

Navigation des mobilen Roboters

Sphero

STUDIENARBEIT

für die Prüfung zum
Bachelor of Engineering
des Studienganges Informationstechnik
an der
Dualen Hochschule Baden-Württemberg Karlsruhe

von

Irtaza Syed

11.05.2015

Matrikelnummer	6519774
Kurs	TINF12B3
Ausbildungsfirma	SEW-Eurodrive GmbH & Co. KG, Bruchsal
Betreuer	Prof. H.-J. Haubner

Erklärung

gemäß § 5 (2) der „Studien- und Prüfungsordnung DHBW Technik“ vom 18. Mai 2009.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

Inhaltsverzeichnis	3
Abbildungerverzeichnis.....	5
Abkürzungsverzeichnis.....	7
1. Einleitung	8
1.1 Aufgabenstellung und Ziel der Arbeit	8
1.2 Geplantes Vorgehen	8
2. Robotik	9
2.1 Roboter	9
2.2 Klassifikation von Robotern	10
2.2.1 Industrieroboter	10
2.2.2 Serviceroboter.....	10
2.2.3 Humanoider Roboter.....	11
2.2.4 Mobile Roboter	12
3. Sphero	18
3.1 Hardware.....	18
3.2 Software.....	20
3.2.1 Sphero API und Firmware	20
3.2.2 Sphero SDK und ADT	23
4. Entwurf und Implementierung von Drive & Draw.....	25
4.1 Designentwurf	25
4.2 Algorithmus	26
4.3 Klassenstruktur.....	28
4.3.1 Die Klasse Drive.....	28
4.3.2 Die Klasse Draw.....	28
4.3.3 Die Klasse LocationData	29
4.3.4 Die Klasse CollisionData	30
4.3.5 MainActivity	31
4.3.6 Zusatzfunktionen	32
5. Ergebnis	34
5.1 Startbildschirm.....	34
5.2 MainActivity	35

5.3 Zeichnung	36
5.4 Darstellung von Kollisionen.....	37
5.5 SettingsActivity	38
5.6 Einstellungsoption: Strichstärke.....	39
5.7 Einstellungsoption: Farbauswahl.....	40
5.8 Einstellungsoption: Kollision Sensitivität	41
6. Ausblick	42
7. Fazit	43
Literaturverzeichnis	44

Abbildungverzeichnis

Abb. 1: Industrieroboter beim Schweißarbeiten an einer Autokarosserie (KUKA AG [9]).....	10
Abb. 2: Autonomer Saugroboter (iRobot Roomba 581) [10].....	11
Abb. 3: Humanoider Roboter Nao (Aldebaran Robotics) [11].....	11
Abb. 4: Mobiler Roboter: Mars-Rover Curiosity (NASA) [12]	12
Abb. 5: Kontrollplan eines mobilen Roboters.....	13
Abb. 6: Arrangement of the legs of various animals	14
Abb. 7: The four basic wheel types.....	15
Abb. 8: Sphero und die Firmen eigene Controller-App [5].....	18
Abb. 9: Sphero Hardware [5]	19
Abb. 10: Kollisionserkennung mittels Schwellenwert.....	21
Abb. 11: Bewegungsmöglichkeiten des Spheros in 3D-Umfeld [5].....	22
Abb. 12: Eclipse IDE mit ADT	23
Abb. 13: Importieren von Sphero SDK Bibliotheken	24
Abb. 14: Designentwurf für Drive & Draw	25
Abb. 15: Algorithmus der App Drive & Draw	26
Abb. 16: Die Klasse Drive	28
Abb. 17: Die Klasse Draw	28
Abb. 18: Die Klasse LocationData	29
Abb. 19: Die Klasse CollisionData	30
Abb. 20: Klassenstruktur für Drive & Draw	31
Abb. 21: SettingsActivity	32
Abb. 22: ColorPickerActivity für die Farbauswahl	33
Abb. 23: Startbildschirm der App Drive & Draw	34
Abb. 24: Zeichenfläche mit Bedienelementen	35
Abb. 25: Mit Sphero erstellte Zeichnung.....	36

Navigation des mobilen Roboters Sphero

Abb. 26: Darstellung von Kollisionen	37
Abb. 27: SettingsActivity mit Einstellungen.....	38
Abb. 28: Einstellungsänderung für die Strichstärke	39
Abb. 29: Farbauswahl in der ColorPickerActivity	40
Abb. 30: Schwellwertänderung für die Kollisionserkennung	41

Abkürzungsverzeichnis

A

ADT	Android Developer Tools
API	Application Programming Interface
Abb.	Abbildung
App	Application

C

CCD	Charged-coupled Device
-----	------------------------

D

dp	Density-independent pixel
----	---------------------------

G

GPS	Global Positioning System
-----	---------------------------

I

iOS	iPhone oder iPad Operatating System
-----	-------------------------------------

L

LED	Light-emitting Diode
LiPo	Lithium-Polymer-Akkumulator

M

mAH	Mega-Amperstunde
MHz	Megahertz

S

SDK	Software Development Kit
-----	--------------------------

1. Einleitung

Automaten sind bereits seit der Antike bekannt. Schon damals wurde mit mechanischen Vorrichtungen experimentiert und Pläne für Automaten entwickelt, die heute noch eine wichtige Bedeutung in der modernen Robotik haben. Diese Arbeit befasst sich mit einigen wichtigen Aspekten der modernen Robotik.

1.1 Aufgabenstellung und Ziel der Arbeit

Sphero ist ein kugelförmiger Roboter von der Firma Orbotix, der mit Android und iOS Geräten über Bluetooth gesteuert wird. Orbotix hat mehrere SDKs für verschiedene Plattformen für Sphero bereitgestellt, mit denen Entwickler den Roboter selbst programmieren können.

Im Rahmen dieser Arbeit soll eine Android App entwickelt werden, die die Navigation des Roboters Sphero ermöglicht. Des Weiteren soll die App mithilfe von Sphero eine zweidimensionale Umrisskarte von einer begrenzten Fläche erstellen und auf dem Bildschirm des verwendeten Geräts (Smartphone/Tablet) anzeigen.

1.2 Geplantes Vorgehen

Das geplante Vorgehen für die Projektarbeit ist wie folgt:

- Einarbeitung in Robotik.
- Einarbeitung Sphero (Hardware, Software).
- Entwicklungsumgebung Eclipse, Android SDK und Spheros eigene SDK kennenlernen.
- Beispiel-Apps entwickeln und analysieren.
- Android App zur Navigation des Roboters Sphero implementieren.
- Funktion zur Kartenerstellung und Visualisierung implementieren.

2. Robotik

Robotik ist eine wissenschaftliche Disziplin, die sich mit der Realisierung und Anwendung von Robotersystemen beschäftigt. In diesem Kapitel werden verschiedene Robotersysteme und einige wichtige Grundlagen für die Entwicklung eines Roboters vorgestellt.

2.1 Roboter

„Roboter sind sensomotorische Maschinen zur Erweiterung der menschlichen Handlungsfähigkeit. Sie bestehen aus mechatronischen Komponenten, Sensoren und rechnerbasierten Kontroll- und Steuerelementen. Die Komplexität eines Roboters unterscheidet sich deutlich von anderen Maschinen durch die größere Anzahl von Freiheitsgraden und die Vielfalt und den Umfang seiner Verhaltensformen.“¹

Aus diesen Definitionen, kann zusammenfassend ein Roboter folgendermaßen beschrieben werden:

Ein Roboter

- ist eine Maschine bzw. ein Automat, der sich bewegen kann.
- ist programmier- und steuerbar.
- wird für Aufgaben in einer bestimmten Umgebung konstruiert.
- hebt sich durch seine Komplexität von gewöhnlichen Maschinen ab.
- dient dazu, mit seinen Fähigkeiten Probleme des Menschen zu lösen.

¹ Thomas Christaller 2001 [3]

2.2 Klassifikation von Robotern²

2.2.1 Industrieroboter

Industrieroboter sind programmierbare Maschinen, die zur Handhabung, Montage und Bearbeitung von Werkstücken dienen. Sie werden meist mit Werkzeugen ausgestattet und für bestimmte Einsatzgebiete in der Industrie festgelegt.

