

Mindroid-Pilotworkshop am 30.08.2017

Inhaltsverzeichnis

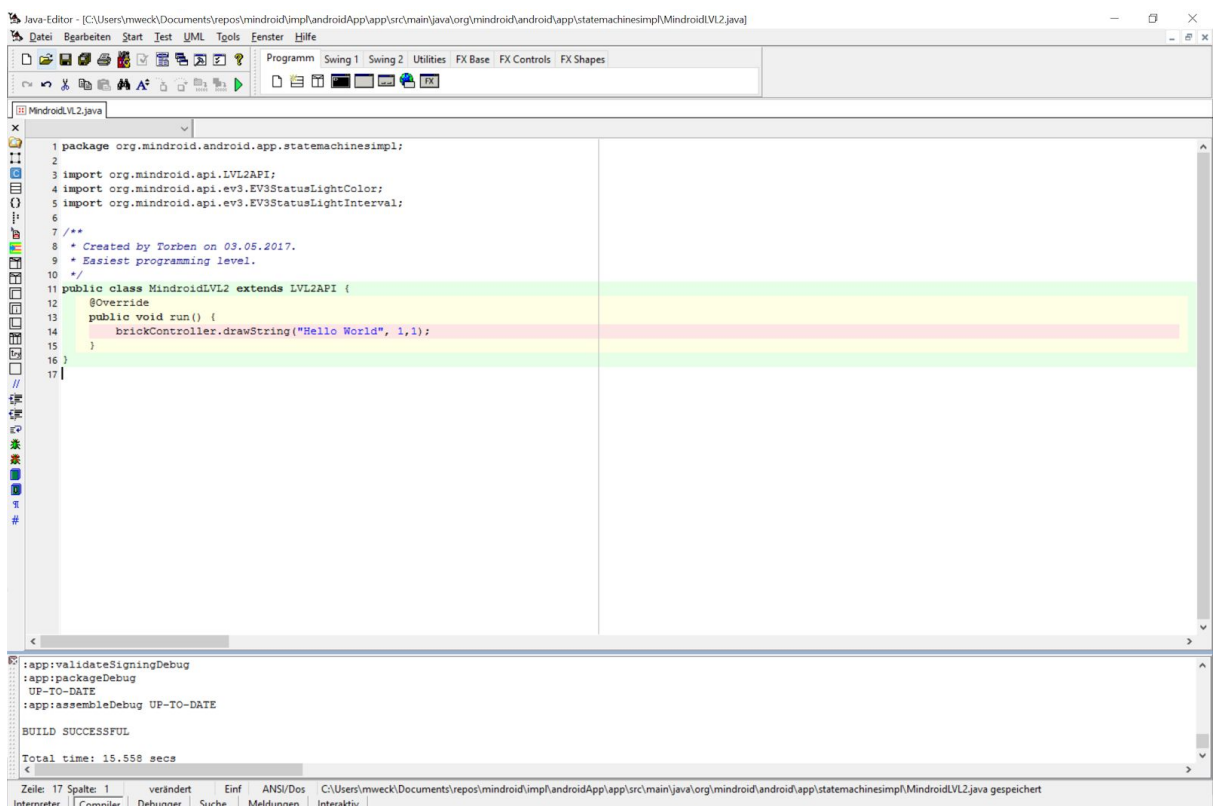
Inhaltsverzeichnis	1
Aufgabe 1: Hallo Welt!	2
Testlauf starten	2
Erklärung des Quelltextes	7
Erste eigene Änderung durchführen	8
Übrigens	8
Aufgabe 2: Den Roboter kennenlernen	9
2.1: Fahren	10
2.2: Abstand messen	11
2.3: Farben messen	12
2.4: Verteiltes "Hallo Welt!"	13
Nächste Schritte	16
Nützliche Imports	16
Aufgabe 3: Wand-Ping-Pong	17
Aufgabe 4: Koordiniertes Wand-Ping-Pong	17
Aufgabe 5: Mähroboter	18
Aufgabe 6: Platooning	19
Aufgabe 7: Verfolgung	20

Aufgabe 1: Hallo Welt!

In der Informatik ist es üblich, ein Hallo-Welt-Programm¹ zu schreiben, wenn man eine Programmierungsumgebung kennenlernt. Deshalb fangen wir damit an.

Testlauf starten

1. Öffne den **Link "Java-Editor"**  auf dem Desktop des Laptops.
2. Es öffnet sich die **Programmierungsumgebung Java-Editor**². In der Mitte findest du den Quelltext des aktuellen Programms, hinterlegt mit Farben, die die Orientierung erleichtern sollen.



```
1 package org.mindroid.android.app.statemachinesimpl;
2
3 import org.mindroid.api.LVL2API;
4 import org.mindroid.api.ev3.EV3StatusLightColor;
5 import org.mindroid.api.ev3.EV3StatusLightInterval;
6
7 /**
8  * Created by Torben on 03.05.2017.
9  * Easiest programming level.
10 */
11 public class MindroidVL2 extends LVL2API {
12     @Override
13     public void run() {
14         brickController.drawString("Hello World", 1,1);
15     }
16 }
17
```

app:validateSigningDebug
app:packageDebug
UP-TO-DATE
app:assembleDebug UP-TO-DATE
BUILD SUCCESSFUL
Total time: 15.558 secs

Wie das Programm genau funktioniert, sehen wir uns gleich an. Zunächst einmal wollen wir es auf das Smartphone spielen, um zu sehen, was passiert.

3. Um das Programm auf das **Smartphone** zu laden, verbindest du es per **USB** mit dem PC.
4. Damit du zum Aufspielen neuer Programme das Handy später nicht immer wieder an den PC stecken musst, starten wir eine **drahtlose**



¹ <https://de.wikipedia.org/wiki/Hallo-Welt-Programm>

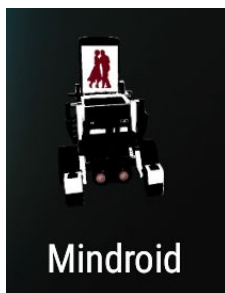
² <http://javaeditor.org/doku.php>

Verbindung. Klicke dazu auf **HandyVerbinden.bat** auf dem Desktop.

