsetzen! Um dies zu realisieren nutzt du einen sogenannten Kippsensor und zur Ausgabe der Warnung einen Lautsprecher, sowie zwei LEDs (grün und rot). Zunächst lernst du den Kippsensor genauer kennen und wirfst einen Blick in das Innere des Sensors. Weiterhin wirst du eine einfache Verzweigung nutzen um festzulegen, wann welche Ausgabe für den Fahrer des Fahrrads notwendig ist.

Abbildung 2: Fahrbahnneigung

MUSIK IN DEINEN OHREN

Als erstes schauen wir uns an, wie wir den Lautsprecher verwenden. Ziel ist es eine kleine Melodie auszugeben, dabei kannst du bei der dazugehörigen Aufgabe zwischen einem Martinhorn von Feuerwehr und Rettungsdienst, oder der Melodie von „Alle meine Entchen“ wählen.

Wie der sogenannte Piezo-Lautsprecher aussieht, kannst du in der Abbildung erkennen, auf der Oberseite ist ebenso ein Pluszeichen eingeprägt, welches anzeigt, dass der darunterliegende Pin, in unserem Fall, mit einem digitalen Pin verbunden wird. Der andere Pin wird selbstverständlich mit `GND` verbunden.

Um den Lautsprecher zu verwenden, benötigst du zwei Funktionen:

```
tone(PIN, Frequenz / Tonhöhe);
```
→ startet den Ton mit der eingegeben Frequenz

```
noTone(PIN);
```
→ beendet die Tonausgabe

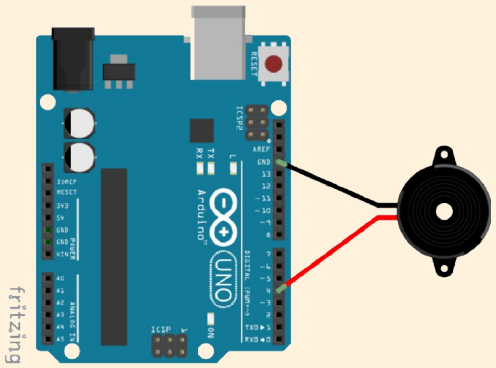
Bei der Tonhöhe kannst du beliebige Frequenzen eingeben, wir wollen uns aber nachfolgend nur auf die in der unteren Tabelle dargestellten Noten beziehen. Diese bildet die Noten mit den dazugehörigen Frequenzen ab.

Note	Frequenz [Hz
c‘	262
cis‘/des‘	277
d‘	294
dis‘/es‘	311
e‘	330
f‘	349
fis‘/ges‘	370
g‘	392

Note	Frequenz [Hz
gis‘/as‘	415
a‘	440
ais‘/b‘	466
h‘	494
c“	523
cis“/des“	554
d“	587
dis“/es“	622

AUFGABE 1 – VERBINDEN DES PIEZO-LAUTSPRECHERS

Zunächst ist es deine Aufgabe den Piezo-Lautsprecher mit dem Arduino zu verbinden. Achte hierbei wie oben bereits skizziert auch auf das eingeprägte Pluszeichen, damit du den Lautsprecher korrekt gepolt verbindest. Realisiere die Schaltung so, wie sie auf der Abbildung rechts dargestellt ist.



Da die Verbindung nun steht können wir nun Töne ausgeben. Die Funktionen die dazu benötigst sind dir alle bekannt. Die einzige die du noch benötigst ist die dir bereits bekannte Wartefunktion `delay(Zeit in Millisekunden)`. Warum ist das notwendig? - Ohne Wartezeit würde das selbige passieren, wie bei unserer blinkenden LED. Das heißt der Ton wird versucht abzuspielen, aber die schnelle Abarbeitungsfrequenz des Prozessors sorgt dafür, dass direkt als nächstes die Funktion zum beenden der Tonausgabe ausgeführt wird.

Von den nächsten zwei Aufgaben musst du nur eine auswählen – je nachdem welche du am liebsten bearbeiten möchtest. Die grundlegende Programmstruktur sollte dabei wie folgt aussehen. Zuerst gibst du Ton aus, wartest eine angemessene und vor allem variable Zeit, dann stoppst du die Tonausgabe und wartest wieder bevor der neue Ton ausgegeben wird.

AUFGABE 2A – MARTINHORN

Das Martinhorn als Folgetonhorn als akustische Warneinrichtung bei Fahrzeugen von Behörden und Organisationen mit Ordnungs- und Sicherheitsaufgaben.

Die Töne müssen im Bereich 360 Hz bis 630 Hz liegen. Die am häufigsten verbaute Variante in Deutschland setzt sich dabei aus den Tönen *dis‘* und *ais‘* zusammen.

