Mindroid Workshop Aufgabenstellung

NeXT Generation on Campus
Computational Engineering
Informationssystemtechnik
TU Darmstadt

Aufgabe 1 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.

Machen die Kriterien Sinn? Überhaupt bewertung machen?

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 2 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 3 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 4 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 bestimmen 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 5 Dancing Robots

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 Platooning-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.

Aufgabe 6 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 6.1 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 6.2 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`**

A EV3 Tasten

Abbildung A.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.

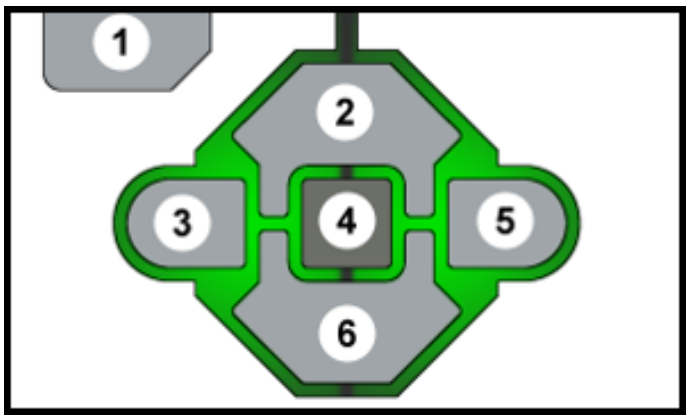


Abbildung A.1: EV3-Tastenbelegung[1]

Die Bedeutung der Tasten kannst du der folgenden Aufzählung entnehmen.

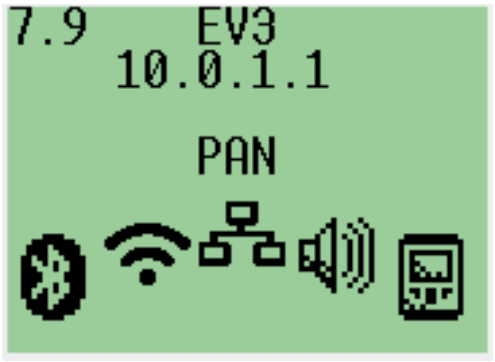
1. **Escape / Zurück**
2. **Up / Hoch**
3. **Left / Links**
4. **Enter / Bestätigen**
5. **Right / Rechts**
6. **Down / Unten**

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

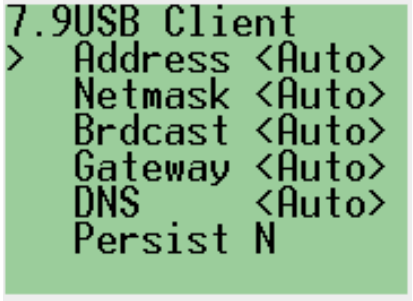
B PAN Einrichtung

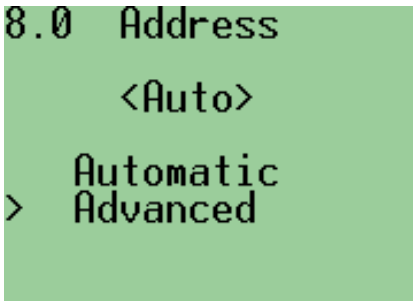
Wird im Hauptmenü noch nicht die richtige IP-Adresse angezeigt, müssen zuerst die PAN[2]-Einstellungen korrigiert werden.

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**.

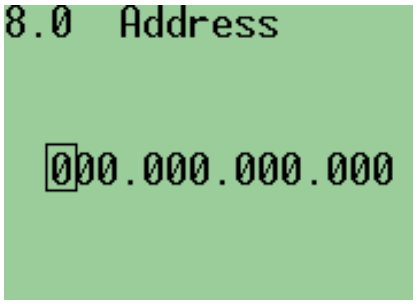


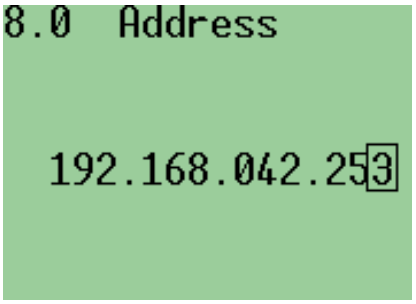
2. Nun navigierst du durch das Menü des Roboters wie auf den Bildern zu sehen und bestätigst jeweils mit der Auswahltaste: **USB-Client - Address - Advanced**

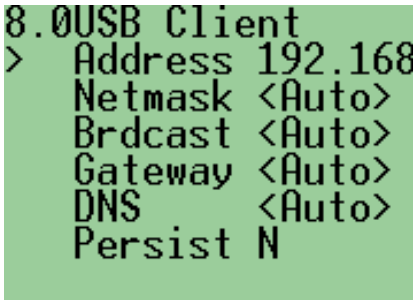




Nun musst du die IP Adresse 192.168.42.253 einstellen. Dazu navigierst du mit den rechts-/links-Tasten zu den einzelnen Ziffern und änderst deren Wert mit den oben-/unten-Tasten. Orientiere dich an den Bildern! Am Ende bestätigst du wieder mit der Enter-Taste.







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

Nun kannst du wieder zu Punkt 3 auf Seite **??** wechseln.

[2] PAN = Personal Area Network

C Troubleshooting

C.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.

D Sensorbelegung

Tabelle D.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 D.1: Sensorbelegung

Tabelle D.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 D.2: Motorbelegung

E Wichtige Funktionen

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

E.1 Fahren

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.

E.2 Sensoren

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

E.3 Kommunikation

Typ	Methode und Beschreibung
boolean	hasMessage() Prüft ob Nachricht vorhanden ist
MindroidMessage	getNextMessage() Ruft nächste Nachricht ab
void	broadcastMessage(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.

E.4 Brick

E.4.1 Display

Typ	Methode und Beschreibung
void	clearDisplay() Löscht den Aktuellen Inhalt des Displays
void	drawString(String text, Textsize textsize, int xPosition, int yPosition) Schreibt den im *text*-Parameter gegebenen Text auf das Display an die durch *xPosition* und *yPosition* definerte Stelle (siehe Abb. E.1) mit Textgröße *textsize*

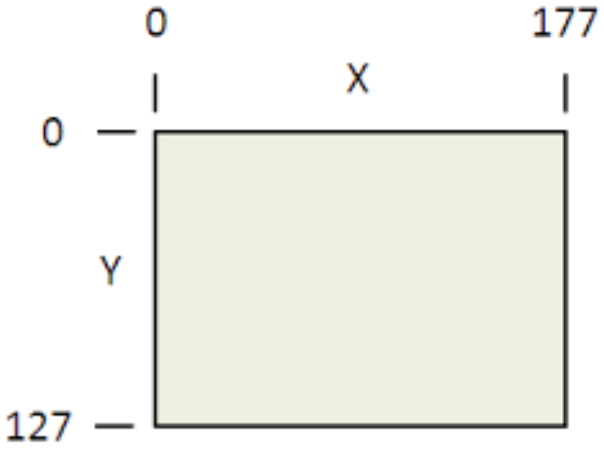


Abbildung E.1: Koordinaten der Pixel des Displays des EV3[3]

E.4.2 Buttons

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 A.1 auf Seite 5 entnehmen

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

E.4.3 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.

E.4.4 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 E.1

Tabelle E.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