

Mindroid Pilotworkshop am xx.06.2018



TECHNISCHE
UNIVERSITÄT
DARMSTADT

FG ES / MAKI
TU Darmstadt

Aufgabe 1 Hallo Welt

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

Aufgabe 1.1 Setup

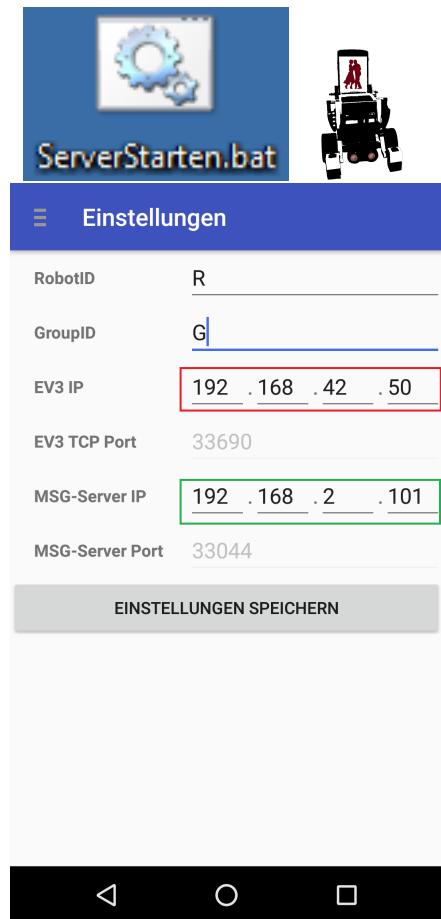
Damit alles funktioniert, musst du deine Geräte zuerst überprüfen und einrichten. Dabei lernst du, wie du die einzelnen Komponenten miteinander verbindest und neu erstellte Programme auf das Smartphone überträgst. Zuerst musst du dazu das Smartphone und den Roboter mit dem **USB-Kabel** verbinden. Überprüfe dabei ob

- Alle Kabel am Roboter ordentlich sitzen und
- Dein Smartphone entweder mit dem WLAN **MD005-2.4GHz** oder **MD0029-2.4GHz** verbunden ist.

