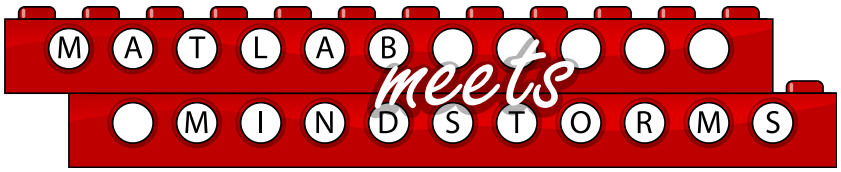
Projekt der Elektrotechnik und Informationstechnik



3. Projektversuch

– Schallsensor –

24. August 2017

Inhaltsverzeichnis

1 Einleitung

Mit dem im LEGO Mindstorms EV3 Education Set enthaltenen Schallsensor ist es möglich, Geräusche zu detektieren und gegebenenfalls darauf zu reagieren. In diesem Versuch sollen Sie den Sensor näher kennenlernen.

Ein wichtiges Hilfsmittel bei der Programmierung von Robotern sind außerdem Schleifen, da die selben Arbeitsschritte oft immer wieder ausgeführt müssen, bis ein Ereignis eintritt. Ziel des Versuches ist daher auch, die Programmierung der `for`-Schleife von MATLAB zu vertiefen.

2 Grundlagen

Im Folgenden wird der Schallsensor und die Funktionen beschrieben, mit denen der Sensor aus MATLAB angesteuert werden kann.

2.1 Mindstorms NXT Schallsensor

Abbildung 1: Schallsensor (© LEGO)

Der Schallsensor im Mindstorms-Kit (siehe Abbildung 1) misst den Schalldruckpegel in seiner Umgebung. Im Gegensatz zu einem einfachen Mikrofon ist der Sensor nicht dazu geeignet, Geräusche oder Sprache aufzunehmen, die später wieder abgespielt werden können. Der Messwert, den der Sensor an den EV3-Brick liefert, ist eher zu vergleichen mit der mittleren „Lautstärke“, die zum Messzeitpunkt am Sensor vorliegt. Es wird also im Sensor bereits ein Teil der Daten durch eine elektrische Schaltung vorverarbeitet. Dadurch geht ein großer Teil der Information darüber, um welches Geräusch es sich tatsächlich gehandelt hat, verloren.

Der Schallsensor hat zwei Betriebsmodi. Im ersten misst der Sensor den Schalldruckpegel in dB. Die dB-Skala ist logarithmisch aufgebaut, d.h. verdoppelt sich der Schalldruck, so erhöht sich der Schalldruckpegel um 6 dB. Der Messbereich des Schallsensors reicht bis 90 dB und umfasst einen Frequenzbereich von 20 Hz bis 18 kHz.

Die Empfindlichkeit des menschlichen Gehörs für Töne unterschiedlicher Frequenzen ist unterschiedlich groß. Töne im Bereich um 3 kHz werden z.B. besonders gut wahrgenommen. Der zweite Betriebsmodus des Sensors ist gedacht, um diesen Effekt zu kompensieren. Er gibt die Messwerte in dBA zurück, einer Variante der logarithmischen Skala, die in der Technischen Akustik eingesetzt wird, beispielsweise bei der Beurteilung von Lärmbelästigungen.

Um Werte des Sensors auslesen zu können, muss natürlich zuerst eine Bluetooth- oder USB-Verbindung mit dem EV3-Brick aufgebaut werden, wie Sie es auch schon aus dem vorangegangenen Versuch kennen. Die dafür benötigten Funktionen sind `ev3_obj=EV3()` `ev3_obj.connect()`, `ev3_obj.disconnect()`.

`ev3_obj.sensorX` Sensorobjekt.

`sensorX` Anschluss, an dem der Sensor angeschlossen ist. Mögliche Werte: `sensor1`, `sensor2`, `sensor3`, `sensor4`

```
SOUNDVALUE = ev3_obj.sensorX.value;
```

`SOUNDVALUE` Ausgelesener Wert des Soundsensors in dB oder dBA

```
ev3_obj.sensorX.mode = MODEVALUE;
```

`MODEVALUE` Modus des Sensors. Mögliche Werte: `DeviceMode.NXTSound.DB`, `DeviceMode.NXTSound.DBA`

2.2 Ampel

Die vorgegebenen Skripte enthalten einen Ampelplot mit 3 Lampen. Jede der Lampen kann einzeln an oder ausgeschaltet werden.