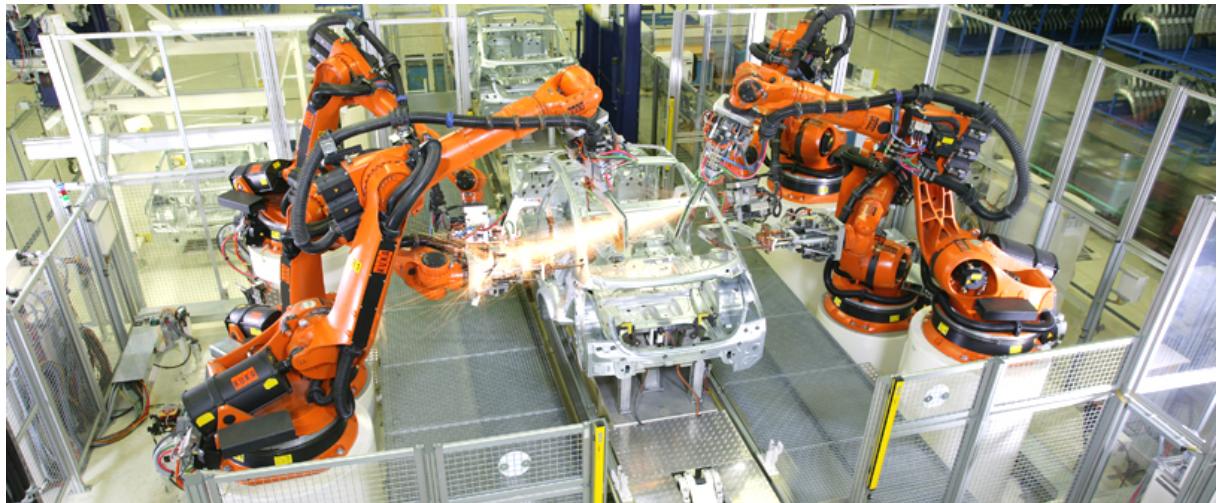


Abb. 1: Industrieroboter beim Schweißarbeiten an einer Autokarosserie (KUKA AG [9])

Die Industrieroboter werden nach Art und Anordnung der Bewegungssachsen unterteilt.

2.2.2 Serviceroboter

Serviceroboter erbringen Dienstleistungen für den Menschen. Sie sind mit Sensoren zur Erfassung der Umwelt und mit Navigationseinrichtungen ausgestattet, damit sie sich autonom in unstrukturierter Umgebung bewegen können. Serviceroboter werden an verschiedenen Anwendungsbereichen (wie in der Landwirtschaft, Medizin, Reinigung usw.) eingesetzt. Für den privaten Bereich werden auch immer häufiger kostengünstige Serviceroboter angeboten (z.B. zum Rasenmähen, Staubsaugen, Transport im Haushalt usw.).

² Vgl. [2], S.17



Abb. 2: Autonomer Saugroboter (iRobot Roomba 581) [10]

2.2.3 Humanoider Roboter

Humanoider Roboter sind dem Menschen nachempfunden. Sie haben vergleichbare kognitive, sensorische und motorische Fähigkeiten. Damit sollen diese mit dem Menschen direkt kommunizieren und interagieren können. Humanoider Roboter sind noch Gegenstand der Forschung. Sie sollen jedoch in einigen Jahren Marktreife erlangen. Zurzeit sind einige Roboter (siehe Abb. 3) bereits in der Lage bestimmte Tätigkeiten aus dem Alltag der Menschen durchführen zu können, wie z.B. Laufen, Tanzen, Musikinstrumente spielen, Lasten tragen usw.

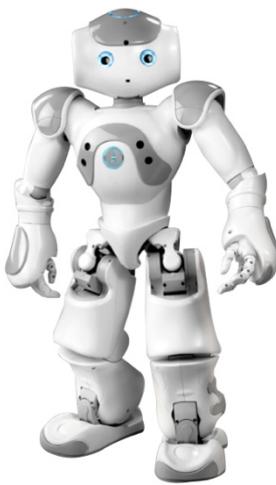


Abb. 3: Humanoider Roboter Nao (Aldebaran Robotics) [11]

2.2.4 Mobile Roboter

Mobile Roboter können sich frei in einer bestimmten Umgebung bewegen. Sie müssen mit Sensoren ausgestattet sein, die ihnen das Erfassen und Verarbeiten der Umgebung ermöglichen. In der Abb. 4 ist der mobile Roboter Mars-Rover Curiosity zu sehen. Der Roboter ist in der Lage sich auf unebenem Terrain zu bewegen und wird eingesetzt, um den Planeten Mars zu erforschen.



Abb. 4: Mobiler Roboter: Mars-Rover Curiosity (NASA) [12]

Mobile Roboter unterscheiden sich stark in Konstruktion und Aufbau durch ihre unterschiedlichen Einsatzgebiete. So sind z.B. Serviceroboter auch mobile Roboter, da diese die Eigenschaften eines mobilen Roboters erfüllen. Eine weitere mobile Roboterart, ist der Spielzeugroboter. Spielzeugroboter sind Kleinroboter, deren Einsatzgebiet im nichtkommerziellen Hobby- und Spielbereich liegt. Ein Beispiel für einen Spielzeugroboter, ist der für diese Studienarbeit verwendete Kleinroboter Sphero. Im Kapitel 3 wird Spheros Aufbau genauer vorgestellt. Vorerst müssen jedoch einige wichtige Aspekte der Entwicklung eines mobilen Roboters verstanden werden.

Das Diagramm in der Abb. 5 zeigt den Kontrollplan eines mobilen Roboters. Der mobile Roboter erfasst als erstes die Daten von seiner Umgebung mit Sensoren, mit denen der Roboter ausgestattet ist. Die Daten werden verarbeitet und für die nächsten Schritte genutzt. Der Roboter könnte z.B. aus den erfassten Daten seine Position feststellen und eine sichere Bahn, ohne Hindernisse planen. Dies geschieht in der Lokalisierungsphase. Nachdem der Roboter die Daten verarbeitet hat, kann er die Bewegung ausführen und die geplante Bahn fahren.

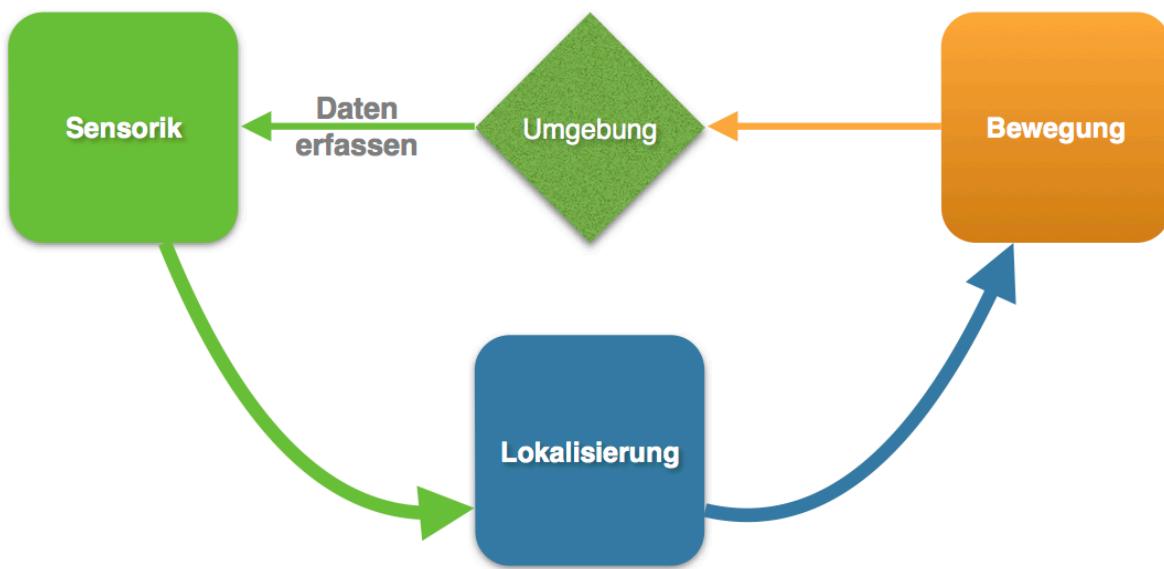


Abb. 5: Kontrollplan eines mobilen Roboters

Im Folgenden werden die im Kontrollplan dargestellten Schritte (Abb. 5) genauer erläutert:

Bewegung (engl. Locomotion):

Ein wichtiger Aspekt bei der Entwicklung eines mobilen Roboters ist die Wahl einer Bewegungsart. Es gibt mehrere unterschiedliche Bewegungsarten die in zwei Hauptkategorien eingeteilt werden:

- **Laufen:** Das Bewegen mit Hilfe von Beinen (engl. **legged mobile robot**).
- **Rollen:** Das Bewegen mit Hilfe von Rädern (engl. **wheeled mobile robot**).

