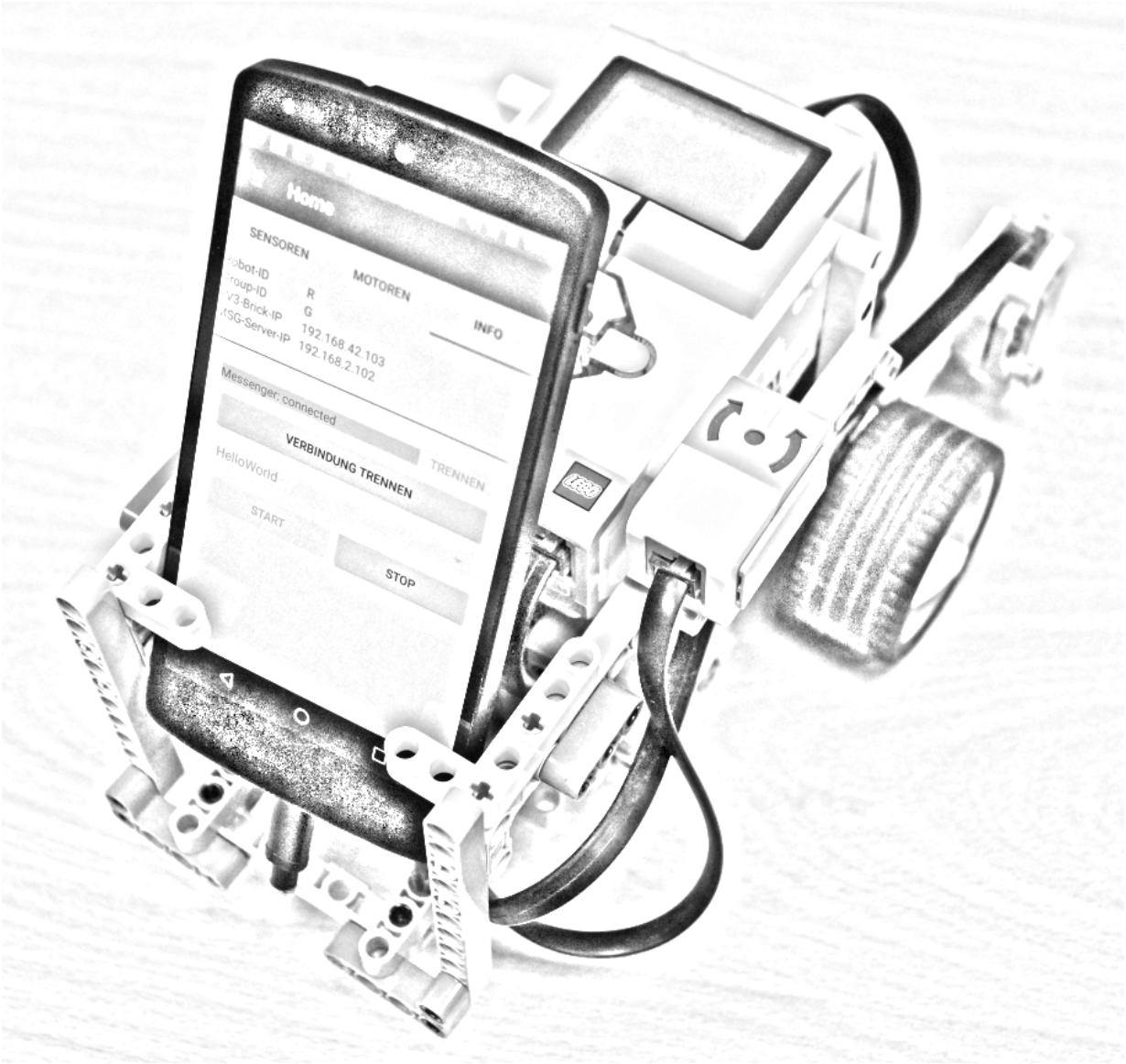
Mindroid Workshop Dokumentation

TECHNISCHE UNIVERSITÄT DARMSTADT

NeXT Generation on Campus
TU Darmstadt



Abschnitt 1 Einführung

In dieser Übersicht werden die Funktionen, die zur Steuerung der Roboter zur Verfügung stehen erklärt. Zur Verdeutlichung ein kleines Beispiel:

Typ	Methode und Beschreibung
void	delay(long milliseconds) Verzögert die Ausführung um die angegebene Zeitspanne (Milisekunden)

Die Spalte *Typ* gibt an, welchen Typ der Rückgabewert der Funktion hat. *void* bedeutet, dass kein Wert zurück gegeben wird. In der Klammer hinter dem Funktionsnamen wird angegeben, welche Parameter die Funktion erwartet, und von welchem Typ diese sein müssen. In unserem Beispiel bedeutet dies, dass die *delay*-Methode einen Parameter vom Typ *long* (ganzzahliger Wert) erwartet, welcher *milliseconds* genannt wird. Ein möglicher Funktionsaufruf sieht wie folgt aus:

```
1     public void run(){
2         delay(1000);
3     }
```

Dabei wird die delay-Methode mit 1000 als Parameter aufgerufen. Das bedeutet, die Ausführung wird um $1000ms$ $(= 1s)$ verzögert.

Abschnitt 1.1 isInterrputed

Damit die Ausführung des Programms auch in Schleifen unterbrochen werden kann, sollte jede Schleife die isInterrupted-Methode abfragen.

Beispiel:

```
1         public void run(){
2             while(!isInterrupted()){
3                 // Schleifeninhalt
4             }
5             for(int i=0; i<10 && !isInterrupted(); i++){
6                 // Schleifeninhalt
7             }
8         }
```

Abschnitt 2 Wichtige Funktionen

Hier eine kleine Übersicht über die wichtigsten Funktionen beim Programmieren der Roboter.

Abschnitt 2.1 Fahren

```
import org.mindroid.api.ImperativeWorkshopAPI
```

Mögliche Eingabewerte für den *speed*-Parameter liegen zwischen 0 und 1000. Eine maximale Geschwindigkeit von 300 sollte ausreichen. Niedrigere Geschwindigkeiten schonen den Akku. Die Distanz wird im *distance*-Parameter immer als Kommazahl in Zentimetern (cm) angegeben (z.B.: 20cm werden als $20.0f$ angegeben)

Typ	Methode und Beschreibung
void	setMotorSpeed(int speed) Bestimmt die Geschwindigkeit für Fahrmethoden ohne *speed*-Parameter.
void void	forward() backward() Fahren mit der von *setMotorSpeed*(...) gesetzten Geschwindigkeit.
void void	driveDistanceForward(float distance) driveDistanceBackward(float distance) Fahren mit der von *setMotorSpeed*(...) gesetzten Geschwindigkeit Die Distanz muss in Zentimetern angegeben werden.
void void void void	forward(int speed) backward(int speed) driveDistanceForward(float distance, int speed) driveDistanceBackward(float distance, int speed) Wie oben, nur dass der *speed*-Parameter die von *setMotorSpeed*() gesetzte Geschwindigkeit überschreibt. Nach Beendigung des Aufrufs, wird wieder die vorher gesetzte Geschwindigkeit genutzt.
void void void void	turnLeft(int degrees) turnRight(int degrees) turnLeft(int degrees, int speed) turnRight(int degrees, int speed) Dreht den Roboter um den im *degrees*-Parameter bestimmten Wert. Der *Speed*-Parameter verhält sich wie bei den anderen Methoden.
void	stop() Stoppt sofort alle Motoren.