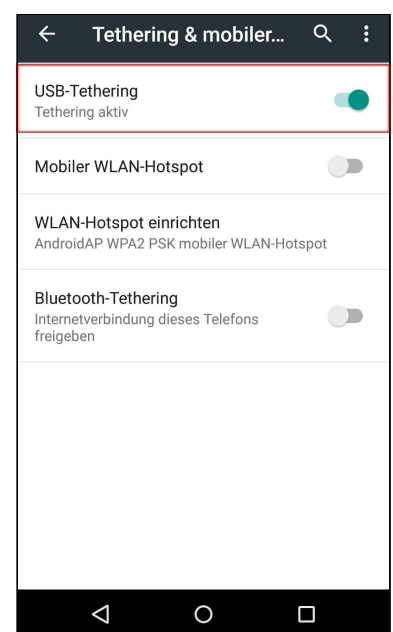
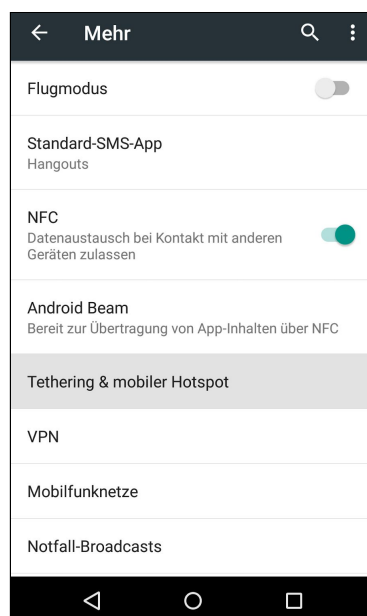
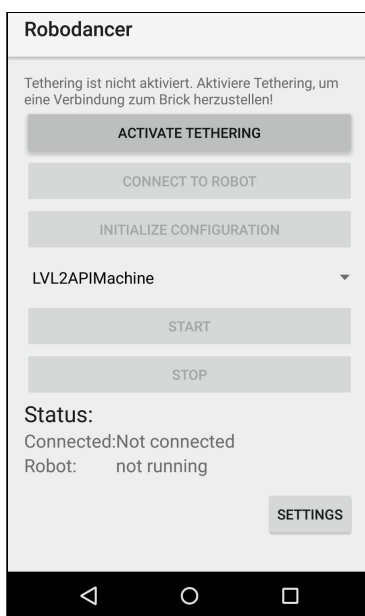
5. Folge den Anweisungen des Skripts, **bis dort steht, dass Schritt 1 geschafft ist.**
Bitte schließe das Fenster nicht, bevor du in der Anleitung dazu aufgefordert wirst!

```
-----
Willkommen! Bitte verbinde das Android-Gerät mit dem Computer und drücke dann Enter:
Die Wifi-Verbindung wird gestartet...
restarting in TCP mode port: 55555
connected to 192.168.4.106:55555
-----
Bitte noch nicht das Fenster schließen. Schritt 1 ist geschafft! Folge nun den Anweisungen in der Anleitung.
```

6. **Trenne** das Handy jetzt vom PC und **schalte** den Roboter **ein**.
7. **Verbinde** das Handy mit dem Roboter.
8. Nun startest du das erste Mal die **Mindroid-App** auf dem Handy!



9. Drücke in der App auf **Activate Tethering** und aktiviere Tethering in den Einstellungen (orientiere dich an den Screenshots)



10. Wenn **Tethering eingeschaltet ist**, kannst du dich wieder dem **Skript auf dem PC** zuwenden.
Tippe **ok** und drücke Enter.

Wenn dort steht, dass das **Verbinden erfolgreich** war, kannst du das Fenster schließen.

```
Bitte noch nicht das Fenster schließen. Schritt 1 ist geschafft! Folge nun den Anweisungen in der Anleitung.ok
connected to 192.168.4.106:5555
-----
Das Verbinden war erfolgreich! Du kannst dieses Fenster schließen.
Drücken Sie eine beliebige Taste . . .
```

11. Lasse das Handy während des Workshops am Roboter stecken, sonst muss die Verbindung erneut hergestellt werden.

12. Wechsele am PC nun zurück in den Java Editor. Drücke den **blauen Play-Button**.



(Der Button ändert seine Farbe nach dem Durchlauf auf Grün.)

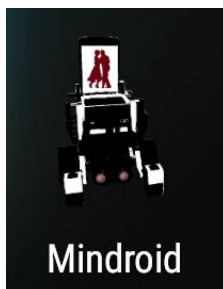
Dein "Hallo Welt"-Programm wird jetzt an das Handy übertragen.

13. Jetzt sollte in der **Konsolenausgabe** am unteren Bildrand in etwa folgender Text erscheinen:

```
.:app:installDebug
Installing APK 'app-debug.apk' on 'Nexus 5 - 5.1.1' for app:debug
Installed on 1 device.

BUILD SUCCESSFUL
Total time: 35.483 secs
exit
```

14. Öffne wieder die **Mindroid-App** auf dem Handy. Nun werden noch ein paar letzte Vorbereitungen getroffen, bevor du dein Programm starten kannst. Die folgenden Schritte musst du nur einmal machen, danach ist der Roboter startbereit.




15. Navigiere im Startmenü des Roboters zu **PAN**. Bestätige mit der **Auswahl taste**.



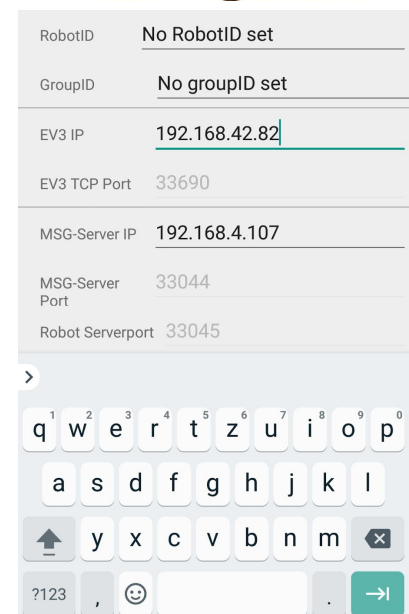
16. Navigiere durch die **USB - Einstellungen**, wie auf den Bildern gezeigt. Bestätige immer mit der **Auswahl**taste.