Ein **legged mobile robot** hat i.d.R. 2 bis 6 Beine. Roboter mit nur einem Bein sind sehr instabil und das statische Gehen ist unmöglich. Um statisches Gehen zu ermöglichen, muss ein Roboter mindestens 6 Beine haben. Dies wird anhand eines Beispiels aus der Natur verdeutlicht (siehe Abb. 6).

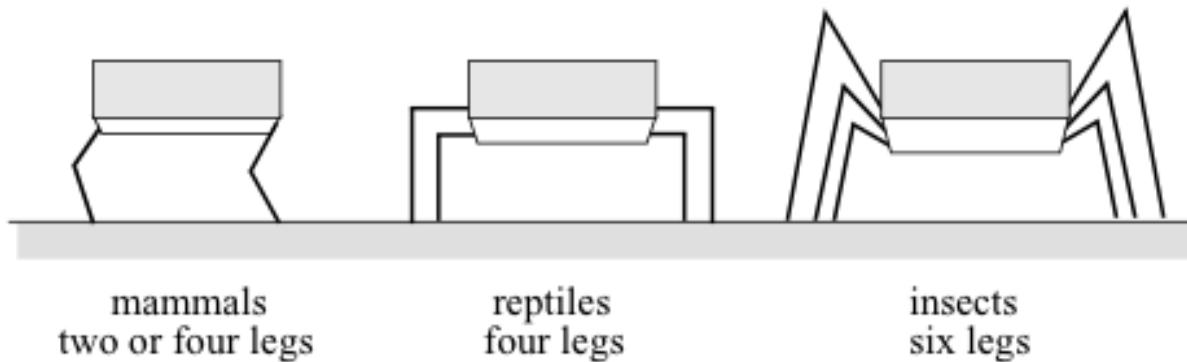


Abb. 6: Arrangement of the legs of various animals³

In der Abb. 6 ist die Anordnung der Beine von drei Tierarten (Säugetiere, Reptilien und Insekten) dargestellt.

Insekten und Spinnen können bereits nach der Geburt laufen. Für sie ist das balancieren beim laufen auf sechs Beinen (bzw. acht Beine bei Spinnen) relativ einfach. Neugeborene Säugetiere mit vier Beinen können nach einigen Versuchen zwar auf vier Beinen stehen, jedoch brauchen diese etwas Zeit, um statisch Gehen zu können. Für Säuglinge mit zwei Beinen ist bereits das Stehen eine Herausforderung. Kleinkinder zum Beispiel, brauchen mehrere Monate, um das Stehen und Gehen zu erlernen, da unsere Beine mit zwei Berührungs punkten zum Boden eine sehr geringe Stabilität, im Gegensatz zu vier oder sechs Beinen bieten.

Das Bewegen mit Rädern ist der am meisten verwendete Bewegungsmechanismus bei mobilen Robotern. Ein **wheeled mobile robot** ist effizient und die Realisierung dieses Bewegungsmechanismus ist relativ einfach. Hierbei ist die Balance nicht mehr das entscheidende Kriterium, da alle Räder, zu jedem Zeitpunkt Kontakt zum Boden haben. Stattdessen sind die Bedienung, die Reibung und die Stabilität die wichtigeren Faktoren bei der Entwicklung eines wheeled mobile robots. Dabei wird untersucht, ob die Räder des Roboters das gewünschte Terrain befahren können und eine suffiziente Bedienung der Geschwindigkeit des Roboters bieten.

³ [1], S. 18, Figure 2.5

Es gibt vier Kategorien von Rädern für mobile Roboter. Diese sind in der Abb. 7 dargestellt:

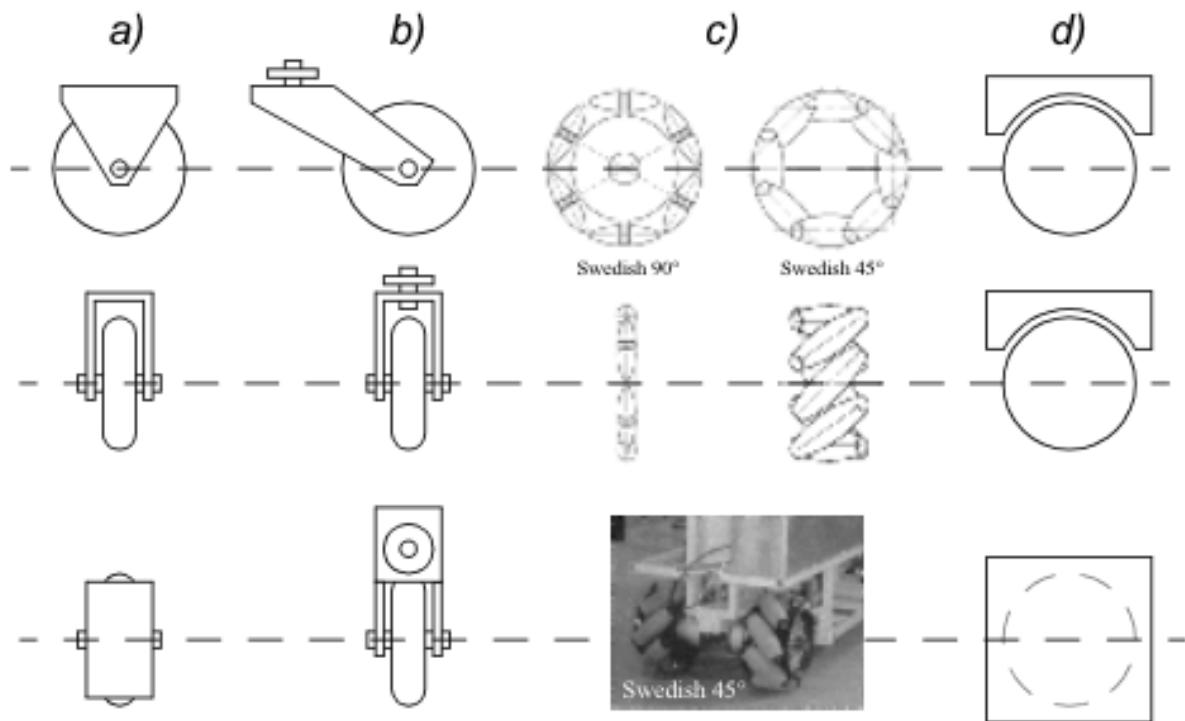


Abb. 7: The four basic wheel types⁴

- a) Standard-Rad (engl. standard wheel):** zwei Freiheitsgrade⁵; Rotation um die Radachse und um die Lenkachse.
- b) Castor-Rad (engl. castor wheel):** zwei Freiheitsgrade; Rotation um die vertikale Radachse und um die Lenkachse.
- c) Mecanum-Rad (engl. swedish wheel):** drei Freiheitsgrade; Rotation um die Radachse; Rotation der einzelnen Rollen und um die Lenkachse.
- d) Ball oder Kugelrad (engl. spherical wheel):** Rotation des Kugels um 360°, in jede Richtung möglich. Technische Realisierung sehr aufwendig.

Das Castor-Rad, Mecanum-Rad und das Kugelrad sind „omnidirektionale“ Räder. Omnidirektionale Räder haben die Eigenschaft zu jedem Zeitpunkt, in jede Richtung (x, y) zu fahren und jegliche Rotation auszuführen. Des Weiteren werden Roboter mit omnidirektionalen Rädern als „holonome“ Roboter bezeichnet. Ein Roboter ist holonom wenn er zu jedem Zeitpunkt jeden verfügbaren Freiheitsgrad nutzen kann.

