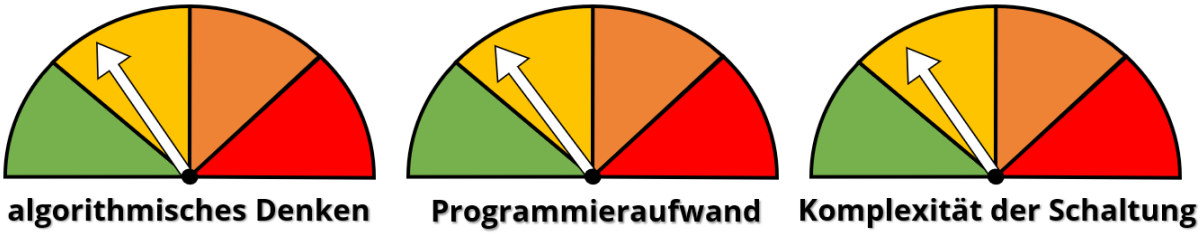
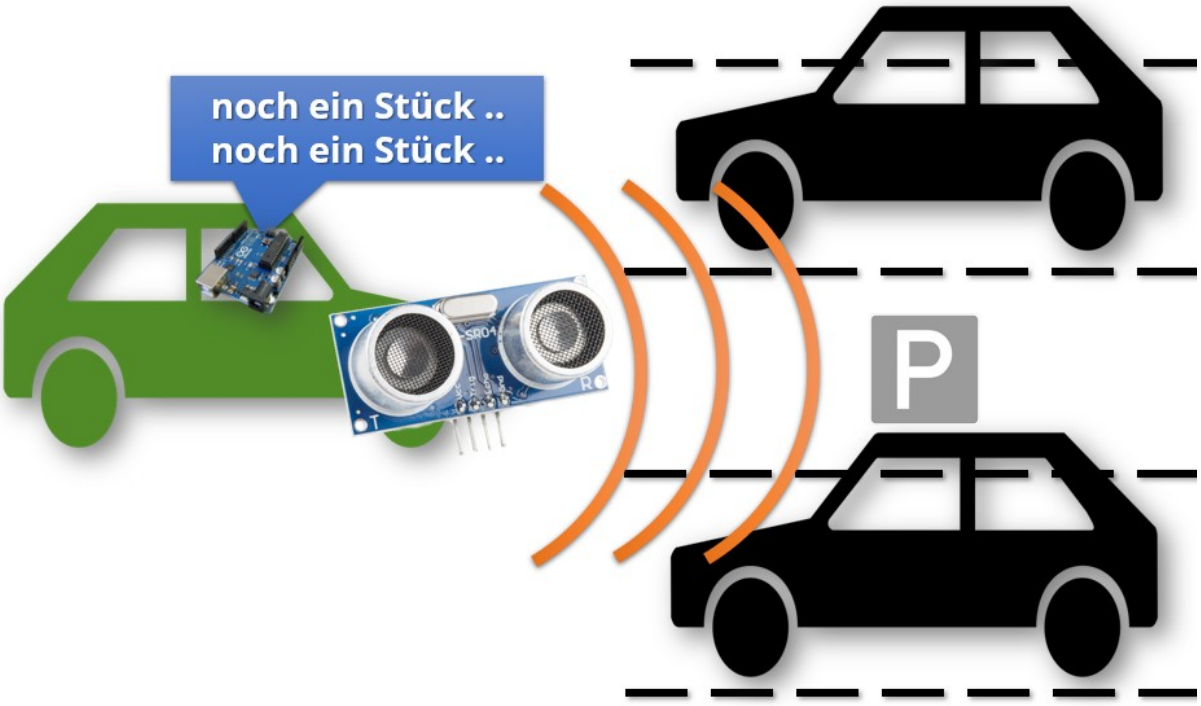
TECHNISCHE UNIVERSITÄT DRESDEN

Arduino Uno

Station 7 - ZUSATZ| smartCar – smartParking





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

ZUSATZAUFGABE

Egal ob du mit dem Ultraschall- oder Infrarotsensor gearbeitet hast, die Zusatzaufgabe ist wieder für alle gleich. Wir wollen unseren bereits funktionierenden Parksensor noch alltagstauglich modifizieren. Was bedeutet das? Im Straßenverkehr kann man natürlich nicht die ganze Zeit auf einen Bildschirm schauen, daher wollen wir optische und akustische Elemente ergänzen. Dies kommt dann bereits sehr nah an in Fahrzeuge verbaute Rückfahrsensoren heran.

RGB-LED UND PIEZO-LAUTSPRECHER

Zu diesem Zweck schauen wir uns nochmal kurz den Lautsprecher und die RGB-LED an. **Der Piezo-Lautsprecher** wird mit `GND` und einem digitalem Pin verbunden. Zur Hilfe ist auf dem Lautsprecher ein Pluszeichen aufgedruckt, dort wird das Kabel zum digitalen Pin angesteckt.