Abschnitt 2.2 Sensoren

```
import org.mindroid.api.ImperativeWorkshopAPI
```

Typ	Methode und Beschreibung
float	getAngle() Liefert den Winkel des Gyrosensors in Grad
float	getDistance() Liefert die vom Ultraschallsensor gemessene Distanz in Zentimetern
Colors Colors	getLeftColor() getRightColor() Liefert den Wert des Linken/Rechten Farbsensors Farbwerte: Colors.BLACK, Colors.BLUE, Colors.BROWN, Colors.GREEN, Colors.RED, Colors.WHITE, Colors.YELLOW, Colors.NONE

Abschnitt 2.3 Kommunikation

```
import org.mindroid.api.ImperativeWorkshopAPI
```

Typ	Methode und Beschreibung
boolean	hasMessage() Prüft ob Nachricht vorhanden ist
MindroidMessage	getNextMessage() Ruft nächste Nachricht ab
void	sendBroadcastMessage(String message) Sendet eine Nachricht an alle Roboter
String	getRobotID() Gibt den Namen des Roboters zurück.
void	sendLogMessage(String logmessage) Sendet eine Nachricht an den Message Server
void	sendMessage(String destination, String message) Sendet eine Nachricht an den *destination*-Roboter

Um eine Nachricht zu empfangen, muss zuerst mit *hasMessage*() überprüft werden ob eine Nachricht vorhanden ist. Liefert *hasMessage*() true zurück, kann mit *getNextMessage*() eine Nachricht abgerufen werden. Das Beispiel in Listing 1 zeigt wie das geht.

```
1         if (hasMessage()){
2           String msg = getNextMessage().getContent();
3         }
```

Listing 1: Beispiel zum Abrufen einer Nachricht

broadcastMessage(...) schickt eine Nachricht an alle mit dem selben Message-Server verbundenen Roboter.

Abschnitt 2.4 MindroidMessage

Um die von *getNextMessage*() zurückgegebene Nachricht verarbeiten zu können, muss ein zusätzlicher import hinzugefügt werden.

import org.mindroid.common.messages.server.MindroidMessage;

Typ	Methode und Beschreibung
String	getContent() Liefert den Inhalt der Nachricht zurück
RobotID RobotID	getDestination() getSource() Liefert die Quelle/das Ziel der Nachricht an

Abschnitt 2.5 Brick

Abschnitt 2.5.1 Display

Typ	Methode und Beschreibung
void	clearDisplay() Löscht den Aktuellen Inhalt des Displays
void void	drawString(String text) drawString(String text, int row) Schreibt den im *text*-Parameter gegebenen Text auf das Display. Der Parameter *row* bestimmt die zu beschreibende Zeile. Wird der Parameter *row* weggelassen, wir in die Mittlere Zeile geschrieben.

[1] https://services.informatik.hs-mannheim.de/~ihme/lectures/LEGO_Files/01_Anfaenger_Graphisch_EV3_BadenBaden.pdf

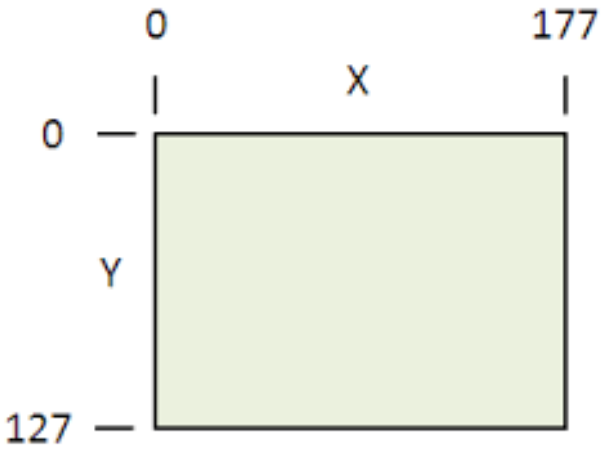


Abbildung Abschnitt 2.1: Koordinaten der Pixel des Displays des EV3[1]

Abschnitt 2.5.2 Buttons

import org.mindroid.impl.brick.Button;

Typ	Methode und Beschreibung
boolean	isDownButtonClicked()
boolean	isEnterButtonClicked()
boolean	isLeftButtonClicked()
boolean	isRightButtonClicked()
boolean	isUpButtonClicked()