⁴ [1], S. 31, Figure 2.19

⁵ Freiheitsgrad: die Zahl der voneinander unabhängigen Bewegungsmöglichkeiten eines Systems.

Anders als beim legged mobile robots können wheeled mobile robots bereits mit zwei Rädern eine statische Stabilität erreichen. Jedoch muss der Massenmittelpunkt des Roboters (mit zwei Rädern) unterhalb der Radachse liegen. Läge der Massenmittelpunkt zu hoch, würde der Roboter umkippen. Deshalb erfordert die statische Stabilität mindestens drei Räder⁶.

Sensorik (engl. Perception):

Ein Sensor (lat. Sensus, dt. der Sinn) ist ein technisches Bauteil, das gemessene physikalische oder chemische Eigenschaften (z.B. Temperatur, Druck oder Entfernung) erfassen und in ein geeignetes elektrisches Signal umformen kann.

Mobile Roboter besitzen Sensoren, um Informationen aus der Umgebung zu erfassen und diese für bestimmte Tätigkeiten einzusetzen. Die Sensoren sind die Sinnesorgane des Roboters. Es werden häufig verschiedene Arten von Sensoren für einen Roboter benötigt, um seine Umwelt und seinen inneren Zustand zu überwachen. In der Robotik unterscheidet man zwischen externen und internen Sensoren:

- **Externe Sensoren:** sammeln Informationen über die Umgebung des Roboters (z.B. die Entfernung, Navigation, Hindernisse, Bilder von der Umwelt usw.)

Externe Sensoren, die oft in der Robotik eingesetzt werden sind zum Beispiel Näherungssensoren zur Ortung von Objekten in einem bestimmten Abstand (z.B. Lichtschranken), Positionssensoren zur Navigation (z.B. GPS), CCD-Kameras⁷ zur Bilddurchnahme usw.

- **Interne Sensoren:** überwachen den inneren Zustand eines Roboters (z.B. die Position, die Orientierung und die Geschwindigkeit).

Interne Sensoren, die oft in der Robotik eingesetzt werden sind zum Beispiel Gyroskope zur Positionsbestimmung oder Messung von Richtungsänderungen, Beschleunigungssensoren zur Bestimmung der Geschwindigkeit des Roboters usw.

⁶ Vgl. [1], S. 33 Kapitel 2.3.1.2

⁷ CCD-Sensor (charged coupled device sensor): wandelt Lichtsignale in elektrische Signale um und wird als Bildsensor in optischen Erfassungseinrichtungen wie Digitalkameras, digitale Videokameras usw. eingesetzt

Lokalisierung (engl. Localisation):

Mithilfe der Lokalisierung ist ein mobiler Roboter in der Lage seine Position festzustellen. Nur auf dieser Grundlage kann der Roboter den Weg zu seiner Zielposition bestimmen. Bei der Lokalisierung wird die Position, bestehend aus den Koordinaten x und y und die Orientierung θ bestimmt. Diese Kombination aus Position und Orientierung wird „Pose“ genannt.

Die Aufgabe der Lokalisierung kann in der Robotik in zwei wesentliche Fällen unterschieden werden⁸:

- **Lokale Lokalisierung**

Bei der lokalen Lokalisierung ist die Startposition bzw. die aktuelle Pose des Roboters in seiner Umwelt bekannt. Bei Bewegung des Roboters wird die Pose kontinuierlich aktualisiert. Dafür werden Sensoren eingesetzt, die die Bewegung des Roboters erfassen. In der lokalen Lokalisierung wird häufig die Odometrie-Sensorik zur Bestimmung der Position und der Orientierung eingesetzt. Hierbei wird bei wheeled-mobile-robots die Anzahl der Radumdrehungen gemessen, während bei den legged-mobile-robots die Anzahl der Schritte zur Messung der Pose verwendet wird. Ein Sensor, der für die Messungen in der Odometrie-Sensorik verwendet wird, ist z.B. der Inkrementalgeber, der pro Umdrehung eine genau definierte Anzahl von Impulsen ausgibt.

- **Globale Lokalisierung:**

Bei der globalen Lokalisierung ist die aktuelle Pose des Roboters in seiner Umwelt nicht bekannt. Der Roboter bestimmt anhand künstlicher und natürlicher Landmarken seine Position. Künstliche Landmarken die häufig bei der Lokalisierung eingesetzt werden sind z.B. Barcodestreifen, GPS, Lichtquellen, Ampeln usw. Natürliche Landmarken sind Umgebungsmerkmale wie Türöffnungen, Wanddecken, Möbel usw. Ist die Pose bestimmt, kann der Roboter mit der lokalen Lokalisierung fortfahren.

Nachdem der Roboter seine Position, mithilfe der Sensorik lokalisiert hat kann er den Pfad zu seiner Zielposition planen und die Bewegung (wie in Abb. 5 dargestellt) ausführen.

⁸ vgl. [14]

3. Sphero

Sphero ist ein kugelförmiger, mobiler Spielzeugroboter von der Firma Orbotix, der mit einem Android- oder iOS-Gerät über Bluetooth verbunden werden kann. Es sind mehrere Apps und Spiele vorhanden, mit denen Sphero gesteuert werden kann:



Abb. 8: Sphero und die Firmen eigene Controller-App [5]

3.1 Hardware

Um den Roboter herum ist eine harte Plastikschale, die ihm seine Wasser- und Stoßfestigkeit verleiht. Im Inneren befindet sich die gesamte Elektronik des Roboters. In der Abb. 9 ist Spheros Hardware dargestellt. Die wichtigsten Komponenten sind mit Nummern [1-6] markiert.



Abb. 9: Sphero Hardware [5]

1. Stützräder auf beiden Seiten
2. Bluetooth Modul: dient zur Verbindung des Roboters mit Android- und iOS-Geräten.
3. Platine mit elektronischen Bauteilen, z.B.: Gyroskop zur Messung von Richtungsänderungen, Beschleunigungssensor zur Bestimmung der Geschwindigkeit sowie Kollisionserkennung, RGB-LED, 75 MHz Prozessor usw.
4. Elektrisch angetriebene Räder auf beiden Seiten
5. Motor, der die Räder (unten) antreibt.
6. Induktionsspule zum kabellosen aufladen (350 mAH LiPo Batterien)

3.2 Software

3.2.1 Sphero API und Firmware⁹

Die Programmierschnittstelle (engl. API)¹⁰ von Sphero stellt über die Hardware mehrere Funktionen zur Verfügung, die mit einer App auf einem Steuergerät (z.B. Android-Gerät) genutzt werden können. Die Funktionen werden von der Firmware implementiert, die fest mit der Hardware verbunden ist und somit eine Zwischenstellung zwischen der Hardware und der App einnimmt.

Verbindungsaufbau über Bluetooth:

Die Verbindung eines Steuergeräts mit Sphero erfolgt über Bluetooth. Nach der Verbindung erfolgt die Datenübertragung zwischen der App und Sphero nach dem Client-Server-Prinzip. Diese Beziehung beschreibt den Informationsfluss zwischen Sphero und der App. Die App sendet als Client Befehle an Sphero, der die Befehle als Server annimmt und nach diesen handelt. Diese Art von Kommunikation wird auch synchrone Kommunikation genannt, bei dem die Kommunikationspartner beim Senden oder Empfangen von Daten immer synchronisieren, also warten, bis die Kommunikation abgeschlossen ist.

Steuerung der Hardware I/O-Elemente:

Die I/O-Elemente von Sphero (RGB-LED und der Motor) können über der API in der App gesteuert werden. Um die RGB Farben zu ändern, werden die drei Farbwerte Rot, Grün und Blau an Sphero gesendet. Sphero ändert die Farbe des LEDs entsprechend den eingestellten Farbwerten. Auch die Rotationsgeschwindigkeit des Motors wird über den eingestellten Wert in der App von Spheros Firmware angepasst.

Kollisionserkennung:

Spheros Kollisionserkennung ist eine, von der Firmware bereitgestellte Funktion, die bei einem Aufprall einen Impuls auslöst. Die Erkennung basiert auf Schwellenwertparameter, die in der App eingestellt werden können.

Die Kollisionserkennung ermittelt die Kollision anhand der x- und y-Achse von Sphero.