17. Gehe dann zurück ins **Startmenü** (Zurück-Taste oben links ). Der PAN-Service wird neu gestartet.



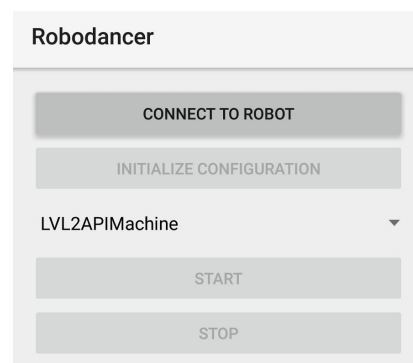
18. Im Startmenü steht jetzt eine IP-Adresse. Vergleiche diese mit der Adresse, die in den **Einstellungen der Mindroid-App** steht. Wenn die Adresse nicht stimmt, passe sie in der App, sodass beide Adressen übereinstimmen.



19. Jetzt kannst du das EV3-Programm starten, wähle dazu **Run Default**. Warte, bis der **Roboter orange leuchtet**.



20. Fertig! Wähle **Verbindung zum EV3-Brick herstellen** in der App auf dem Handy. Danach wähle **Initialisiere Konfiguration**.



21. Wähle in der Liste den Eintrag **Mindrobot** aus und drücke **Start**. Dein "Hello World"-Programm wird jetzt ausgeführt!



Erklärung des Quelltextes

Nachdem du den Roboter erfolgreich getestet hast, gehen wir jetzt daran, uns den Quelltext näher anzusehen.

```
1 package org.mindroid.android.app.statemachinesimpl;
2
3 import org.mindroid.api.LVL2API;
4
5 public class MindroidLVL2 extends LVL2API {
6     @Override
7     public void run() {
8         clearDisplay();
9         drawString("Hello World!", 1, 1);
10    }
11 }
```

- Das **Verhalten des Roboters** befindet sich in der **run-Methode** (Zeilen 7-9). Wenn ein Mindroid-Programm ausgeführt wird, werden die Befehle in dieser Methode nacheinander abgearbeitet.
 - **clearDisplay()** löscht den Display-Inhalt.
 - **drawString(text, xPosition, yPosition)** schreibt einen gegebenen Text an die gegebenen Koordinaten (xPosition, yPosition).
- Daneben gibt es noch die sogenannten **“imports”**. Da die Programm-Bibliotheken der MindroidApp sehr groß sind, hat jede Klasse einen ausführlichen Namen, der dabei hilft, den Überblick zu bewahren. Der Teil vor dem Klassennamen heißt **Paket** (engl. package). Zum Beispiel ist die Klasse **LVL2API** im Paket **org.mindroid.api**.

Erste eigene Änderung durchführen

Um zu sehen, wie die Entwicklung deiner eigenen Roboter-Software im Folgenden ablaufen wird, wirst du nun am Hello-World-Programm eine **kleine Änderung durchführen** und anschließend das **aktualisierte Programm auf den Roboter laden**.

1. Ändere deine **main-Methode**, sodass sie dem folgenden Screenshot ähnelt.

```
1 package org.mindroid.android.app.statemachinesimpl;
2
3 import org.mindroid.api.LVL2API;
4
5 import java.text.SimpleDateFormat;
6 import java.util.Date;
7
8 public class MindroidLVL2 extends LVL2API {
9     @Override
10    public void run() {
11        SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyyy");
12        clearDisplay();
13        drawString("Datum: " + formatter.format(new Date()), 1, 1);
14    }
15 }
```

- Die Imports von **java.text.SimpleDateFormat** und **java.util.Date** sind neu hinzugekommen.
- Außerdem gibt es nun eine Variable **formatter** (Zeile 11), die in der Lage ist, das aktuelle Datum formatiert auszugeben (**formatter.format(new Date())**, Zeile 13)).

2. Betätige jetzt die Schaltfläche **“Starten”** .
3. Nachdem im Compiler-Fenster die Meldung **“BUILD SUCCESSFUL”** erschienen ist, wird auf dem Handy die **Mindroid-App automatisch geschlossen**.
4. Öffne die (neuinstallierte) Mindroid-App.
5. Wähle **“Verbindung zum EV3-Brick herstellen”**
6. Wähle **“Initialisiere Konfiguration”**
7. Wähle im Dropdown **“Mindrobot”** aus und betätige **“Start”**.
8. Auf dem Display sollte nun das aktuelle Datum ausgegeben werden.

Übrigens

Falls du erfahren möchtest, wie man bspw. die **aktuelle Uhrzeit** mithilfe von **SimpleDateFormat** ausgibt, klicke mit Rechts auf den Klassennamen **SimpleDateFormat** und wähle den Eintrag **“API-Hilfe.”** Im sich öffnenden Fenster findest du die **Dokumentation der Formatierungsparameter**, die im Konstruktor verwendet werden.