AUFGABE 2B - „ALLE MEINE ENTCHEN

Die Melodie von „Alle meine Entchen“ hat folgende Noten:

c‘ d‘ e‘ f‘ g‘ g‘ a‘ a‘ a‘ a‘ g‘ a‘ a‘
a‘ a‘ g‘ f‘ f‘ f‘ f‘ e‘ e‘ g‘ g‘ g‘ g‘ c‘

Für die musikalisch Interessierten ist nachfolgend auch nochmal die vollständige Tonleiter abgebildet.

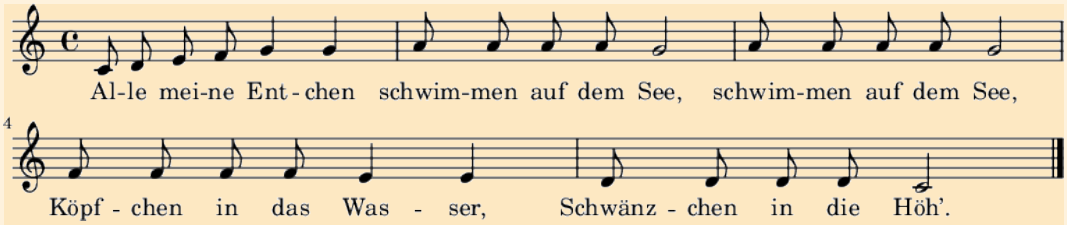
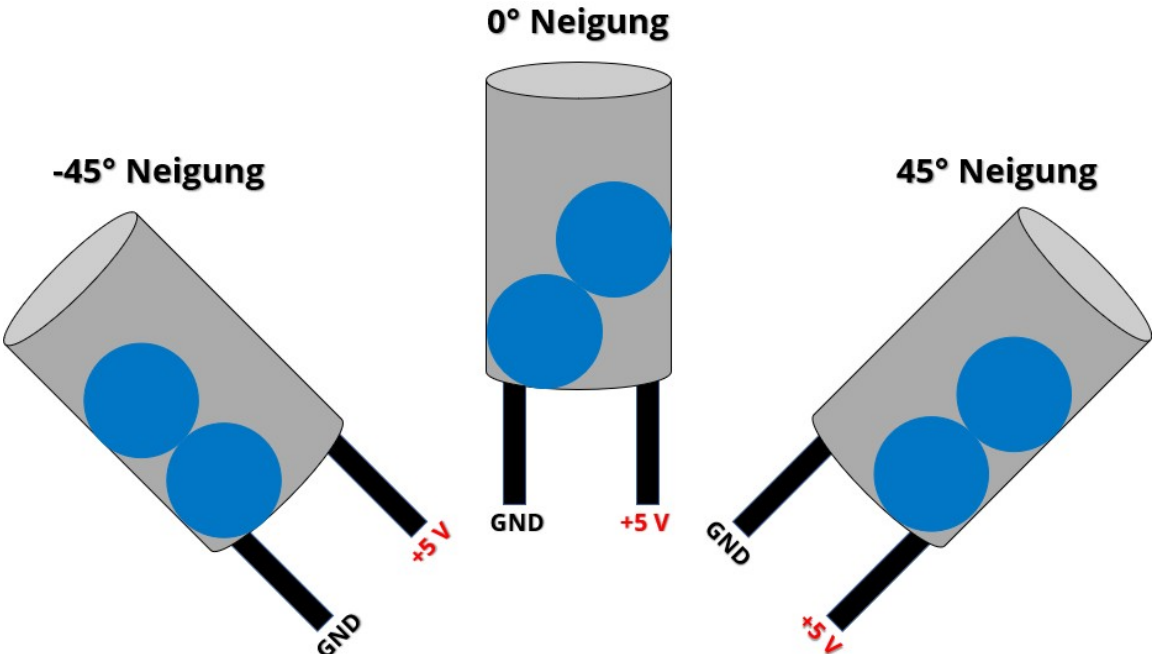


Abbildung 3: Noten für "Alle meine Entchen"

AUFBAU DES KIPPSENSOR



Wie auf der Abbildung deutlich zu sehen ist, besteht der Kippsensor im Inneren aus zwei frei beweglichen Kugeln. Je nach Position des Sensors liegen die Kugeln alle an einer Seite oder sie ordnen sich so an, dass beide Seiten der Hülle berührt werden. Wie an den Pins erkennbar ist, wollen wir einen Stromfluss durch den Sensor realisieren, durch anlegen von `+5V` und `GND`. Wir wissen, dass ein Stromfluss nur zustande kommt, wenn es sich um einen geschlossenen Stromkreis handelt.