⁹ Vgl. [7]

¹⁰ API (application programming interface): Ein Programmteil, der Funktionen und Befehle eines Softwaresystems anderen Programmen zur Verfügung stellt.

Die Schwellenwerte für die x- und y-Achse werden unabhängig voneinander kontrolliert. Das Diagramm in der Abb. 10 zeigt die bei der Kollisionserkennung beteiligten Parameter.

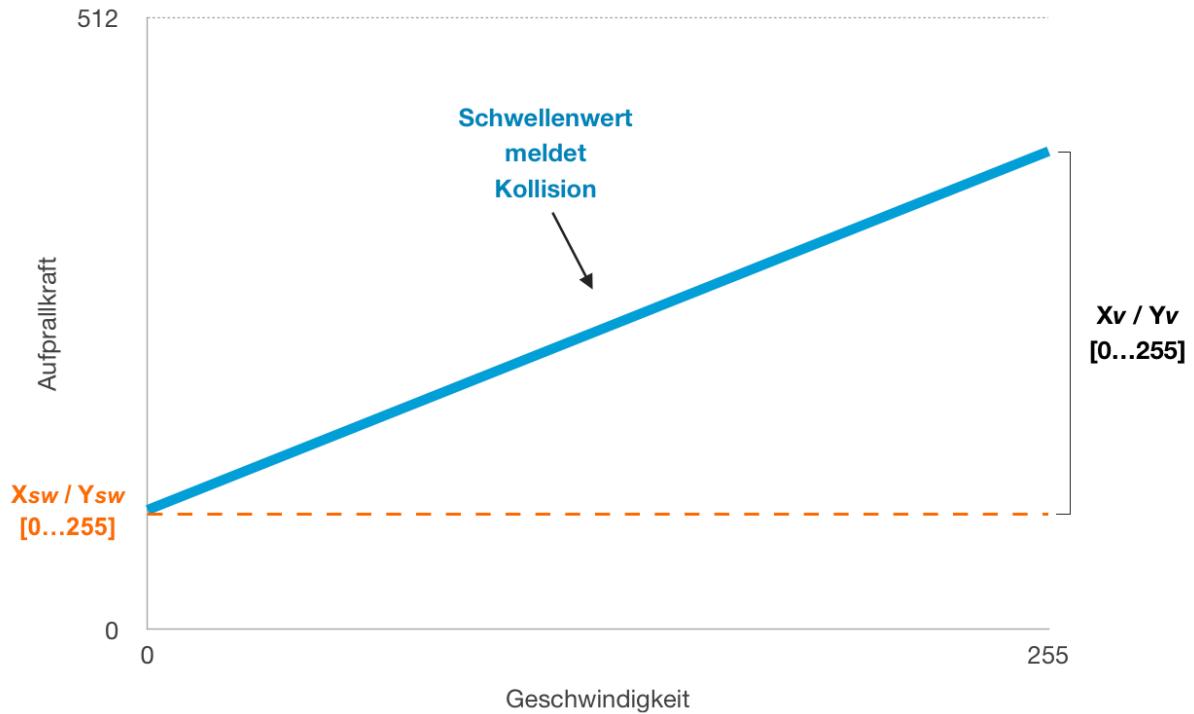


Abb. 10: Kollisionserkennung mittels Schwellenwert

X_{sw} und Y_{sw} sind zwei Schwellenwertparameter für die x- und y-Achse. Sobald eine Aufprallkraft signalisiert wird, die höher ist als der Schwellenwert, wird eine Kollision an die App gemeldet. Die Geschwindigkeitsparameter X_v und Y_v werden mit den Schwellenwertparametern zusammenaddiert.

$$X_v = X_v + X_{sw} \quad Y_v = Y_v + Y_{sw}$$

Diese Geschwindigkeitsparameter beschreiben den Schwellenwert bei der maximalen Geschwindigkeit vor der Kollision.

Positionsgeber:

Der Positionsgeber ist, genauso wie die Kollisionserkennung eine von der Firmware bereitgestellte Funktion, die die Echtzeit-Position und die Geschwindigkeit von Sphero bereitstellt. Dabei wird die Position (x, y Koordinaten) in cm und die Geschwindigkeit in mm/s angegeben.

In der Abb. 11 sind Spheros mögliche Bewegungsrichtungen dargestellt. Momentan erlaubt die Firmware die Bewegung nur in x- und y-Richtung. Die Bewegung auf der z-Koordinate wird mit der jetzigen Elektronik und der Firmware nicht unterstützt.

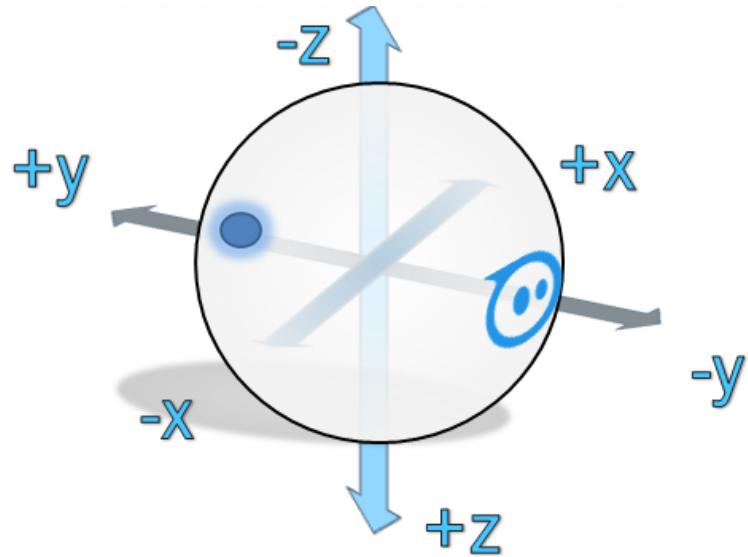


Abb. 11: Bewegungsmöglichkeiten des Spheros in 3D-Umfeld [5]

Der in der Abb. 11 dargestellte blaue Punkt zeigt die Richtung an, in der sich Sphero bewegt. In der Abbildung bewegt sich Sphero in die $-y$ Richtung. Der Punkt wird mithilfe einer blauen LED angezeigt und ist um 360° rotierbar. Somit lässt sich Sphero in jede Richtung, auf der xy-Ebene bewegen.

3.2.2 Sphero SDK und ADT

Für die Entwicklung der Android App für Sphero wird die Entwicklungsumgebung Eclipse 4.2 und die Programmiersprache Java verwendet. Um Android Apps zu entwickeln wird zusätzlich noch ein SDK (engl. Software Development Kit) benötigt. Ein SDK stellt Softwareentwicklern Werkzeuge und Anwendungen bzw. Bibliotheken zur Verfügung, um eine Software zu erstellen. Das Android SDK wird von Android Development Tools (kurz ADT) von Google angeboten und wird in Eclipse integriert.

In der Abb. 12 ist die Entwicklungsumgebung Eclipse mit der installierten Android SDK zu sehen.

