

EV3-Toolbox: How-To

Die neue Toolbox zur Ansteuerung der Lego Mindstorms basiert auf OOP. Zentral stehen die *EV3-Klasse*, die *Sensor-Klasse* und die *Motor-Klasse*, mit denen der User größtenteils arbeiten wird. Die Toolbox bildet darin den physikalischen Brick auf einen Virtuellen ab. Eine Instanz der EV3-Klasse hat also **vier Motor- und vier Sensorobjekte**, deren „physischer“ Status sich wiederum abfragen lässt und über die Befehle direkt an die Devices gesendet wird. Im Folgenden werde ich einen kurzen Überblick über den Umgang mit der Toolbox geben – Verbindung herstellen, einfache Befehle schicken, wichtige Funktionen, Statuswerte abfragen, etc. -> Sollten einige Dinge unklar bleiben, habe ich außerdem eine erste (unfertige) Version der Doku geschrieben. Mit *help *Klasse/Methode/Parameter** können in MATLAB Dinge wie erlaubte Parameterwerte abgefragt werden. Für die Klassen selber finden sich dort auch noch ein paar Beispielskripte.

- **Verbindung herstellen**

Entweder USB oder Bluetooth sind möglich. Zur USB-Verbindung einfach den Brick per USB-Kabel anschließen. Zur BT-Verbindung muss *per Konsole* das `btconnect`-Skript aufgerufen werden. (Der Brick-Name findet sich u. A. ganz oben auf dem Display des eingeschalteten Bricks, standardmäßig *EV3*)

```
btconnect *Brick-Name*
```

Beides ließ sich erfolgreich testen; für die Bluetooth-Verbindung wurde der Dongle „LogiLink BT0015 V4.0 EDR Class1 Micro“ benutzt.

Zum Erstellen und Verbinden des EV3-Objektes in MATLAB dann:

```
brickObj = EV3();  
brickObj.connect('ioType', 'usb'); % Bei USB-Verbindung  
brickObj.connect('ioType', 'bt', 'serPort', '/dev/rfcomm0'); % Bei BT-  
Verbindung
```

Den Pfad zum seriellen Port bei einer Bluetooth-Verbindung steht nach erfolgreicher Verbindung per `btconnect` in der Konsole und sollte meistens `/dev/rfcomm0` sein. Ein sauberes Disconnect ist mit `brickObj.disconnect()` möglich. Alles andere (löschen der Objekte ohne disconnect z. B.) kann langfristig zu Problemen führen.

- **Allg. Kommunikation**

Da die allg. Kommunikation mit dem EV3 und seinen Sensoren und Motoren größtenteils intuitiv sein sollte, gehe ich hierauf nur kurz ein. Vollständige Listen aller Parameter und Methoden finden sich in der Doku der jeweiligen Klasse.

Nach Erstellen der Verbindung lassen sich Befehle zum Brick schicken.

brickObj.beep() lässt den Brick bspw. einen kurzen Ton abspielen; über den Parameter *brickObj.isConnected* lässt sich abfragen, ob eine Verbindung hergestellt werden konnte. Die Motor- und Sensorobjekte sind nach ihren entsprechenden Ports benannt, sensor1-4 und motorA-D. Ein angeschlossener Motor an Port A lässt sich bspw. wie folgt starten:

```
%brickObj.motorA.update();  
brickObj.motorA.power = 50 ;  
brickObj.motorA.start();  
cnt = brickObj.motorA.tachoCount;
```

(*update()* ist nur nötig, wenn der physische Motor erst nach Erstellen der Verbindung angeschlossen wurde, dazu gleich mehr.)

Ähnlich funktioniert das Abfragen von Sensorwerten:

```
%brickObj.sensor1.update();  
val = brickObj.sensor1.value;
```

- **update()**

Die *update()-Methode*, die jede der drei Klassen zur Verfügung stellt, ist dazu da, den virtuellen Brick (aka das EV3-Objekt) auf einen Stand mit dem physischen Brick zu halten. Wenn man Sensoren oder Motoren an- oder abschließt, muss also entweder für den jeweiligen Port (wie im Beispiel oben) oder für alle auf einmal (*brickObj.update()*) die *update()*-Methode aufgerufen werden, die ihrerseits Parameter wie *sensorAtPort* und *motorAtPort* setzt.

- **setProperty()**

Als Alternative zum ‚Parameter per Hand setzen‘, was für viele Parameter auf einmal sehr umständlich ist, bieten die Klassen die Methode *setProperty()* an, mit der mehrere Parameter auf einmal gesetzt werden. Hierbei ist Reihenfolge der Parameter egal. Beispielsweise für einen Motor:

```
brickObj.motorA.setProperty('power', 50, 'limitValue', 360,  
 'speedRegulation', 'on');
```

- **status, type, mode**

Bei Sensoren und Motoren lassen sich über die drei Parameter *status*, *type* und *mode* (die nach jedem *update()*-Aufruf aktualisiert werden) der aktuelle „Status“ des jeweiligen Ports abfragen. *status* steht dabei für den Verbindungsstatus, *type* für den Device-Typ (Motor, Farbsensor, Gyrosensor, etc.) und *mode* für den Modus des Devices (z. B. Ambient oder Reflected bei Farbsensoren, Degrees oder Rotations bei

Motoren, etc.) -> *mode* ist dementsprechend der Modus, in dem Werte abgefragt werden. Eine Liste aller möglichen Status-Parameter findet sich in `typedata.ods` und in den jeweiligen Klassen (`ConnectionType`, `DeviceType`, `DeviceMode`).

- **SyncMotor**

Zur Ansteuerung zweier Motoren gleichzeitig gibt es die `SyncMotor`-Klasse. Mit dieser lassen sich zwei Motoren gleichzeitig über je einen Befehl starten und stoppen; das Festhalten eines Motors stoppt auch gleichzeitig den Anderen. Dies ist nützlich, wenn man eine Art Gefährt mit Rädern baut, das geradeaus fahren soll. Um ein `SyncMotor`-Objekt zu erstellen, bietet die `EV3`-Klasse eine `coupleMotors()`-Methode an.

```
syncedMotor = brickObj.coupleMotors('AB', 'power', 50);
```

Mit `syncedMotor` kann nun wie mit einem normalen Motor-Objekt gearbeitet werden. (Mit ein paar Einschränkungen; der Tachostand kann beispielsweise nicht von beiden Motoren gleichzeitig ausgelesen werden)

- **Infos**

- mehrere Bricks gleichzeitig: Einmal erfolgreich getestet, aber noch nicht ausreichend geprüft
- Bluetooth-Verbindung: Mit dem alten Dongle zur Verbindung mit den NXTs ließ sich keine Verbindung herstellen, auch sonst scheinen viele BT-Adapter Probleme zu machen
- zum Aufbau der Toolbox: die hier vorgestellten Klassen stellen nur das oberste Layer der Toolbox dar. Jede Klasse hat nach Erstellen der Verbindung ein Objekt der Klasse `Brick`. Dieses Objekt wird als Interface zu den Methoden der darunter liegenden Layer benutzt.
- Das manuelle Erstellen von Motor- und Sensorobjekten ist aktuell nicht vorgesehen, aber möglich. Die Motor- und Sensor-Klassen haben auch die Methoden `connect()` und `disconnect()`. Diese erwarten ein Objekt der Klasse `Brick` (s. voriger Punkt). Da sähe bspw. wie folgt aus:

```
m = Motor(Brick('ioType', 'usb'));
```