`switchLamp` Lampe ein-/ausschalten.

```
switchLamp(LAMP,'VALUE')
```

`LAMP` Lampe die angesteuert werden soll. Mögliche Werte: `1`, `2`, `3`

`VALUE` Entweder `'on'` oder `'off'`.

2.3 MATLAB-Grundlagen

Die wichtigsten Programmstrukturen, die wir in diesem Versuch benötigen, sind die `for`-Schleife und die Funktionsdeklaration.

2.3.1 `for`-Schleife

Eine `for`-Schleife in MATLAB hat beispielsweise folgende Form:

```
for x = 1:10
  x
end
```

Der Schleifenkopf (erste Zeile) bestimmt dabei die Start- und Stoppbedingungen der Schleife. In diesem Beispiel ist `x` die Schleifenvariable. `1:10` ist in MATLAB-Syntax ein Array mit 10 Elementen, nämlich genau den Zahlen 1 bis 10. Wenn die Schleife ausgeführt wird, erhält `x` je genau einmal hintereinander den Wert eines der Elemente.

Der Schleifenkörper (der Teil zwischen dem Kopf und der letzten Zeile) wird dann für jeden Wert einmal ausgeführt. D.h. im ersten Durchlauf hat `x` den Wert 1, dann 2, usw. bis 10. Danach ist die Abarbeitung der Schleife beendet und das Programm fährt mit

den Befehlen nach der Schleife fort. Da der Schleifenkörper auch aus mehreren Befehlen bestehen kann, wird das Ende der Schleife mit `end` gekennzeichnet.

Um das Programm für den Mensch, der es programmiert, besser lesbar zu machen, rückt man den Schleifenkörper um einen Tabulator nach rechts ein. Dies ist für die korrekte Abarbeitung der Schleife jedoch nicht erforderlich.

In diesem Beispiel ist der Befehl `x` der Schleifenkörper. Es wird also in jedem Durchlauf der Wert der Variable `x` ausgegeben. Da dieser sich mit jedem Durchlauf ändert, gibt das Beispiel nacheinander die Werte 1 bis 10 aus.

2.3.2 MATLAB-Funktionen

Wenn man in MATLAB eine eigene Funktion deklarieren möchte, muss man dies stets in einer eigenen Datei tun. Die Datei sollte dabei genau so heißen wie die Funktion selbst. Zusätzlich muss im Kopf der Datei spezifiziert werden, welche Argumente die Funktion entgegennimmt und welche Rückgabewerte sie hat:

```
function [a, b, c] = meinefunktion(d, e, f);

<Befehle> ...
```

Hierbei sind `d` bis `f` Argumente, die der Aufrufer an die Funktion übergibt. `a` bis `c` sind Rückgabewerte der Funktion, die dem Aufrufer übergeben werden, wenn die Funktion abgearbeitet ist. In diesem Beispiel könnte der Aufrufer den Befehl

```
[x, y, z] = meinefunktion(1, 2, 3);
```

geben. Dadurch wird `meinefunktion` mit den Werten `d = 1`, `e = 2` und `f = 3` abgearbeitet. Das Ergebnis der Berechnung wird funktionsintern zwar in den Variablen `a` bis `c` gespeichert, jedoch bestimmt der Aufrufer der Funktion, dass die Ergebnisse in den Variablen `x` bis `z` abgelegt werden, sobald die Funktion fertig ist. Für den Aufrufer der Funktion sind die Variablen `a` bis `c` nicht sichtbar.

2.3.3 Debugger

Als Computer noch mit mechanischen Bauelementen arbeiteten, konnte es passieren, dass sich tatsächlich ein Insekt in die Eingeweide einer solchen Rechenmaschine verirrte und dort z.B. einen Schalter zur Fehlfunktion brachte. Daher stammt der Ausdruck „Bug“ (engl.: Käfer, Insekt) für Programmfehler. In heutigen Zeiten handelt es sich jedoch meistens nicht um einen Fehler der Hardware (wie z.B. defekter Hauptspeicher), sondern meistens um Fehler, die der Programmierer selbst gemacht hat. Da man zunächst die Codezeile finden muss, in der die Ursache für das Fehlverhalten des Programms steckt, dauert die Suche nach solchen Fehlern meistens länger als die Behebung (und meist auch länger als erwartet).