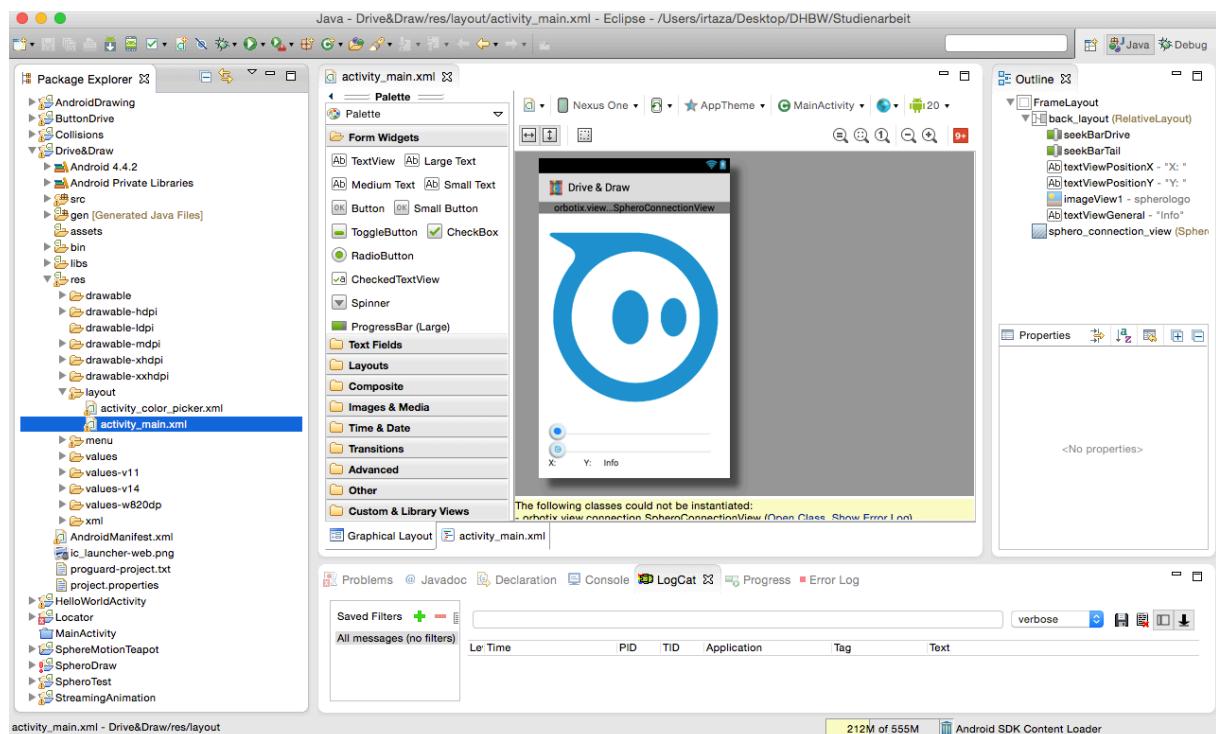


Abb. 12: Eclipse IDE mit ADT

Zur Erstellung der grafischen Oberfläche für die App, stellt ADT mehrere GUI-Komponenten wie Buttons, Textfelder, Fortschrittsbalken usw. zur Verfügung (siehe Abb. 12 Mitte). Des Weiteren ist in der SDK ein USB-Treiber enthalten, mit dem Android-Projekte auf dem angeschlossenen Android-Gerät installiert und debuggt werden können.

In Eclipse wird nun ein Android-Projekt angelegt. In diesem Projekt wird die Android-App für die Navigation von Sphero sowie die Kartenerstellung der Umgebung implementiert. Da die App das Fahren und die Steuerung von Sphero, sowie das

Zeichnen mit Sphero ermöglichen soll, wird auch der Name der App nach den App-Funktionen benannt: „Drive & Draw“.

Um eine Android App für Sphero zu entwickeln wird ein weiteres SDK und einige Bibliotheken benötigt, mit denen alle Funktionalitäten von Spheros Firmware genutzt und implementiert werden können. Die Bibliotheken werden per Hand in dem Bibliothek Ordner des Android-Projekts hinzugefügt (in der Abb. 13 blau umrandet).

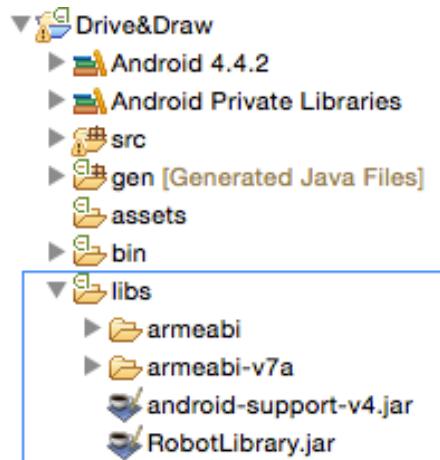


Abb. 13: Importieren von Sphero SDK Bibliotheken

Das Sphero SDK stellt weitere GUI-Komponenten zur Verfügung, die für die Entwicklung von Android-Apps für Sphero angepasst sind.

Nun kann die Android-App für die Navigation des Roboters Sphero entwickelt werden. Die Entwicklung der App und der Algorithmus für die Navigation von Sphero und für die Kartierung wird im nächsten Kapitel vorgestellt.

4. Entwurf und Implementierung von Drive & Draw

4.1 Designentwurf

Im ADT wird zuerst das Design für die App entworfen. Es werden verschiedene Steuerelemente benötigt, die im Folgenden beschrieben werden:

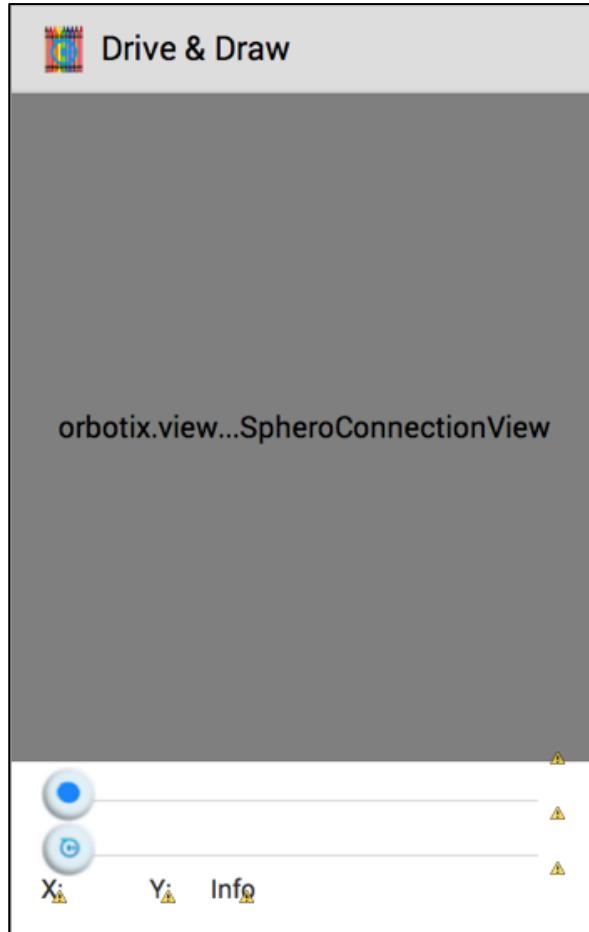


Abb. 14: Designentwurf für Drive & Draw

Für die Verbindung des Geräts mit Sphero, wird eine Sphero-Verbindungsanzeige (SpheroConnectionView) verwendet. Diese Anzeige bzw. Steuerelement wird von der Sphero SDK bereitgestellt. Darin werden alle aktiven Sphero-Roboter angezeigt, wenn das Bluetooth des Geräts eingeschaltet ist. Der Benutzer kann seinen Sphero Roboter aus der Liste auswählen und sich damit verbinden.

Für die Steuerung von Sphero werden zwei Fortschrittsbalken verwendet. Der obere Fortschrittsbalken (in der Abb. 14) ist für die Richtungsänderung (um 360°) und der untere zum Fahren (mit Geschwindigkeitsstufen von 0 – 10).

Des Weiteren werden zwei Textanzeigen für die Koordinatenanzeige (x, y) benötigt und eine weitere Textanzeige, um den Benutzer bestimmte Informationen, wie z.B.

die Geschwindigkeit und den Richtungsgrad des Roboters oder Warnung bei Kollisionen auszugeben.

Als letztes wird ein Canvas-Element angelegt, der für die Kartenerstellung benötigt wird. Ein Canvas ist ein, mit Höhen- und Breitenangaben beschriebener Bereich, in den gezeichnet werden kann.

4.2 Algorithmus

Bevor die App und die einzelnen Elementen implementiert werden, wird ein Algorithmus entworfen, der die Abläufe der App darstellen soll. Mithilfe des Entwurfs wird eine saubere und effiziente Entwicklung der App ermöglicht.

Das Sequenzdiagramm in der Abb. 15 stellt den Entwurf bzw. den Ablauf des Algorithmus dar.