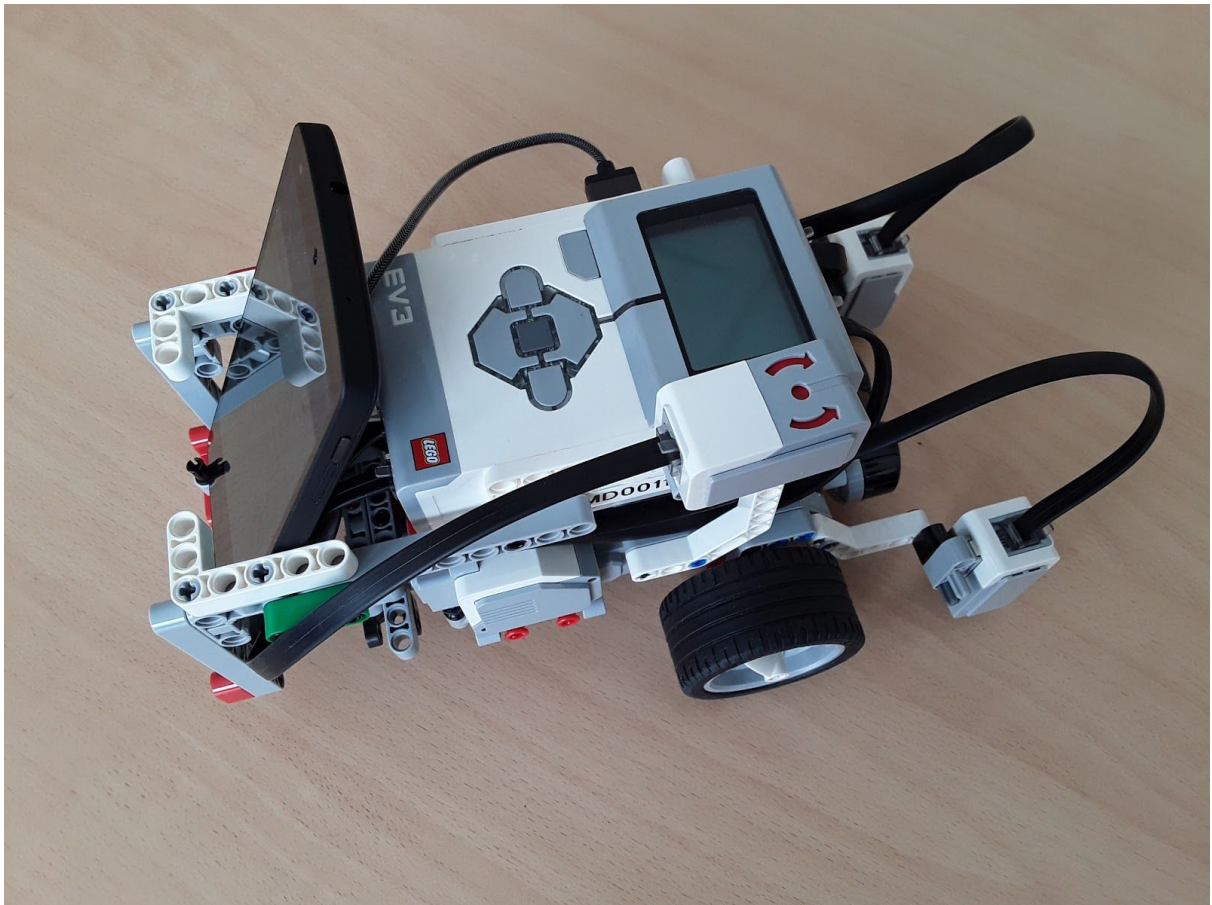
Die vollständige Doku aller Klassen für die Mindroid-App erhältst du, indem du auf dem Desktop den Link **“Mindroid Doku”** anklickst.

Die vollständige Doku aller Klassen der Java-Standardbibliothek erhältst du, indem du auf dem Desktop den Link **“JDK Doku”** anklickst oder die folgende URL aufrufst:

<https://docs.oracle.com/javase/7/docs/api/index.html>

Aufgabe 2: Den Roboter kennenlernen

In dieser Aufgabe wollen wir gemeinsam die Fähigkeiten des Roboters kennenlernen. Das Display ist nützlich, um bestimmte Informationen schnell auszugeben, aber wirklich nützlich wird der Roboter erst durch seine beiden **Antriebsmotoren**, den **Gyrosensor**, die **beiden Lichtsensoren** und den **Ultraschallsensor**.



2.1: Fahren

Wir schauen uns als Erstes an, wie man den Roboter fahren lassen kann. Der Roboter unterstützt folgende **Bewegungsrichtungen**:

- Vorwärts fahren,
- Rückwärts fahren,
- Drehen nach Rechts und
- Drehen nach Links

Um die Fahr-Fähigkeiten des Roboters kennenzulernen, wirst du den folgenden Code nachprogrammieren, der ein etwas spezielles **Quadrat** fährt.

```
7 public class MindroidLVL2 extends LVL2API {
8     @Override
9     public void run() {
10         for (int i = 0; i < 3 && !isInterrupted(); ++i) {
11             int angle = 70;
12             forward();
13             delay(2000);
14             turnRight(angle);
15             forward();
16             delay(2000);
17             turnLeft(angle);
18             backward();
19             delay(2000);
20             turnRight(angle);
21             backward();
22             delay(2000);
23             turnLeft(angle);
24         }
25         stopMotors();
26     }
}
```

- Alle benötigten Methoden werden von der Klasse MindroidLVL2 bereitgestellt. Wir brauchen daher **keine zusätzlichen Imports**.
- In Zeile 10 bestimmt die **for-Schleife**, dass der Roboter das Quadrat dreimal abfahren soll.
- Der **forward()**-Befehl setzt den Roboter in Bewegung. Dabei fährt der Roboter solange, **bis er einen anderslautenden Befehl erhält** (wie bspw. stopMotors() in Zeile 25).
- Anschließend wird für 2 Sekunden **gewartet** (Zeile 13).
- Daraufhin vollführt der Roboter eine **Drehung nach Rechts** um (angeblich) 70°. Wir verwenden 70° statt 90°, da der Roboter aktuell noch etwas "überdreht".
- Nach der nächsten Geraden (Zeile 15) dreht der Roboter nach **links** (**turnLeft(angle)**, Zeile 17), um anschließend die verbleibenden beiden Geraden rückwärts zu fahren (**backward()**-Befehl, Zeile 18 und 21)

2.2: Abstand messen

Um dich mit dem **Ultraschall-Distanzsensor** vertraut zu machen, implementiere den folgenden Code nach. Er stellt einen einfachen **Parksensor** dar, der wie folgt funktioniert.