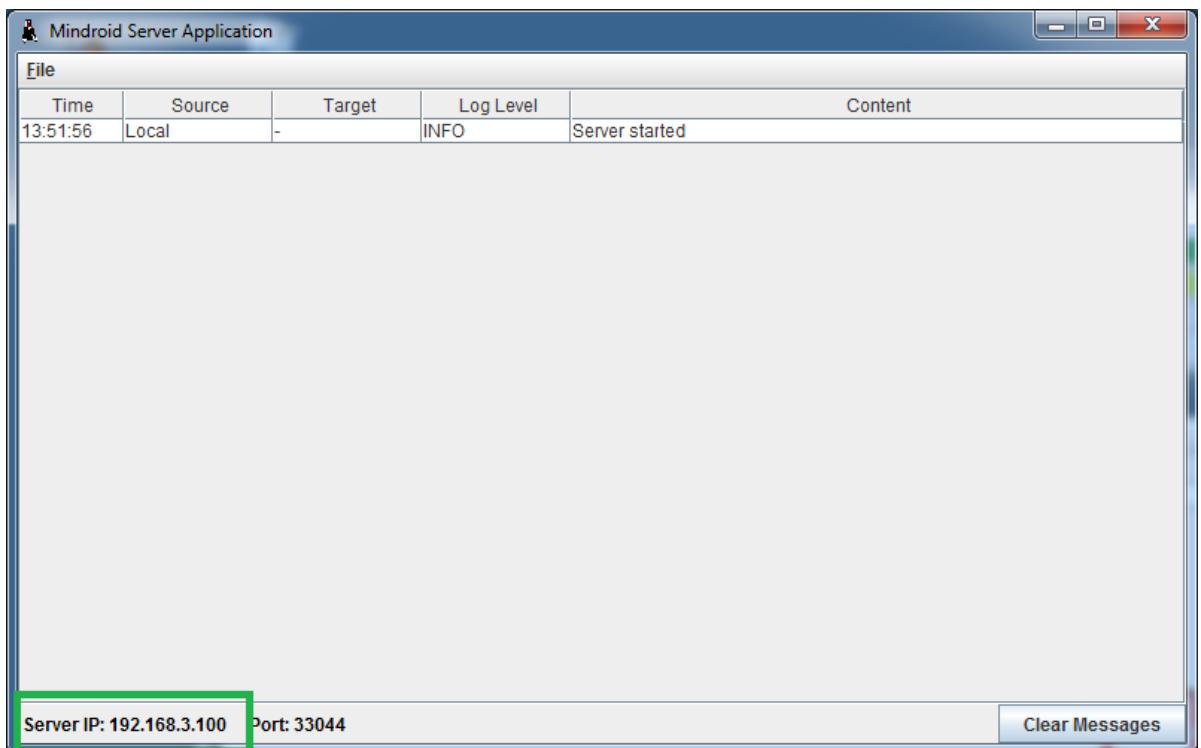
Aufgabe 1.1.1 App und PC verbinden

Um neue Programme einfach und schnell auf das Smartphone übertragen zu können, muss zuerst das Smartphone mit dem PC verbunden werden. Dazu sind folgende Schritte nötig.

1. Starte den **Message-Server**, indem du die **ServerStarten.bat** auf dem Desktop des Computers mit einem Doppelklick ausführst. Der Server erledigt ein paar Einstellungen. Außerdem ist er später wichtig, damit die Roboter untereinander kommunizieren können.
2. Um eine dauerhafte Verbindung herzustellen, müssen wir nun das Smartphone mit dem Server bekannt machen. Dazu öffnest du auf dem Smartphone die **Mindroid-App**.
3. Der Message-Server zeigt dir links unten in der Ecke seine IP-Adresse an. Diese musst du nun der Mindroid-App mitteilen. Dazu navigierst du über das Menü oben links in den **Einstellungen-Bildschirm**. Dort trägst du unter **MSG-Server IP** die IP des Servers ein. Ist dies erledigt, kannst du zurück zum Hauptbildschirm der App und auf **Verbinden** klicken. Die App verbindet sich nun mit dem Server.



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



Aufgabe 1.1.2 Programme übertragen

Neue Programme schreiben und übertragen wir in der Entwicklungsumgebung "Java-Editor".

1. Starte diesen indem du die Verknüpfung auf dem Desktop startest. In der Mitte findest du den Quelltext des aktuellen Programms, hinterlegt mit Farben, die die Orientierung erleichtern sollen.

```
Java-Editor - [C:\Users\mweck\Documents\repos\mindroid\impl\androidApp\app\src\main\java\org\mindroid\android\app\statemachinesimpl\MindroidLV2.java]
Datei Bearbeiten Start Test UML Tools Fenster Hilfe
Programm Swing 1 Swing 2 Utilities FX Base FX Controls FX Shapes
MindroidLV2.java
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 MindroidLV2 extends LVL2API {
12     @Override
13     public void run() {
14         brickController.drawString("Hello World", 1,1);
15     }
16 }
//
```

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

Zelle: 17 Spalte: 1 verändert Einf ANSI/Dos C:\Users\mweck\Documents\repos\mindroid\impl\androidApp\app\src\main\java\org\mindroid\android\app\statemachinesimpl\MindroidLV2.java gespeichert

Interpreter Compiler Debugger Suche Meldungen Interaktiv

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

2. Dazu klickst du auf den blauen Play-Button oben links (Der Button ändert seine Farbe nach dem Durchlauf auf Grün).
- Dein "Hallo Welt"-Programm wird jetzt an das Handy übertragen.
3. In der Konsolenausgabe am unteren Bildrand sollte 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