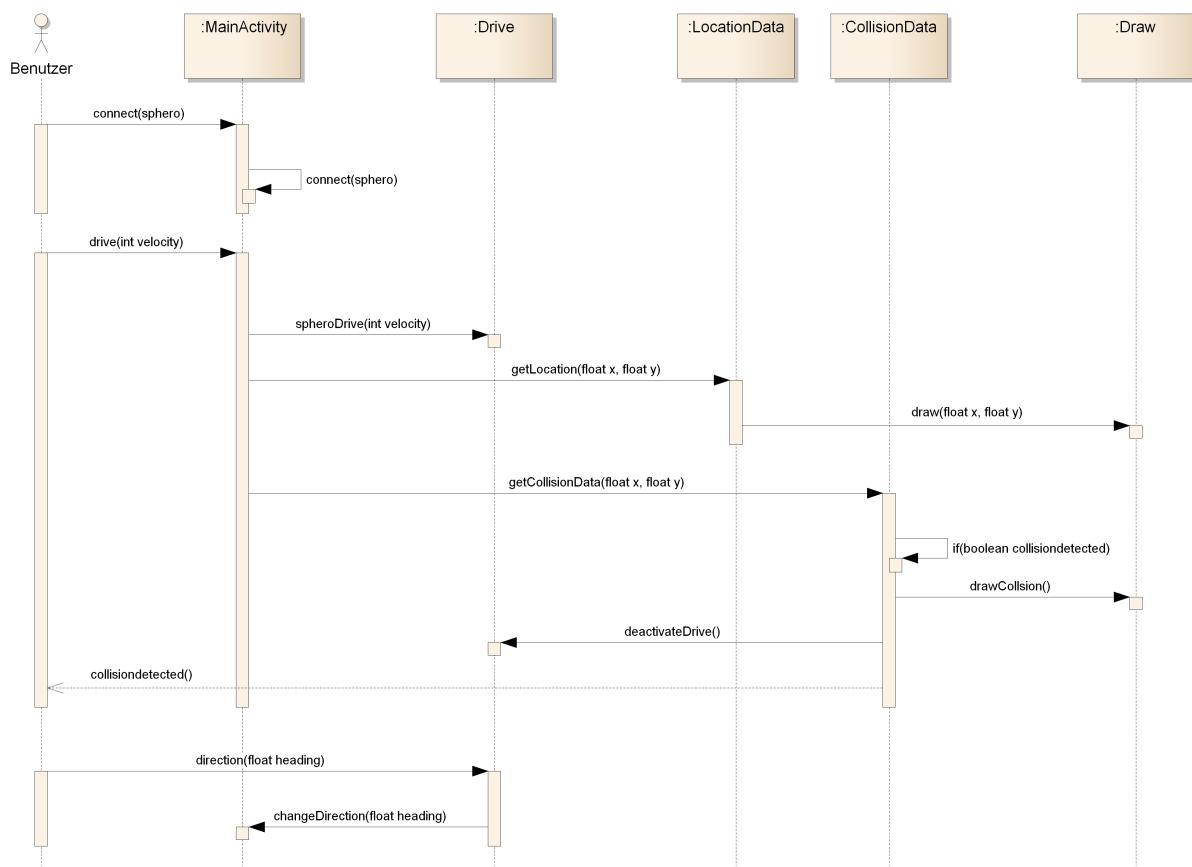


Abb. 15: Algorithmus der App Drive & Draw

Der Benutzer wird zuerst aufgefordert ein Sphero-Roboter aus der Liste (falls vorhanden) auszuwählen. Die App verbindet dann das Gerät mit dem ausgewählten Roboter.

Nachdem der Benutzer sich mit Sphero verbunden hat wird ihm die „MainActivity“ angezeigt. Android-Apps bestehen in der Regel aus mehreren Activities. Eine Activity stellt eine Bildschirmseite in einer App dar. Die MainActivity ist die Hauptbildschirmseite der App, in der sich die in der Abb. 14 dargestellten GUI-Komponenten befinden.

Der Benutzer kann nun, mithilfe der Steuerelemente Sphero bedienen. Möchte der Benutzer fahren, wird diese Aktion an die MainActivity gesendet. Für die einzelnen Funktionen der App werden Unterklassen angelegt, in der die Funktionen implementiert werden. Die MainActivity leitet die vom Benutzer ausgelöste Aktion „Fahren“ an die Klasse „Drive“ weiter. Die Drive-Klasse führt die Aktion aus. Gleichzeitig fordert MainActivity die Klasse „LocationData“ dazu auf, seinen Algorithmus auszuführen. LocationData berechnet die aktuelle Position von Sphero. Die Position wird in x- und y-Koordinaten ausgegeben. Diese Koordinaten sendet LocationData an die Klasse „Draw“, die auf dem Canvas jede Position von Sphero aufzeichnet. Parallel zu der Zeichnung wird die Klasse „CollisionData“ von MainActivity aufgerufen. CollisionData überprüft bei jeder Bewegung von Sphero, ob eine Kollision erkannt wurde. Die Implementierung der Klassen wird im nächsten Abschnitt genauer erläutert.

Möchte der Benutzer die Richtung des Roboters ändern, wird diese Aktion ebenfalls an MainActivity gesendet, die wiederum den Befehl für die Richtungsänderung an die Klasse „Drive“ weiterleitet. Die Klasse Drive führt die Aktion aus und löst keine weiteren Aktionen mehr aus.

4.3 Klassenstruktur

In dem Projekt werden nun, entsprechend dem Algorithmus alle Klassen angelegt und implementiert:

4.3.1 Die Klasse Drive

Die Klasse Drive implementiert zwei Fortschrittsbalken (engl. Seekbars). Die Fortschrittsbalken werden jeweils mit einer Ereignisbehandlungsroutine (engl. listener) verknüpft (siehe Abb. 16). Diese Ereignisbehandlungsrouterien werden immer dann ausgeführt, wenn ein bestimmtes Ereignis auftritt. In diesem Fall wird das Ereignis von dem Benutzer ausgeführt, indem er die Fortschrittsbalken zum Fahren oder für Richtungsänderung bedient.

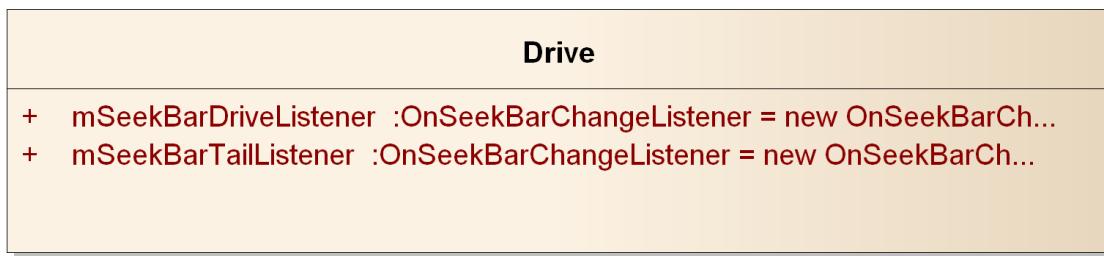


Abb. 16: Die Klasse Drive

4.3.2 Die Klasse Draw

In der Klasse Draw werden einige Methoden angelegt, die dem Benutzer für die Zeichnungen der Karte unterschiedliche Funktionen bereitstellen (siehe Abb. 17):

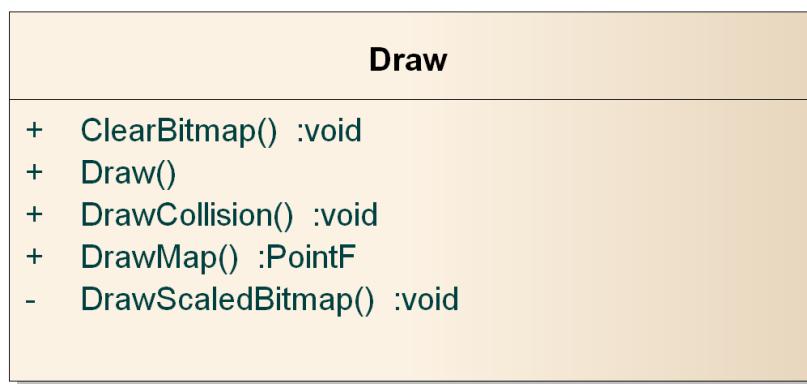


Abb. 17: Die Klasse Draw

Im Folgenden werden die Methoden kurz erläutert:

- **ClearBitmap():** löscht die Zeichnung auf dem Zeichenfeld und setzt es im Ausgangszustand.