- Liegt die Distanz zu einem Objekt vor dem Roboter **unter 15cm**, dann blinkt die Status-LED schnell **rot** und es wird **“Oh oh :-O”** ausgegeben.
- Liegt die Distanz **zwischen 15cm und 30cm**, dann blinkt die Status-LED **gelb** und es wird **“Hm :-/”** ausgegeben.
- Liegt die Distanz **über 30cm**, dann leuchtet die Status-LED **grün** und es wird **“OK :-)”** ausgegeben..

```
1 package org.mindroid.android.app.statemachinesimpl;
2
3 import org.mindroid.api.ev3.EV3StatusLightColor;
4 import org.mindroid.api.ev3.EV3StatusLightInterval;
5 import org.mindroid.api.LVL2API;
6
7 public class MindroidLVL2 extends LVL2API {
8     @Override
9     public void run() {
10         String previousState = "";
11         clearDisplay();
12         drawString("Parking sensor", 1, 1);
13         while (!isInterrupted()) {
14             clearDisplay();
15             if (distanceLessThan(0.30f) && distanceGreaterThan(0.15f)) {
16                 drawString("Hm :-/", 1, 1);
17                 if (!previousState.equals("hm")) {
18                     setLED(EV3StatusLightColor.YELLOW,
19                         EV3StatusLightInterval.BLINKING);
20                 }
21                 previousState = "hm";
22             } else if (distanceLessThan(0.15f)) {
23                 drawString("Oh oh :-O", 1, 1);
24                 if (!previousState.equals("oh")) {
25                     setLED(EV3StatusLightColor.RED,
26                         EV3StatusLightInterval.DOUBLE_BLINKING);
27                 }
28                 previousState = "oh";
29             } else {
30                 drawString("OK :-)", 1, 1);
31                 if (!previousState.equals("ok")) {
32                     setLED(EV3StatusLightColor.GREEN,
33                         EV3StatusLightInterval.ON);
34                 }
35                 previousState = "ok";
36             }
37             delay(100);
38         }
39     }
40 }
```

- Wie in **Zeile 13** zu sehen ist, läuft das Programm in einer **Endlosschleife**, bis der “Stop”-Knopf in der App betätigt wird.

- Wir müssen uns jeweils den **vorherigen Zustand** in der Variablen **previousState** (Zeile 10) merken, da wir ansonsten all 100ms den Zustand der LED zurücksetzen würden, was zu keinem Blinken führt. Mithilfe von **previousState** ändern wir den LED-Modus nur dann, wenn wir müssen.

2.3: Farben messen

In dieser Aufgabe lernst du die Farbsensoren des Roboters kennen. Der folgende Quelltext liest kontinuierlich den aktuell gemessenen Farbwert des linken und rechten Lichtsensors aus (**getLeftColor()** bzw. **getRightColor()**, Zeilen 10-11).

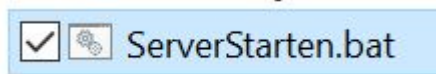
```
1 package org.mindroid.android.app.statemachinesimpl;
2
3 import org.mindroid.api.LVL2API;
4 import org.mindroid.impl.statemachine.properties.sensorproperties.Color;
5
6 public class MindroidLVL2 extends LVL2API {
7     @Override
8     public void run() {
9         while (!isInterrupted()) {
10             float leftColorValue = getLeftColor();
11             float rightColorValue = getRightColor();
12             clearDisplay();
13             drawString("Colors", 1, 1);
14             drawString("L: " + describeColor(leftColorValue), 1, 17);
15             drawString("R: " + describeColor(rightColorValue), 1, 33);
16             delay(100);
17         }
18     }
19
20     private static String describeColor(final float colorValue) {
21         if (colorValue == Color.NONE) return "None";
22         if (colorValue == Color.BLACK) return "Black";
23         if (colorValue == Color.BLUE) return "Blue";
24         if (colorValue == Color.GREEN) return "Green";
25         if (colorValue == Color.YELLOW) return "Yellow";
26         if (colorValue == Color.RED) return "Red";
27         if (colorValue == Color.WHITE) return "White";
28         if (colorValue == Color.BROWN) return "Brown";
29         return "unknown";
30     }
31 }
```

- Die Methode **describeColor** (Zeilen 20-30) zeigt, wie du den Rückgabewert in einen **lesbaren Text** umwandelst.
- In den Zeilen 13-15 siehst du, wie man auf dem Display **mehrzeiligen Text** ausgeben kann. Die Buchstaben haben jeweils eine **Höhe von 16 Pixeln**, sodass die zweite Zeile an der y-Position 17 und die dritte Zeile an der y-Position 33 beginnt.
- Um die **Qualität der Farbmessung** näher zu betrachten, haben wir für dich Farbtafeln mit allen sieben unterstützten Farben des EV3-Lichtsensors vorbereitet. Bei welchen Farben funktioniert die Erkennung gut, bei welchen eher weniger?
- Der Farbsensor kann auch zur **Erkennung von Abgründen** eingesetzt werden: Welche Farbwerte werden gemessen, wenn er Roboter auf der Tischplatte steht und wenn die Farbsensoren über den Tischrand ragen?

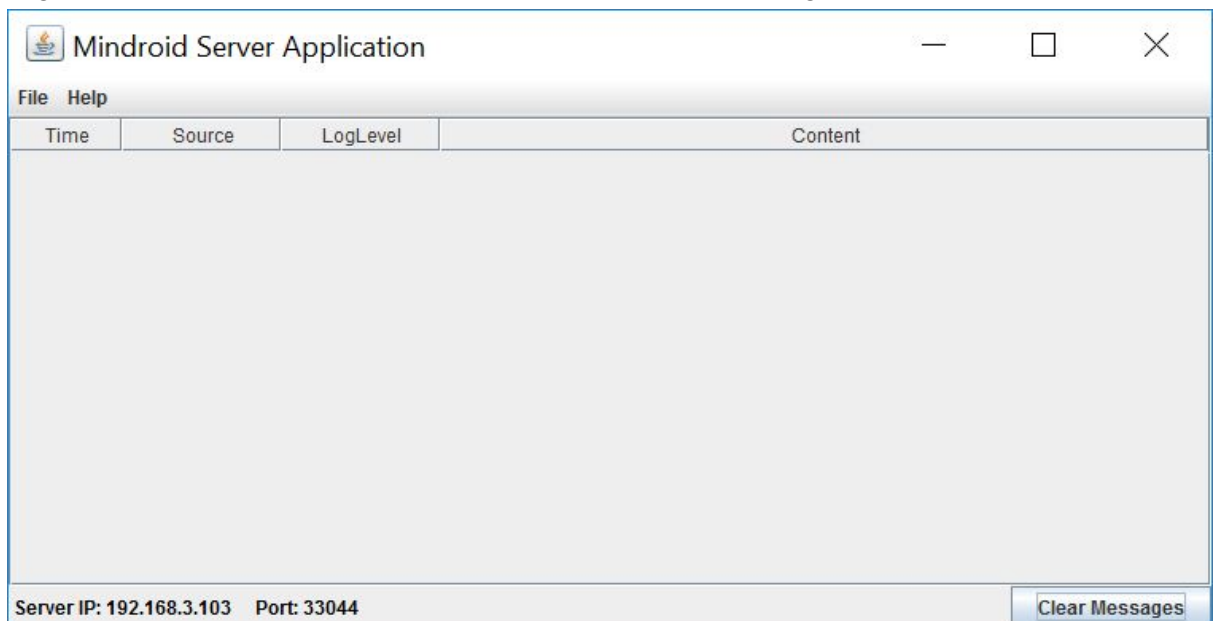
2.4: Verteiltes “Hallo Welt!”