Wie in vielen anderen Entwicklungsumgebungen auch gibt es in MATLAB einen Debugger. Mit diesem Werkzeug kann man ein Programm während der Abarbeitung anhalten,

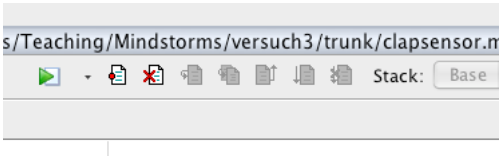


Abbildung 2: Debugger-Menüleiste

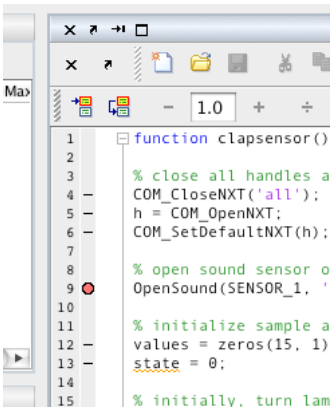


Abbildung 3: Gesetzter Breakpoint in Zeile 9

den Inhalt aller Variablen inspizieren, dann schrittweise mit der Abarbeitung fortfahren, etc. Damit lassen sich Bugs wesentlich schneller finden als durch Betrachten des fehlerhaften Programmverhaltens.

Zum Benutzen des Debuggers muss man lediglich in einer Zeile des Programms einen *Breakpoint*, also den Punkt, an dem die Abarbeitung des Programms anhalten soll, setzen. Dies geschieht in MATLAB, indem man im Editor auf den Strich rechts neben der Zeilennummer klickt. Es erscheint dann ein roter Kreis (Abb. 3). Startet man dann das Programm, wird es vorläufig an dieser Stelle unterbrochen, und die in Abb. 2 gezeigten Symbole in der Menüleiste werden aktiv. Man kann dann das Programm schrittweise und interaktiv abarbeiten lassen (Step), weiter ausführen lassen bis zum nächsten Breakpoint oder Programmende (Continue) oder auch den Debugmodus verlassen (Exit). Im Debugmodus erscheint vor dem Prompt im Befehlsfenster ein ‚K'. Dies bedeutet, dass man die Variablen, die in der aktuellen Funktion gültig sind, im Workspace betrachten und sogar verändern kann, bevor das Programm weiterläuft. Im Befehlsfenster kann man Variablen plotten lassen und im Prinzip auch beliebige andere Befehle ausführen.

3 Durchführung

Dauer: ca. 180 Minuten

3.1 Sensor testen (ca. 30 min)

1. Schließen Sie den Brick per USB an den Rechner an,erstellen Sie ein Objekt und öffnen Sie eine USB-Verbindung

2. Schließen Sie den Schallsensor an den ersten Sensorport des Bricks an, die Ausgabe der Werte soll in dB erfolgen.

   ```
   ev3_obj.sensor1.mode=DeviceMode.NXTSound.DB;
   ```

3. Lesen Sie einen einzelnen Messwert mit Hilfe der Methode `value` aus. Vergewissern Sie sich, dass der Sensor Daten liefert, die den Erwartungen entsprechen (z.B. indem Sie einmal ein lautes Geräusch erzeugen, während Sie den Sensor auslesen). Beachten Sie, dass die Ausgabe des Befehls nicht angezeigt wird, wenn Sie ein Semikolon ans Ende stellen!

4. Schließen Sie die Verbindung wieder mit der Methode `disconnect`.

3.2 Sensor auslesen (ca. 90 min)

Wir wollen nun über einen längeren Zeitraum Messwerte auslesen. Dazu bedienen wir uns der `for`-Schleife von MATLAB. Da die Programmierung von Schleifen auf der Kommandozeile umständlich werden kann, erstellen wir eine eigene Funktion, die man immer wieder aufrufen kann.

1. Legen Sie mit Hilfe des MATLAB-Editors eine neue Datei mit dem Namen `soundsensor.m` an. Benutzen Sie dafür den Befehl `edit` oder das MATLAB-Menü.

2. Legen Sie den Funktionskopf an. Unsere Funktion `soundsensor` benötigt keine Argumente und gibt auch keinen Wert zurück.

3. Zu Beginn der Funktion soll – wie in Aufgabe 3.1 – die USB-Verbindung geöffnet und der Schallsensor initialisiert werden. Geben Sie also die entsprechenden Befehle in den Editor ein.

4. Der Hauptteil der Funktion soll aus einer `for`-Schleife bestehen, die die Laufvariable `i` von 1 bis 1000 durchzählt. Platzieren Sie innerhalb der Schleife eine `value` Methode, so dass diese 1000 mal ausgeführt wird.