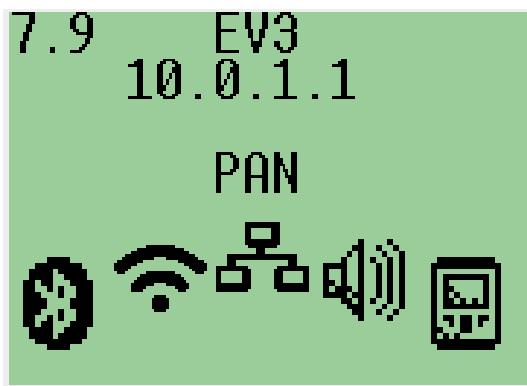
Total time: 35.483 secs
exit
```

Das Programm ist nun übertragen.

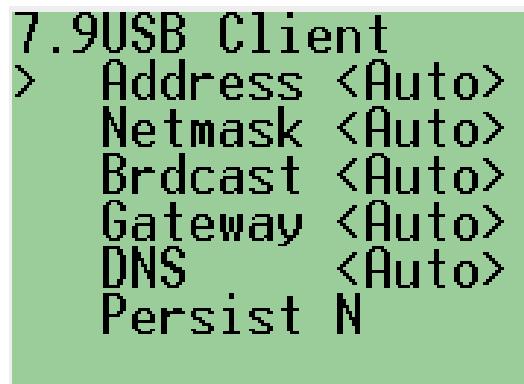
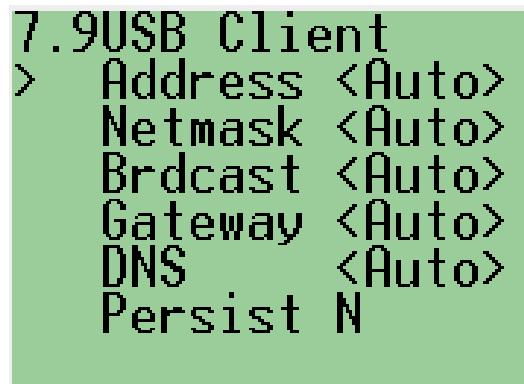
Aufgabe 1.1.3 Roboter und App verbinden

Zuletzt müssen nun noch Smartphone und Roboter miteinander verbunden werden. Dazu müssen zunächst ein paar Einstellungen am Roboter vorgenommen werden.

1. Dazu musst du zuerst in das **PAN-Menü** des Roboters navigieren. Wechsle mit den Links-/Rechts-Tasten bis du den Menüpunkt **PAN** siehst und betätige die **Auswahltaste**.



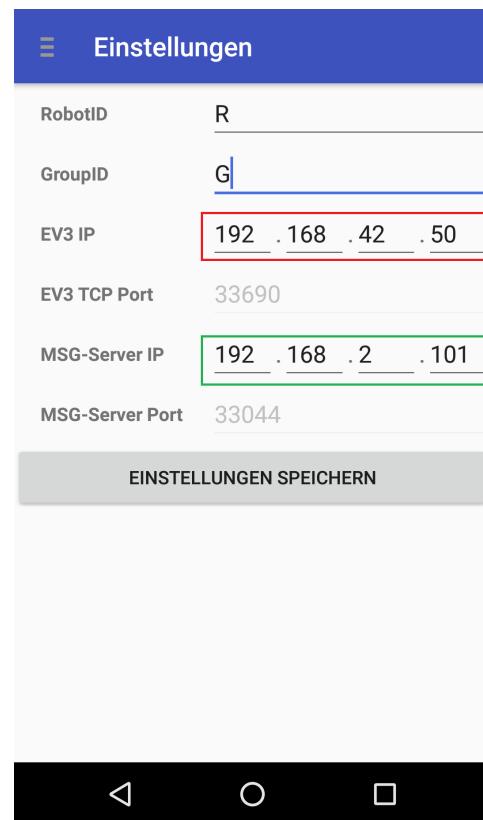
2. Nun navigierst du durch das Menü des Roboters wie auf den Bildern zu sehen und bestätigst jeweils mit der Auswahlaste: **PAN - USB-Client - Address - Automatic**



3. Mit der **Zurück**-Taste kommst du zurück in das Hauptmenü und die Einstellungen werden übernommen.



4. Zurück im Startmenü siehst du nun wie auf dem Bild zu sehen eine **IP-Adresse**. Um die Verbindung zwischen App und Roboter herstellen zu können, muss diese Adresse nun in den Einstellungen der App hinterlegt werden.



5. Nun kannst du das **EV3-Programm** starten. Wähle dazu **Run Default** im Hauptmenü des Roboters aus und bestätige mit der Auswahltaste.



6. Um nun das Hello-World-Programm zu starten, wähle es in der Liste der App aus und drücke auf **Starten**. Es wird ausgeführt, und auf dem Bildschirm des Roboters erscheint "Hallo Welt!"

Aufgabe 1.2 Der Quelltext erklärt

Nachdem du den Roboter erfolgreich eingerichtet und das erste mal getestet hast, gehen wir jetzt daran, uns den Quelltext näher anzusehen.