In der vorherigen Aufgaben hast du kennengelernt, wie ein Programm auf einem einzelnen Roboter ausgeführt wird. Als nächstes wollen wir die **Roboter miteinander sprechen lassen**. Auch hier starten wir mit einem einfachen (diesmal verteilten) “Hallo Welt!”-Programm. Die Kommunikation wird über eine zentrale sogenannte **“Server”-Applikation** ablaufen, die auf eurem Entwicklungsrechner läuft und die Kommunikation zwischen den Robotern organisiert. Zum Einrichten müssen zunächst folgende Konfigurationsschritte durchgeführt werden:

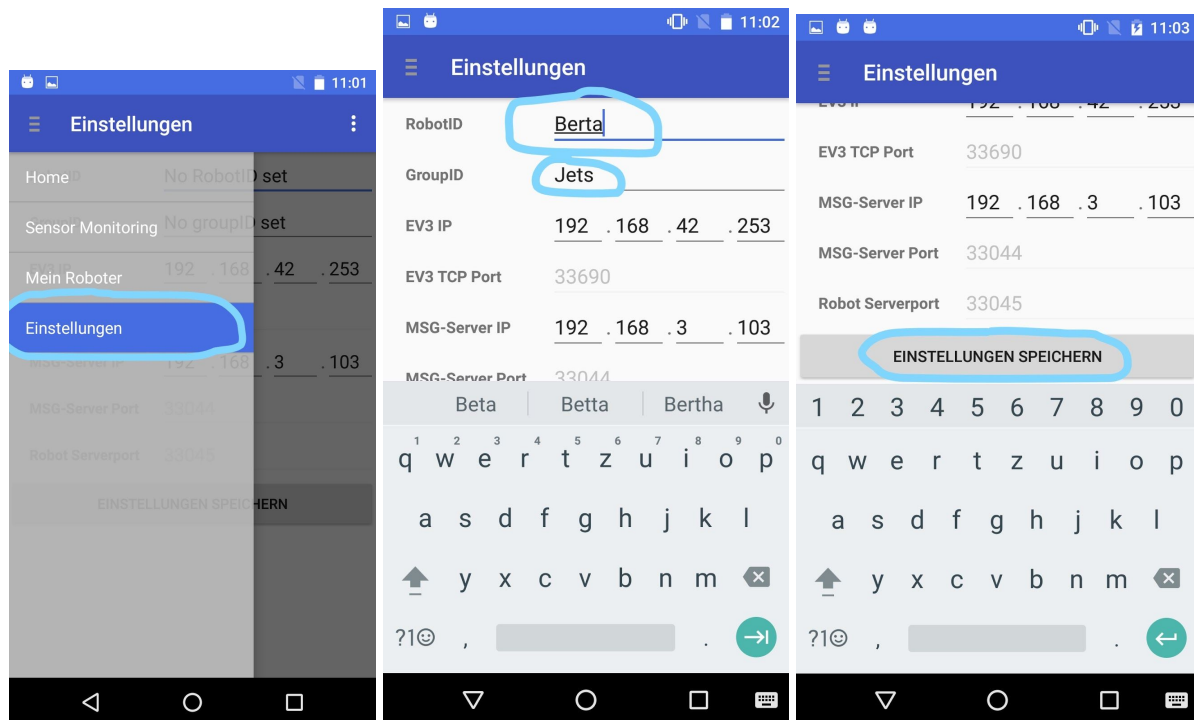
1. Stell zunächst sicher, dass **Handy** und **Computer** sich im gleichen **WLAN-Netz** befinden.
2. **Starte den Server**, indem du auf dem Desktop das folgende Skript laufen lässt:



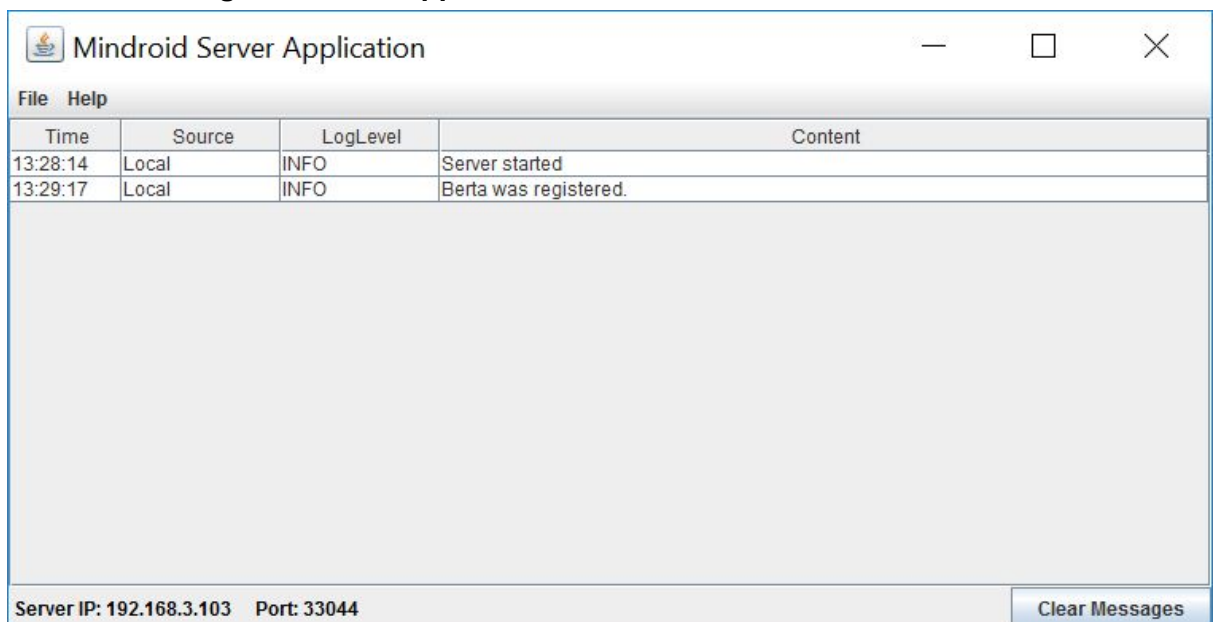
3. Nun öffnet das **Hauptfenster der Mindroid-Server-Applikation**. Im Moment sind noch keine Nachrichten angezeigt, da sich noch kein Roboter beim Server angemeldet hat. Dazu müssen als nächstes die Roboter konfiguriert werden.