5. Am Ende der Funktion soll die USB-Verbindung wieder abgebaut werden. Fügen Sie also hinter der `for`-Schleife einen Befehl zum Schließen der Verbindung an. Die Funktion besteht jetzt aus drei Teilen: Initialisieren der Verbindung und des Sensors, Lesen der Daten in einer Schleife, Schließen der Verbindung. Speichern Sie die Funktion.

6. Gehen Sie zurück ins MATLAB-Hauptfenster und führen Sie die Funktion mit dem Befehl `soundsensor` aus. In der Ausgabe sollten jetzt nacheinander 1000 Messwerte erscheinen, wie Sie vom Sensor gemessen werden. Falls keine Ausgabe erscheint, stellen Sie sicher, dass hinter dem `value`-Befehl kein Semikolon steht, was die Ausgabe unterdrückt. Korrigieren Sie die Funktion und wiederholen Sie diesen Schritt falls nötig.

Wir wollen nun die Messdaten speichern, damit wir die Daten auf einfache Art und Weise auswerten können. Dazu ändern wir die Funktion so, dass Sie die gemessenen Daten in einem MATLAB-Array zurückgibt.

7. Gehen Sie zurück in den Editor. Ändern Sie den Funktionskopf so, dass die Funktion eine Variable namens `values` an den Aufrufer der Funktion zurückgibt.

8. Im ersten Teil der Funktion (vor der Schleife) müssen wir das Array `values` initialisieren. Es soll zunächst keine Daten enthalten. Fügen Sie daher den Befehl

   ```
   values = [];
   ```

   vor der Schleife in die Funktion ein.

9. Die Messwerte sollen nun – anstatt direkt ausgegeben zu werden – in jedem Schleifendurchlauf an das Array `values` angehängt werden. Realisieren Sie dies, indem Sie dem Element von `values` mit dem Index `i` in der Schleife den gemessenen Wert zuweisen.

10. Speichern Sie die Funktion, gehen Sie wieder zurück in das MATLAB-Hauptfenster und lesen Sie 1000 Messwerte ein, indem Sie den Befehl

    ```
    v = soundsensor;
    ```

    eingeben. Der Befehl sollte ohne Ausgabe verlaufen. Nach einigen Sekunden sollte `v` ein Array mit 1000 Werten sein.

11. Da die Messdaten nun im Array `v` gespeichert sind, können wir sie auch nach der eigentlichen Messung verarbeiten. Stellen Sie sie als Beispiel mit dem Befehl `plot` grafisch dar. Stimmt der Verlauf des Schalldrucks mit dem überein, was Sie erwartet haben? Wiederholen Sie die Messung und die Darstellung ein paar mal, um das zu überprüfen. Hinweis: Da die Anzahl der Messpunkte pro Sekunde davon abhängt, wie leistungsfähig der PC ist, müssen Sie die Anzahl der Messwerte ggf. erhöhen, um über einen längeren Zeitraum messen zu können.

3.3 Klatschsensor (ca. 60 min)

In diesem Teil des Versuchs wollen wir ein Programm schreiben, das nicht nur Messdaten einliest, sondern auch auf Ereignisse reagiert. Dazu verknüpfen wir die Messungen vom Schallsensor mit dem Plot einer Ampel. Wenn der Schalldruck einen gewissen Wert überschreitet, wird die oberste Lampe ein- bzw. ausgeschaltet. Dadurch bekommen wir einen einfachen Klatschsensor.

1. Öffnen Sie die Datei `clapsensor.m`. Darin befindet sich bereits folgendes Gerüst:

```
function clapsensor()
% open a new object
ev3_obj=EV3();
ev3_obj.connect('usb','beep','on');

% set mode to dB
ev3_obj.sensor1.mode = DeviceMode.NXTSound.DB;

% clap detection threshold
clapThreshold = 30;
numSamples = 15;

% initialize sample array and state of lamps
values = zeros(numSamples, 1);
changes = zeros(numSamples - 1, 1);
states = [0 0 0];

% initially, create the figure without data, all lamps off
plot_handles = []; % will be initiliazed by clapsensorPlot
plot_handles = clapsensorPlot(plot_handles, values, values, [0 0 0], 0);

max_iterations = 50;
iterations = 0;
while iterations < max_iterations  && isvalid(plot_handles.h_fig)
% start loop
    iterations = iterations + 1;

    % get a new sample from the sensor
    s =  %...

    % throw away oldest sample and add the new one at the end
    %...

    % display plot and lamps
    clapsensorPlot(plot_handles, values, changes, states, clapThreshold);

    % wait 10ms between samples
    pause(0.01);
end

% close object
```

```
ev3_obj.disconnect();
end
```

Die Funktion soll Messwerte vom Sensor einlesen, aber immer nur die 15 neuesten Messwerte in der Variable `values` vorhalten. In der Schleife fehlen noch die Befehle, um einen Messwerte einzulesen und den ältesten Messewert zu verwerfen.