```

1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4 import org.mindroid.impl.brick.Textsize;
5
6 public class HelloWorld extends ImperativeWorkshopAPI {
7

```

```

8  public HelloWorld() {
9      super("Hello World");
10 }
11
12 @Override
13 public void run() {
14     clearDisplay();
15     drawString("Hello World", Textsize.MEDIUM, 10, 50);
16 }
17 }
```

Das Verhalten des Roboters befindet sich in der **run-Methode** (Zeilen 13-16). Wenn ein Mindroid-Programm ausgeführt wird, werden die Befehle in dieser Methode nacheinander abgearbeitet.

- **clearDisplay()** löscht den Display-Inhalt
- **drawString(text, textSize, xPosition, yPosition)** schreibt einen gegebenen Text(**text**) an die gegebenen Koordinaten (**xPosition**, **yPosition**) und verwendet dabei die definierte Schriftgröße (**textSize**).
- Der Konstruktor in den Zeilen 8 bis 10 gibt unserem Programm einen Namen (Zeile 8). Mit dem Aufruf von **super("Hello World")** bestimmen wir, unter welcher Bezeichnung unser Programm später ausgewählt werden kann. Es ist sinnvoll an beiden Stellen aussagekräftige Bezeichnungen zu verwenden.

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 **ImperativeWorkshopAPI** im Paket **org.mindroid.api**.

Aufgabe 1.3 Erste eigene Änderungen vornehmen

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.

Öffne dazu zuerst die **HelloDate.java**-Datei. Ändere den Code nun, sodass er dem folgenden Code ähnelt:

```

1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4 import org.mindroid.impl.brick.Textsize;
5
6 import java.text.SimpleDateFormat;
7 import java.util.Date;
8
9 public class HelloDate extends ImperativeWorkshopAPI {
10
11     public HelloDate() {
12         super("Hello Date");
13     }
14
15     @Override
```

```

16     public void run() {
17         SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyy");
18         clearDisplay();
19         drawString("Datum: " + formatter.format(new Date()), Textsize.SMALL, 1, 1);
20     }
21 }
```

- Die Imports von `java.text.SimpleDateFormat` und `java.util.Date` sind neu hinzugekommen.
- Außerdem gibt es nun eine Variable `formatter` (Zeile 17), die in der Lage ist, das aktuelle Datum formatiert auszugeben (`formatter.format(new Date())`, Zeile 19)).

Um die Änderungen nun auf das Handy zu übertragen, muss die App neu installiert werden.

1. Betätige jetzt die Schaltfläche “Starten” .
2. Nachdem im Compiler-Fenster die Meldung “BUILD SUCCESSFUL” erschienen ist, wird auf dem Handy die Mindroid-App automatisch geschlossen und wieder geöffnet.
3. Wähle “Verbindung zum EV3-Brick herstellen”
4. Wähle im Dropdown “HelloWorld” aus und betätige “Start”.
5. 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**.

Aufgabe 2.1 Fahren - die Antriebsmotoren

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 nach-programmieren, der ein etwas **spezielles Quadrat** fährt.

```
1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4
5
6 public class DriveSquare extends ImperativeWorkshopAPI {
7
8     public DriveSquare() {
9         super("Drive Square");
10    }
11
12    @Override
13    public void run() {
14        for (int i = 0; i<3 && !isInterrupted(); i++) {
15            int angle = 90;
16            forward();
17            delay(1000);
18            turnRight(angle);
19            forward();
20            delay(1000);
21            turnLeft(angle);
22            backward();
23            delay(1000);
24            turnRight(angle);
25            backward();
26            delay(1000);
27            turnLeft(angle);
28        } // end of for
29        stop();
30    }
31 }
```

- Alle benötigten Methoden werden von der Klasse **ImperativeWorkshopAPI** bereitgestellt. Wir brauchen daher keine zusätzlichen Imports.
- In Zeile 14 bestimmt die **for-Schleife**, dass der Roboter das Quadrat dreimal abfahren soll.
- Der **forward()**-Befehl (Zeile 16) setzt den Roboter in Bewegung. Dabei fährt der Roboter solange, bis er einen anderslautenden Befehl erhält (wie bspw. **stop()** in Zeile 29).
- Anschließend wird für 2 Sekunden **gewartet** (Zeile 16). In dieser Zeit fährt der Roboter vorwärts.
- Daraufhin vollführt der Roboter eine Drehung nach Rechts um 90° (**turnRight(angle)**, Zeile 18).
- Nach der nächsten Geraden (Zeile 19) dreht der Roboter nach links (**turnLeft(angle)**, Zeile 21), um anschließend die verbleibenden beiden Geraden rückwärts zu fahren (**backward()**-Befehl, Zeile 22 und 25).

Aufgabe 2.2 Der Ultraschallsensor - Abstand Messen

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

1. 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.
2. Liegt die Distanz **zwischen 15cm und 30cm**, dann blinkt die Status-LED gelb und es wird “**Hm :/-**” ausgegeben.
3. Liegt die Distanz **über 30cm**, dann leuchtet die Status-LED grün und es wird “**OK :-)**” ausgegeben.