4. Orientiere dich an folgenden Screenshots und gib deinem Roboter und der **Gruppe**, in dem sich der Roboter befindet, einen **Namen** deiner Wahl. Stelle sicher, dass sich alle beteiligten Roboter in der gleichen Gruppe befinden (also über die gleiche Gruppen-ID verfügen).



5. Nach dem **Neustart** der Mindroid-App meldet sich nun Berta beim Server an. Dass die Anmeldung erfolgreich war, erkennst du an einem neuen Eintrag im **Nachrichten-Log der Server-Applikation**.



6. Wiederhole nun den Schritt für einen weiteren Roboter und gib ihm einen Namen deiner Wahl (nachfolgend gehen wir davon aus, dass der zweite Roboter **“Robert”** heißt).

Über die Methode **“wasMessageReceived(String source, String message)”** kannst du prüfen, ob eine Nachricht mit gegebenen Inhalt (im String **message**) und vom angegebenen Versender (**source**) empfangen wurde. Mit der Methode **“sendMessage(String destination, String message)”** kannst du Nachrichten (String **message**) an einen anderen

Roboter (**destination**) verschicken. Nutze nun deine Kenntnisse aus den vorherigen Aufgaben und implementiere ein verteiltes "Hallo Welt!"-Programm:

1. **Berta** soll bei Programmstart sofort eine Nachricht an Robert schicken mit dem Inhalt "**Hallo Robert!**".
2. **Robert** soll beim Programmstart auf eine Nachricht von Berta mit dem Inhalt "Hallo Robert!" **warten**.
3. Sobald Robert die Nachricht von Berta erhalten hat, soll Robert auf seinem Display eine Nachricht über die Methode **drawString** mit dem Inhalt "**Nachricht von Berta erhalten!**" ausgeben.
4. Achte auf die Log-Ausgabe in der Server-Applikation, hier kannst du den Nachrichtenversand nachvollziehen.

Nächste Schritte

Du hast jetzt alle wesentlichen Fähigkeiten des Roboters kennengelernt. Nun ist es an der Zeit, dass du deine eigenen Projekte umsetzt. Die folgenden Aufgaben geben dir dazu Anregungen.

Falls du bestimmte Sensorwerte zur Kalibrierung nutzen willst, ist dir die Anwendung **Sensor Monitoring** bestimmt eine Hilfe.

1. Wähle dazu auf der Übersichtsseite des der Mindroid-App im **Dropdown** "**SensorMonitoring**" aus.
2. Betätige "**Start.**"
3. Öffne anschließend im **Menü** links oben den Eintrag "**Sensor Monitoring**".
4. Du siehst die aktuell gemessenen Werte der Sensoren
 - Der Wert "**Color ID**" beschreibt die aktuell gemessene Farbe unter dem linken Sensor. Ihre Werte sind in der Klasse *org.mindroid.impl.statemachine.properties.sensorproperties.Color* dokumentiert. Wenn du ein weißes Blatt Papier direkt unter den linken Farbsensor legst, sollte der Wert auf 6 (= Weiß) wechseln (manchmal wird auch 2 (= Blau) gemessen). Wenn du einen schwarzen Gegenstand direkt unter den Sensor legst, sollt der Wert auf 1 wechseln. Die Farbbestimmung mittels Color-ID ist leider relativ unpräzise.
 - Der Wert "**Distance**" gibt die aktuell vom Ultraschallsensor gemessene Distanz in Metern an. Bewege deine Hand vor dem Sensor vor und zurück und beobachte die Veränderung des Wertes.
 - Der Wert "**Angle**" gibt den aktuell gemessenen Winkel des Gyro-Sensors an. Drehe den Roboter um die Hochachse und beobachte die Veränderung des Wertes.

Nützliche Imports

- **import org.mindroid.api.LVL2API** für die Elternklasse
- EV3-Statuslicht:
 - Farbe: **import org.mindroid.api.ev3.EV3StatusLightColor;**
Werte: **EV3StatusLightColor.GREEN, .RED, .YELLOW, .OFF**
 - Blinkhäufigkeit: **import org.mindroid.api.ev3.EV3StatusLightInterval;**
Werte: **EV3StatusLightInterval.ON, .BLINKING, .DOUBLE_BLINKING**

Aufgabe 3: Wand-Ping-Pong

Nutze deine Kenntnisse, um den Roboter wie einen Ping-Pong-Ball **in gerader Linie** zwischen zwei Wänden/Gegenständen/... hin und her fahren zu lassen.

1. Der Roboter soll solange geradeaus fahren, bis er eine Wand erkennt. Er soll dabei **so nah wie möglich an die Wand heranfahren**, ohne mit ihr zu kollidieren.
2. Dann soll er ein kleines Stück rückwärts fahren und sich um 180° drehen
 - Beachte, dass **turnLeft/turnRight** leider im Moment nicht exakt arbeiten und du die Drehung ca. 25-40° früher abbrechen musst, da sonst der Roboter über das Ziel hinausschießt.
3. Beginne bei 1.