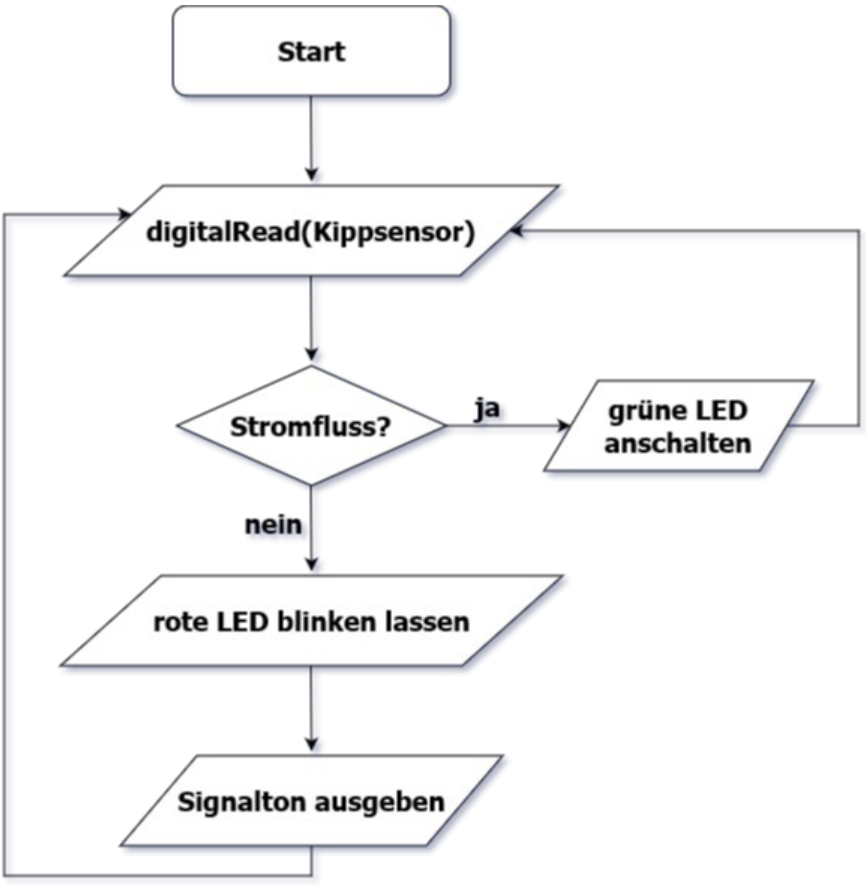
AUFGABE 3 – WANN FLIESST STROM

Trage auf den freien Felder ein Kreuz ✘ oder einen Haken ✔ ein, um festzulegen wie sich der Stromfluss verhält.

Neigung	Stromfluss?
-45° Neigung	☐
0° Neigung	☐
45° Neigung	☐

WAS PASSIERT WANN?

Bereits beim Auslesen des Tasters in der nullten Station haben wir eine Funktion genutzt, welche erkennt ob am digitalen Pin `+5V` oder `GND` anliegen. Diese lautete `digitalRead(PIN)` und gab jeweils `HIGH` oder `LOW` zurück. Den Programmablauf kann man auch mit Hilfe sogenannter Programmablaufdiagramme darstellen, wie dies unterhalb gezeigt ist.



AUFGABE 4 – SCHALTUNG REALISIEREN

Bevor wir uns mit dem Code auseinandersetzen, realisieren wir zunächst die Schaltung. Dazu benötigen wir zwei LEDs (rot, grün), den Kippsensor und einen Piezo-Lautsprecher (bereits verbaut) . Deine Schaltung sollte dann so aussehen, wie dies die untenstehende Grafik zeigt. Achte darauf, den Kippsensor genauso zu verbinden wie du es bereits beim Taster gemacht hast.