Nach jedem Messwert wartet das Programm 10 Millisekunden, um nicht unnötig viele Messwerte verarbeiten zu müssen. Da die Verarbeitung des Messwerts durch den PC aber unterschiedlich lange dauern kann (z.B. wenn andere Prozesse CPU-Ressourcen verbrauchen), ist diese Vorgehensweise relativ ungenau. Um Messwerte in gleichmäßigen Intervallen zu nehmen, kann man in Matlab auch Timer-Objekte verwenden. Diese werden in Versuch 5 (Lichtsensor) genauer beschrieben.

Die Funktion

```
clapsensorPlot(plot_handles, values, changes, states, clapThreshold)
```

dient zur Darstellung der Messerte. Sie erzeugt ein Fenster mit 5 Plots, 2 für Messwerte und 3 für die Ampeln. Zunächst sind alle Plots leer, bzw. die Ampeln aus. In den folgenden Unterpuntkten werden schrittweise sinnvollere Argumente übergeben und die Plots somit vervollständigt.

2. Fügen Sie an den Stelle, die mit den drei Punkten gekennzeichnet ist, Befehle ein, um einen neuen Messwert einzulesen und den ältesten Wert zu verwerfen. Speichern Sie, wechseln Sie zum MATLAB-Hauptfenster und führen Sie die Funktion aus. Sie sollten sehen, wie die jeweils 15 letzten Messwerte in einem MATLAB-Fenster durchlaufen.

3. Fügen Sie jetzt eine `if`-Anweisung hinzu, die prüft, ob der aktuellste Messwert einen bestimmten Schwellwert überschreitet. Welcher Wert sinnvoll ist, müssen sie experimentell bestimmen (vorgegebener Beispielwert 30). Falls der Schwellwert überschritten ist (also ein Klatschen gemessen wurde), soll wiederum geprüft werden, welchen Zustand die oberste Lampe aktuell hat, und dann in den jeweils anderen Zustand geschaltet werden. Die Variable `states` dient dazu, den Status der Lampen, d.h. ob sie momentan an- oder ausgeschaltet sind, zu speichern, zunächst sind alle Lampen aus `states = [0 0 0 ]`. *Hinweis:* Sie müssen dazu zwei `if`-Anweisungen ineinander verschachteln!

   Setzen sie innerhalb der äußeren `if`-Anweisung einen Breakpoint, so dass das Programm jedesmal angehalten wird, wenn der Schwellwert überschritten ist. So können Sie schrittweise (mit ‚Step') überprüfen, ob der Inhalt der `if`-Anweisung richtig ausgeführt wird und dann (mit ‚Continue') das Programm fortsetzen.

4. Um das Verhalten des Klatschsensors zu verbessern, kann man anstelle von großen Schalldruckwerten auch große *Änderungen* des Schalldrucks als Kriterium verwenden. Dies entspricht einer Flankendetektion: Die Lampe schaltet nur dann um, wenn der Schalldruck um ein gewisses Maß *ansteigt*. Bestimmen Sie dazu mit Hilfe des

`diff`-Befehls die Änderung des Schalldrucks in den letzten 15 Messwerten. Speichern Sie das Ergebnis in der Variable `changes`, damit die Änderungen ebenfalls geplottet werden. Modifizieren Sie dann die `if`-Anweisung so, dass sie das letzte Element dieses Arrays mit einem Schwellwert testet. Verwenden Sie auch hier wieder den Debugger, um das Programm zu beobachten. Reagiert das Progamm sowohl auf steigende als auch auf fallende Flanken? Ist dies gewünscht?

5. (Zusatzaufgabe) Nun sollen noch zwei zusätzliche Lampen der Ampel angesteuert werden, verändern Sie die Funktion so, dass sie auf einen, zwei bzw. drei Klatscher reagiert. Zählen sie hierzu die Anzahl der Klatscher in den 15 Messwerten. Je nach Anzahl der Klatscher sollen eine, zwei oder alle drei Lampen an- bzw. ausgeschaltet werden. *Hinweis:* Bis jetzt haben Sie immer nur den aktuellsten Messwert bzw. die aktuellste Messwertdifferenz als Schaltkriterium verwendet. Jetzt benötigen Sie auch die restlichen der 15 Messwerte.