Bewertungskriterien:

- (1P) Ist die 180°-Drehung exakt?
- (1P) Vergleich: Wie nahe kommst du an die Wand heran?
- (1P) Vergleich: Wie schnell schaffst du eine Runde (Fahren → Wand+Drehen → Fahren)?

Aufgabe 4: Koordiniertes Wand-Ping-Pong

Diese Aufgabe lehnt sich an die vorherige an. Allerdings sollen sich nun zwei Roboter abstimmen. Beide Roboter starten nebeneinander und blicken in die gleiche Richtung.

1. **Roboter A** fährt solange, bis er eine Wand entdeckt. Er **setzt zurück, dreht sich um und bleibt stehen**.
2. Roboter A sendet Roboter B eine **“Start”-Nachricht**.
3. Daraufhin setzt sich **Roboter B** in Bewegung, bis er auf die **Wand** trifft. Daraufhin **setzt Roboter B zurück, wendet und bleibt stehen**.
4. Roboter B sendet Roboter A eine **“Weiter”-Nachricht**.
5. Beginne bei 1.

Bewertungskriterien:

- (1P) Roboter kollidieren nicht.
- (1P) Vergleich: Wie schnell schaffen die Roboter eine “Runde” (A: Fahren → Wand+Drehen+Stop; B: Warten → Fahren → Wand+Drehen+Stop)? Hier sollen die Roboter nicht möglichst nahe, sondern nahe genug an die Wand heranfahren (Abstand der Vorderachse kleiner 20cm).

Aufgabe 5: Mähroboter



(<https://i2.wp.com/www.traumgarten.nrw/wp-content/uploads/2015/04/Maehroboter-Garten.jpg>)

Nutze deine Kenntnisse, um den Roboter in einem mit schwarzem (oder weißem) Klebeband abgesperrten Bereich herumfahren zu lassen (so ähnlich wie beispielsweise viele Mähroboter arbeiten).

1. Wie beim Wand Ping-Pong soll der Roboter erstmal geradeaus fahren.
2. Wenn er eine Grenze erkennt, soll er zurücksetzen und sich eine neue Richtung aussuchen.
3. Beginne bei 1.

Tipp: Überprüfe, welche Color-IDs auf dem Boden und den Begrenzungen erkannt werden, um festzustellen, wann der Roboter an die Umzäunung gelangt ist.

Bewertungskriterien:

- (1P) Der umgrenzte Bereich darf **nicht verlassen** werden! Keines der Räder darf die Umgrenzung berühren!
- (1P) Vergleich: Wer schafft es **am schnellsten**, alle vier Seiten eines viereckigen Bereichs zu "treffen"?

Aufgabe 6: Platooning

In dieser Aufgabe geht es darum, zwei Roboter hintereinander her fahren zu lassen, ohne dass es einen Auffahrunfall gibt. Aktuell forschen zahlreiche Unis und Unternehmen unter dem Schlagwort **Platooning** an genau dieser Problemstellung bei echten LKWs und PKWs: Die Fahrzeuge fahren dabei so nahe, dass sie den Windschatten des Vorfahrenden ausnutzen können.



(<http://newsroom.scania.com/en-group/files/2012/04/11127-010.jpg>)

Roboter A und B werden hintereinander platziert, sodass sie in die gleiche Richtung blicken. Ziel ist es zunächst, dass Roboter B den **Abstand** zu Roboter A in einem bestimmten **Toleranzbereich** hält. Die Distanzangaben im Folgenden sind nur mögliche Werte - du bestimmst selbst, was geeignete Grenzwerte sind.

1. Roboter A fährt los. Sobald der Abstand zwischen Roboter A und Roboter B größer als 35cm wird, beginnt Roboter B aufzuschließen.
2. Wird der Abstand kleiner als 25cm, hält Roboter B die Geschwindigkeit von Roboter A.
3. Wird der Abstand kleiner als 15cm, lässt Roboter B sich zurückfallen (oder fährt sogar rückwärts).

Bewertungskriterien

- (1P) Roboter A und B fahren mind. 1 Meter hintereinander her ohne Kollisionen (weder mit anderen Robotern noch mit der Wand).
- (1P) Abstand bleibt in einem von euch zuvor bestimmten Bereich (Schnur), wenn beide Roboter vorwärts fahren.
- (1P) Vergleich: Geringstmöglicher Abstand, bei dem keine Kollision stattfindet.

Aufgabe 7: Verfolgung

Es gibt die Tanzfigur "Verfolgung" beim Cha-Cha-Cha. Dabei verfolgt jeweils ein Tanzpartner den anderen, bis beide sich umdrehen und die Rollen wechseln. Diese Figur ist tatsächlich nicht sehr weit vom Platooning-Beispiel aus [der vorherigen Aufgabe](#) entfernt.



(<https://youtu.be/OrxzuvHvOLc?t=1m11s>)

Der Ablauf soll dieses Mal wie folgt aussehen:

1. **Roboter A** übernimmt zunächst das Kommando und **fährt voraus**, während **Roboter B** einen möglichst **gleichbleibenden Abstand** hält.
2. **Roboter A** beschließt nach einer gewissen Zeit, dass nun die **Drehung** folgt. Er **stoppt** und sendet eine "**Drehen**"-Nachricht an Roboter B.
3. **Roboter B stoppt, wendet** und sendet Roboter A eine "**Gedreht**"-Nachricht.
4. Daraufhin **dreht Roboter A** ebenfalls um 180°.
5. Nun **tauschen** Roboter A und B die **Rollen**: Roboter B fährt voraus und gibt den Ton bis zur nächsten Drehung an.

Bewertungskriterien

- (1P) Einmal Verfolgung hin (Roboter A ist der Führende) und einmal Verfolgung zurück (Roboter B ist der Führende) und dabei keine Kollision!
- (1P) Es sollte klar erkennbar sein (bspw. per LED-Statusleuchte), wer aktuell der "Führende" ist.