Welche Befehle benötigen wir zur Steuerung des Lautsprecher? Zum Starten des Lautsprechers, mit einer ausgewählten Tonfrequenz brauchst du:

```
tone(PIN, TONHÖHE);
```

als Befehl. Soll der Ton wieder ausgeschaltet werden kannst du dies mit:

```
noTone(PIN);
```

realisieren. **Die RGB-LED** ist in der Lage verschieden gemischte Farben darzustellen, pro Farbe können wir Werte zwischen 0 und 255 einstellen. Die Grundfarben sind dabei rot, grün und blau. Wichtig ist dabei zu beachten, dass die jeweiligen Kabel an die mit einer Tilde (~) gekennzeichneten Pins kommen und das diesmal das längere Beinchen mit `GND` verbunden werden muss. Damit wir nicht immer alle drei Farbwerte einstellen müssen nutzen wir eine **Funktion**:

```
void rgbfarben(int rot, int gruen, int blau){
     analogWrite(LEDrot, rot);
     analogWrite(LEDgruen, gruen);
     analogWrite(LEDblau, blau);
}
```

Um die Farbwerte zu setzen müssen wir dann im Programm selbst die Funktion aufrufen. Dies kannst du beispielhaft für die Farbe rot tun mit:

```
rgbfarben(255, 0, 0)
```

AUFGABE 1 – RGB-LED, LAUTSPRECHER UND FUNKTION

Bevor wir mit der Programmierung starten und unseren Parksensor mit Leben erfüllen, müssen wir erst einmal einiges an Vorarbeit leisten. Beginnen solltest du mit dem Verbinden des **Piezolautsprechers**. Der mit dem Plus gekennzeichnete Pin geht zu einem digitalen Pin deiner Wahl, der andere wird mit `GND` verbunden. Initialisiere die dazugehörige Variable, schreibe sie zur Kontrolle auf den Strich:

Piezo-Lautsprecher: _____

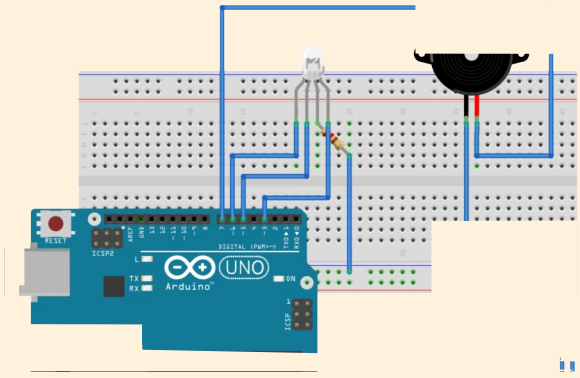
Im Anschluss verbindest du die **RGB-LED**. Orientiere dich dabei an der untenstehenden Abbildung. Initialisiere auch hierfür die notwendigen Variablen für die jeweiligen Farbe. Mache dir zuvor Notizen.

rote Farbe: _____

grüne Farbe: _____

blaue Farbe: _____

Lege die jeweiligen Pins für die LED mit Hilfe der Funktion `pinMode(PIN, OUTPUT)` als Output fest. Dies geschieht innerhalb der `setup()`-Methode. Als neue Funktion, dass heißt weder in der `setup()`–, noch in der `loop()`-Methode gilt es nun noch die Funktion für die Farbdarstellung einzufügen.



Natürlich gehört hier noch der jeweilige Sensor dazu, da dieser aber unterschiedlich sein kann, wurde dieser weggelassen.

FUNKTIONALITÄT HERSTELLEN

Die Verzweigungen existieren ja bereits in Abhängigkeit von der Entfernung. Jetzt wollen wir, dass sich die Farbe, je näher wir dem Hindernis kommen, von grün zu rot ändert. Weiterhin soll ein Tonsignal ertönen, was ebenfalls mit Annäherung zum Hindernis die Abstände zwischen den Tönen verkürzt.

Wie werden wir vorgehen? Nach den Verzweigungen geben wir die Tonfolge aus, lassen die LED leuchten und schreiben im seriellen Monitor die aktuelle Entfernung. Den jeweiligen Befehlen geben wir allerdings keine absoluten Werte, sondern Variablen. Diese werden vorab in den jeweiligen Verzweigungen mit Werten belegt.

Nehmen wir, als Beispiel zur Erklärung, eine Entfernung zwischen 20 und 60 Zentimeter.

```
…
else if(wert <=60 && wert >20){
     pause = 100; // Pause zwischen den einzelnen Tönen
     frequenz = 600; // Frequenz des Tons
     rgbfarben(100,255,0); // Farbe der RGB-LED (gelb)
}
…
```

Wir schreiben in jede Verzweigung Werte für die Pausen, die Tonhöhe und setzen die Farbe der RGB-LED. Nach allen Verzweigungen geben wir dann den Ton mit Hilfe der Variablen aus.

```
tone(PIN,frequenz);
delay(pause);
noTone(PIN);
```

Zum Abschluss schalten wir noch die RGB-LED aus. Warum überhaupt? Durch das Ausschalten erreichen wir ein Blinken und auf bewegte, beziehungsweise veränderliche, Elemente reagiert das menschliche Auge deutlich besser. Weiterhin folgt noch die Ausgabe der Entfernung zum Hindernis, zur Kontrolle.

```
rgbfarben(0,0,0);
Serial.println(entfernung);
```

Was natürlich noch fehlt ist die Initialisierung der hier verwendeten Variablen, dabei handelt es sich immer um Ganzzahlen, also um den Datentyp `integer`.

```
int pause = 0;
int frequenz = 0;
```

AUFGABE 2 – OPTISCHES UND AKUSTISCHES FEEDBACK

Beginne zunächst damit die Variablen für den Pausen- und Frequenzwert zu initialisieren. Ergänze danach den notwendigen Code zur Steuerung des Piezo-Lautsprechers und zum Ausschalten der LED.

Im Anschluss kannst du in deinen Verzweigungen mit den verschiedensten Werten experimentieren. Variiere dabei LED-Farbe, Pausenzeit und Tonhöhe. Testen kannst du dies durch Variation des Abstandes zu einem Hindernis.

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*

und

Sinnbild Personenkraftwagen, Krumpi, CC 0, https://creativecommons.org/publicdomain/zero/1/deed.de, https://commons.wikimedia.org/wiki/File:Sinnbild_PKW.svg

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

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