```
1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4 import org.mindroid.api.ev3.EV3StatusLightColor;
5 import org.mindroid.api.ev3.EV3StatusLightInterval;
6 import org.mindroid.impl.brick.Textsize;
7
8 public class ParkingSensor extends ImperativeWorkshopAPI {
9
10    public ParkingSensor() {
11        super("Parking Sensor");
12    }
13
14    @Override
15    public void run() {
16        String previousState = "";
17        clearDisplay();
18        drawString("Parking sensor", Textsize.MEDIUM, 10, 10);
19        while (!isInterrupted()) {
20            clearDisplay();
21            if(getDistance() < 0.30f && getDistance() > 0.15f) {
22                drawString("Hm :/-", Textsize.MEDIUM, 10, 10);
23                if (!previousState.equals("hm")) {
24                    setLED(EV3StatusLightColor.YELLOW,
25                            EV3StatusLightInterval.BLINKING);
26                }
27                previousState = "hm";
28            } else if (getDistance() < 0.15f) {
29                drawString("Oh oh :-O", Textsize.MEDIUM, 10, 10);
30                if (!previousState.equals("oh")) {
31                    setLED(EV3StatusLightColor.RED,
32                            EV3StatusLightInterval.DOUBLE_BLINKING);
33                }
34                previousState = "oh";
35            } else {
36                drawString("OK :-)", Textsize.MEDIUM, 10, 10);
37                if (!previousState.equals("ok")) {
38                    setLED(EV3StatusLightColor.GREEN,
39                            EV3StatusLightInterval.ON);
39                }
40                previousState = "ok";
41            }
42        }
43    }
44}
```

```

42         }
43         delay(100);
44     }
45 }
46 }
```

- Um die LED ansteuern zu können, müssen wir die Pakete `org.mindroid.api.ev3.EV3StatusLight` und `org.mindroid.api.ev3.EV3StatusLightInterval` importieren.
- Wie in Zeile 19 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 17) merken, da wir ansonsten alle 100ms den Zustand der LED zurücksetzen würden, was das Blinken verhindert. Mithilfe von `previousState` ändern wir den LED-Modus nur dann, wenn wir müssen.

Aufgabe 2.3 Die Farbsensoren - Farbe 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()` in Zeilen 16 und 17).

```

1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4 import org.mindroid.impl.brick.Textsize;
5 import org.mindroid.impl.statemachine.properties.Colors;
6
7 public class ColourTest extends ImperativeWorkshopAPI {
8
9     public ColourTest() {
10         super("Colour Test");
11     }
12
13     @Override
14     public void run() {
15         while (!isInterrupted()) {
16             Colors leftColorValue = getLeftColor();
17             Colors rightColorValue = getRightColor();
18
19             clearDisplay();
20             drawString("Colors", Textsize.MEDIUM, 1, 1);
21             drawString("L: " + describeColor(leftColorValue), Textsize.MEDIUM, 1, 17);
22             drawString("R: " + describeColor(rightColorValue), Textsize.MEDIUM, 1, 33);
23             drawString("Distance: " + getDistance(), Textsize.MEDIUM, 1, 51);
24             delay(500);
25         }
26     }
27
28     private static String describeColor(final Colors colorValue) {
29         if (colorValue == Colors.NONE)    return "None";
30     }
31 }
```

```

30     if (colorValue == Colors.BLACK)    return "Black";
31     if (colorValue == Colors.BLUE)     return "Blue";
32     if (colorValue == Colors.GREEN)    return "Green";
33     if (colorValue == Colors.YELLOW)   return "Yellow";
34     if (colorValue == Colors.RED)      return "Red";
35     if (colorValue == Colors.WHITE)    return "White";
36     if (colorValue == Colors.BROWN)   return "Brown";
37     return "unknown";
38 }
39 }
```

- Die Methode **describeColor** (Zeilen 29-39) zeigt, wie du den Rückgabewert in einen lesbaren Text umwandelst.
- In den Zeilen 20-21 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 der Roboter auf der Tischplatte steht und wenn die Farbsensoren über den Tischrand ragen?

Aufgabe 2.4 Kommunikation zwischen Robotern

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 läuft über das bereits vorgestellten “Server”-Programm, welches ihr vorhin schon auf dem Entwicklungsrechner gestartet habt. Damit die Roboter voneinander unterschieden werden können, benötigt jeder einen eigenen Namen. Um diese Einstellungen ändern zu können, müsst ihr die Verbindung zum Server erst einmal trennen. Navigiert nun wieder in das Einstellungs-Menü der App und gebt den Robotern Namen. Stellt sicher, dass die Roboter auch in Gruppen eingeteilt sind. Die Screenshots zeigen euch wie das geht.

Screenshots für Namen einstellen, Gruppe einstellen

Wiederhole diesen Schritt nun auch für den zweiten Roboter. In unserem Beispiel gehen wir davon aus, die Roboter heißen Robert und Berta.

Wir möchten nun, dass Berta eine Nachricht mit dem Inhalt **“Hallo Robert!”** an den Nachrichtenserver versendet. Robert soll diese Nachricht empfangen und die Nachricht auf seinem Display ausgeben. Dazu sind zwei unterschiedliche Programme für Robert und Berta notwendig.