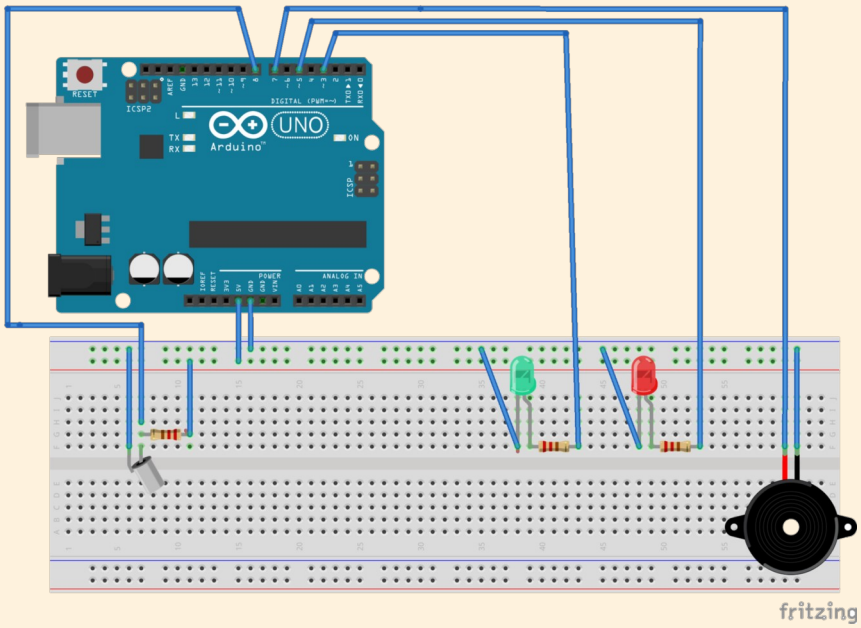


Abbildung 4: Neigungssensor

Wir müssen es also nun im Programmcode hinbekommen, irgendwie eine Entscheidung zu fällen, ob ein Winkel von `45°` erreicht beziehungsweise überschritten ist. Dazu nutzen wir die sogenannte einfache Verzweigung. Vereinfacht ausgedrückt kann man diese Struktur wie folgt erklären:

WENN Stromfluss bzw. `digitalRead(Pin)` gleich `HIGH` ist

DANN _____ ; _____ ;

SONST _____ ; _____ ;

Trage auf den Strichen ein, wann ein Tonfolge hörbar sein soll und wann welche LED (grün oder rot) aufleuchten soll.

Im Programmcode wird dies dann wie rechtsstehend umgesetzt. Innerhalb der Klammer überprüfen wir unsere Bedingung, sprich ob an unserem Pin (das mittlere Kabel unserer Kippsensorschaltung) `HIGH` oder `LOW` anliegt. Dazu nutzen wir die Funktion `digitalRead(PIN)`.
Wir wissen, dass bei einem geschlossen Stromkreis an diesem Pin `HIGH` anliegt, wodurch wir nun die Bedingung mit dem Vergleichsoperator „==“ formulieren können:

```
if(BEDINGUNG){
      LED ________
} else {
      LED ________
}
```

```
if(digitalRead(PIN) == HIGH) { … } else { … }
```

AUFGABE 5 – AKUSTISCHE UND OPTISCHE WARNUNG

Zunächst startest du wie gewohnt damit, für jedes Bauteil den dementsprechenden Pin in einer sinngebenden Variable zu speichern.

Danach legst du die Pins für die LED als `OUTPUT` und den Pin für den Kippsensor als `INPUT` fest. Nutze dazu die Funktion `pinMode(PIN, OUTPUT oder INPUT)`.

Füge dann die einfache Verzweigung hinzu und steuere damit das Verhalten vom Lautsprecher und der LED in Abhängigkeit davon, ob die Bedingung für den Stromfluss erfüllt oder eben nicht erfüllt ist.

Grafik auf dem Deckblatt: *A Arduino Uno board, JotaCartas, CC BY-SA 2.0, https://creativecommons.org/licenses/by/2.0/deed.en, https://commons.wikimedia.org/wiki/File:Arduino-uno-perspective-transparent.png*

Abbildung 1: *Grafik Park assistent, Nozilla, CC BY-SA 3.0, https://creativecommons.org/licenses/by-sa/3.0/, https://de.wikipedia.org/wiki/Einparkhilfe#/media/Datei:Parking_Assist.jpg*

Abbildung 2: *Traffic sign Slope Gefälle 20 per cent (Germany), LepoRello, CC BY-SA 3.0, https://creativecommons.org/licenses/by-sa/3.0/deed.de, https://de.m.wikipedia.org/wiki/Datei:Traffic_sign_Slope_Gef%C3%A4lle_20_per_cent_(Germany).JPG*

Abbildung 3: *Melodie, RobTorgel, CC BY-SA 3.0, https://creativecommons.org/-licenses/by-sa/3.0/deed.de,https://de.wikipedia.org/wiki/Alle_meine_Entchen*

Abbildung 4: *Neigungssensor, Ralf Snieders, CC BY-SA 3.0, https://creativecommons.org/licenses/by-sa/3.0/deed.de, https://funduino.de/nr-09-arduino-neigungssensor*

Screenshots: *fritzing electronics made by easy und Arduino IDE 1.8.12 (windows)*

Alle weiteren Grafiken: *Patrick Binkert, EduInf@TUD*