Die Funktionen liefern *true* wenn der entsprechende Button gedrückt wurde. Die Benennung der Buttons kannst du Abbildung Abschnitt 3.1 auf Seite 8 entnehmen

Abschnitt 2.5.3 Sound

Typ	Methode und Beschreibung
void	setSoundVolume(int volume)
void	playBeepSequenceDown()
void	playBeepSequenceUp()
void	playBuzzSound()
void	playDoubleBeep()
void	playSingleBeep()

Der Parameter *volume* nimmt Werte von 0 bis 10 entgegen.

Abschnitt 2.5.4 LED

Typ	Methode und Beschreibung
void	setLED(int mode) Lässt die LED des EV3 im angegebenen Modus leuchten Der Parameter *mode* kann entweder als Ganzzahl von 0 bis 9 oder als Konstante angegeben werden. Siehe Tabelle Abschnitt 2.1

Tabelle Abschnitt 2.1: Funktion der einzelnen Modi der LED

Modus (Parameter *mode*)		**Farbe**	**Intervall**
Wert	Konstante		
0	LED_OFF	Aus	Aus
1	LED_GREEN_ON	Grün	Dauer
2	LED_GREEN_BLINKING	Grün	Blinken
3	LED_GREEN_FAST_BLINKING	Grün	Schnell Blinken
4	LED_YELLOW_ON	Gelb	Dauer
5	LED_YELLOW_BLINKING	Gelb	Blinken
6	LED_YELLOW_FAST_BLINKING	Gelb	Schnell Blinken
7	LED_RED_ON	Rot	Dauer
8	LED_RED_BLINKING	Rot	Blinken
9	LED_RED_FAST_BLINKING	Rot	Schnell Blinken

Abschnitt 3 EV3 Tasten

Abbildung Abschnitt 3.1 zeigt dir wie die Tasten am EV3-Brick genannt werden. Die Enter-Taste wird zum Bestätigen genutzt, mit der Escape-Taste, geht es ein Menü zurück.

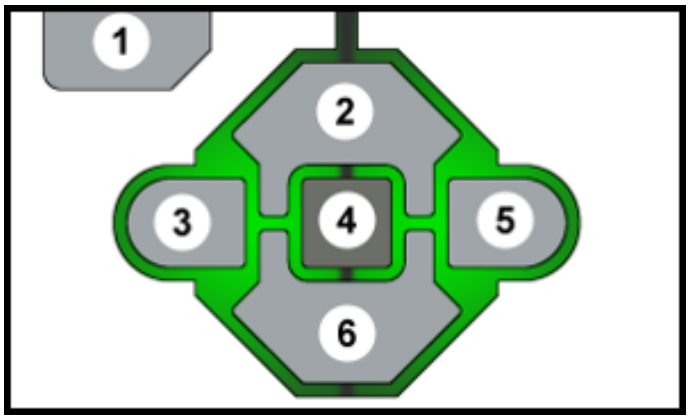


Abbildung Abschnitt 3.1: EV3-Tastenbelegung[2]

Die Bedeutung der Tasten kannst du der folgenden Aufzählung entnehmen.

1. **Escape / Zurück**
2. **Up / Hoch**
3. **Left / Links**
4. **Enter / Bestätigen**
5. **Right / Rechts**
6. **Down / Unten**

Abschnitt 4 Kurze Übersicht über Java

Schleife mit Bedingung	while(Bedingung) {Programmcode} Beispiel: while(i<100){...}
Zählschleife	for(Start; Bedingung; Zählschritte) {Programmcode} Beispiel: for(int i=0;i<10;i++){...}

Bedingung	if(Bedingung)
	{wenn die Bedingung wahr ist, wird dieser Code ausgeführt}
	else
	{wenn die Bedingung falsch ist, wird dieser Code ausgeführt}

[2] Quelle http://www.ev3dev.org/images/ev3/labeled-buttons.png