```

1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4
5 public class HelloWorldPingB extends ImperativeWorkshopAPI {
```

```

7  public HelloWorldPingB() {
8      super("Hello World Ping Berta");
9  }
10
11     @Override
12     public void run() {
13         clearDisplay();
14         sendMessage("Robert", "Hallo Robert!");
15     }
16 }
```

- Bei Programmstart sendet Berta in Zeile 14 eine Nachricht an **Robert** mit den Inhalt “**Hallo Robert!**”

```

1 package org.mindroid.android.app.workshopSolutions;
2
3 import org.mindroid.api.ImperativeWorkshopAPI;
4 import org.mindroid.impl.brick.Textsize;
5
6 public class HelloWorldPingR extends ImperativeWorkshopAPI {
7
8     public HelloWorldPingR() {
9         super("Hello World Ping Robert");
10    }
11
12    @Override
13    public void run() {
14        clearDisplay();
15        while(!isInterrupted()){
16            if (hasMessage()){
17                String msg = getNextMessage().getContent();
18                if (msg.equals("Hallo Robert")){
19                    drawString("Nachricht von Berta erhalten", Textsize.MEDIUM, 1, 60);
20                }
21            }
22            delay(100);
23        };
24    }
25 }
```

- Robert überprüft mit **hasMessage()** (Zeile 16) ob neue Nachrichten auf dem Message-Server vorhanden sind.
- Sobald eine Nachricht vorliegt, wird der Inhalt der Nachricht in die Variable **msg** gespeichert (Zeile 16).
- die Nachricht wird nun mit dem String “**Hallo Robert!**” verglichen². Stimmen beide überein, schreibt Robert auf sein Display einen Text (Zeile 19).

² Beachte: Strings werden in Java nicht mit == verglichen, sondern mittels der **equals()**-Methode

Aufgabe 2.5 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, sollte 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.

Aufgabe 2.6 Nützliche Imports

- `import org.mindroid.api.ImperativeWorkshopAPI` 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. Anweisungen:

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
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. Anweisungen:

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



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 Vorausfahrenden ausnutzen können.

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



Beim Cha-Cha-Cha gibt es die Tanzfigur "Verfolgung". Dabei verfolgt jeweils ein Tanzpartner den anderen, bis beide sich umdrehen und die Rollen wechseln. Diese Figur ist tatsächlich nicht sehr weit vom Plaftooning-Beispiel aus der vorherigen Aufgabe entfernt. 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.

A Troubleshooting

A.1 Installation über WLAN funktioniert nicht

Im Message-Server über **File->Connected Devices** schauen ob alle Smartphones in der Liste auftauchen und der ADB-state auf **connected** steht. Ist dies nicht der Fall, tippe in der App auf **TRENnen** und stelle die Verbindung danach erneut her. Falls das nicht klappt, kontaktiere einen Betreuer. Falls das Installieren per WLAN gar nicht mehr funktioniert, kann jederzeit eine USB-Verbindung zwischen Smartphone und PC hergestellt werden und darüber die App installiert werden.

B Sensorbelegung

B.1 Zeigt die standardmäßige Sensorbelegung, wie sie in der App unter “Mein Roboter” definiert sein muss.

Anschluss	Sensortyp	Modus
1	Farbe	ColorID
2	Ultraschall	Distance
3	Gyroskop	Angle
4	Farbe	ColorID

Tabelle B.1: Sensorbelegung

B.2 Zeigt den standardmäßigen Motoranschluss, wie er in der App unter “Mein Roboter” definiert sein muss.

Anschluss	Motor
A	Large Regulated Motor
B	-
C	-
D	Large Regulated Motor

Tabelle B.2: Motorbelegung