- **Draw()**: Konstruktor von der Klasse Draw, in der bestimmte Einstellungen für die Zeichnung vorgenommen werden und die Zeichenfläche bzw. das Canvas-Element initialisiert wird.
- **DrawCollision()**: zeichnet einen roten Punkt an der Stelle, an der eine Kollision erkannt wurde.
- **DrawMap()**: zeichnet die Position von Sphero auf dem Zeichenfeld und erstellt somit eine Karte.
- **DrawScaledBitmap()**: vergrößert die Zeichenfläche mit der Zeichnung um 1/3 der Displaygröße, sobald Sphero die Grenzen der Zeichenfläche erreicht.

4.3.3 Die Klasse LocationData

LocationData implementiert eine Ereignisbehandlungsroutine namens LocatorListener. Diese Ereignisbehandlungsroutine wird bei jeder Bewegung von Sphero ausgelöst. Da die App für die Kartierung jede Bewegung von Sphero aufzeichnen soll, wird die Methode „DrawMap()“ von der Klasse Draw in dieser Ereignisbehandlungsroutine aufgerufen. Somit wird die Kartierung mit der Echtzeit-Position von Sphero (mPositionX, mPositionY) ermöglicht.



Abb. 18: Die Klasse LocationData

4.3.4 Die Klasse CollisionData

Auch die Klasse CollisionData implementiert eine Ereignisbehandlungsroutine (CollisionListener), welche bei jeder Kollision ausgelöst wird. Für die Kollisionserkennung wird die im Kapitel 3.2.1 genannte Berechnung verwendet. Die Kollision tritt immer dann auf, wenn der in der App festgelegte Schwellenwert durch einen Aufprall überschritten wird. Die Kollision führt die Methode „DrawCollision()“ von der Klasse Draw auf, die dem Benutzer die Kollision durch einen roten Punkt auf der Zeichenfläche mitteilt.

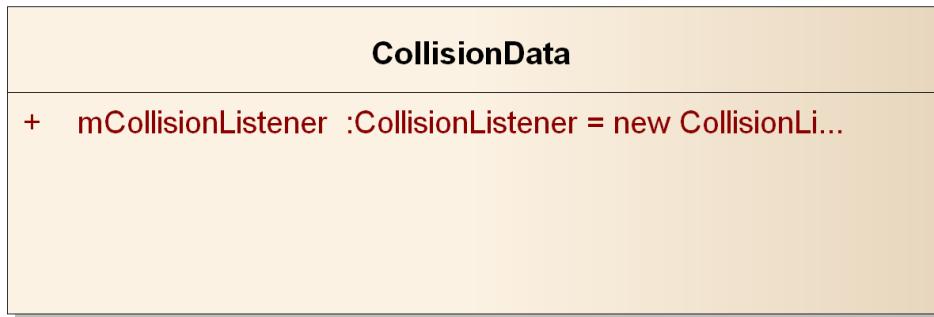


Abb. 19: Die Klasse CollisionData

4.3.5 MainActivity

In der Abb. 20 ist die komplette Klassenstruktur für MainActivity dargestellt. MainActivity hat Zugriff auf alle Unterklassen und ist somit eine Schnittstelle zwischen dem Benutzer und der App. MainActivity nimmt die Aktionen des Benutzers entgegen und leitet diese an die jeweiligen Unterklassen weiter.

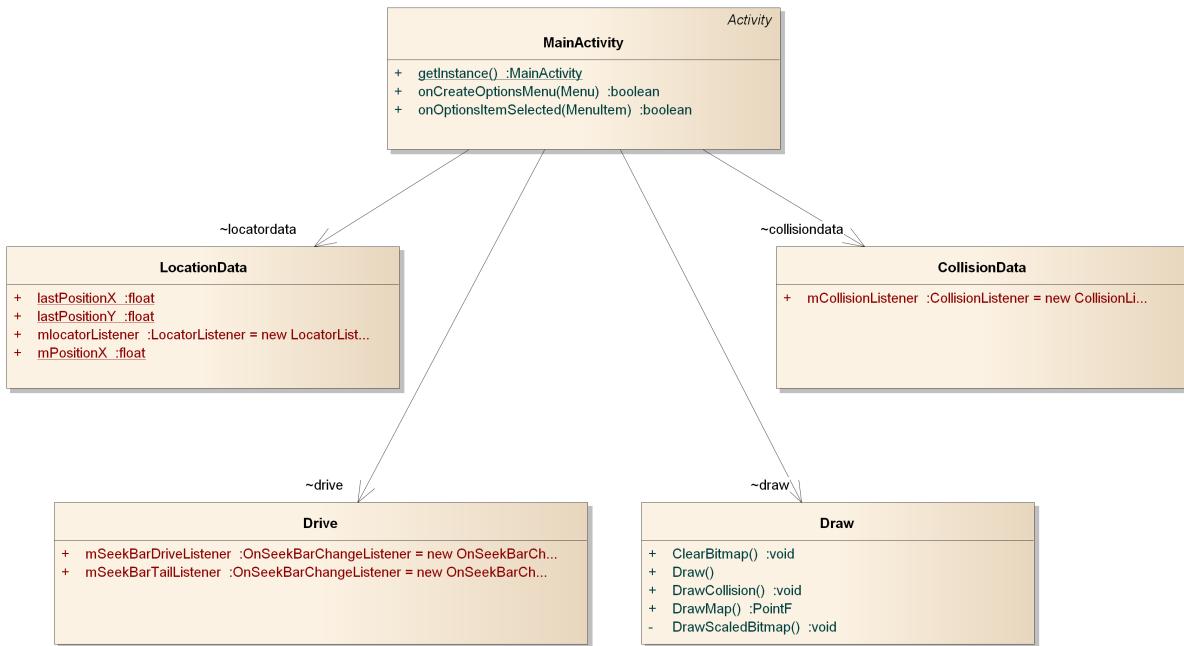


Abb. 20: Klassenstruktur für Drive & Draw

4.3.6 Zusatzfunktionen

Die App stellt dem Benutzer einige weitere Zusatzfunktionen zur Verfügung. Viele Android-Apps besitzen Einstellungen, die dem Benutzer erlauben bestimmte Eigenschaften und Verhaltensweisen der App zu modifizieren.

Auch in der App Drive & Draw werden dem Benutzer bestimmte Modifikationen ermöglicht. Die Einstellungen werden in einer separaten Activity („SettingsActivity“) zur Verfügung gestellt:

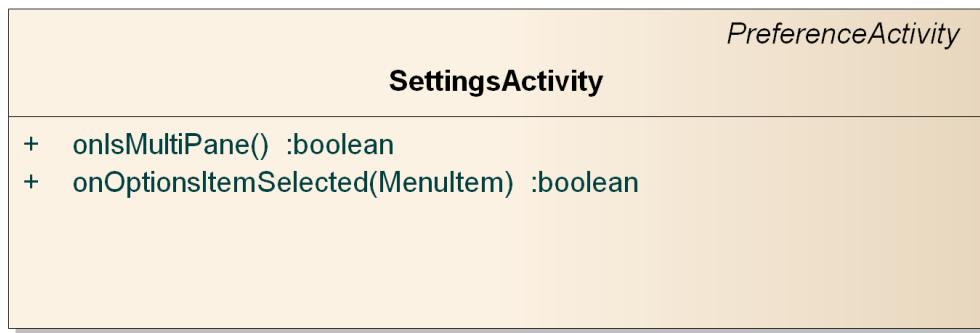


Abb. 21: SettingsActivity

Darin sind mehrere variable Einstellungen implementiert, deren Änderung einen direkten Einfluss auf die App hat.

Eines davon ist die freie Farbauswahl des Zeichenstifts für die Zeichnung auf der Zeichenfläche sowie die Farbauswahl des RGL-LEDs von Sphero.

Die Farbauswahl erfolgt über vier Fortschrittsbalken: Drei Fortschrittsbalken für die Einstellung der Farbwerte Rot, Grün und Blau und ein Fortschrittsbalken für den Alphawert, der die Helligkeit der Farbwerte bestimmt. Die Steuerelemente werden in einer neuen Activity („ColorPickerActivity“ siehe Abb. 22) angelegt, die von der SettingsActivity gestartet wird.

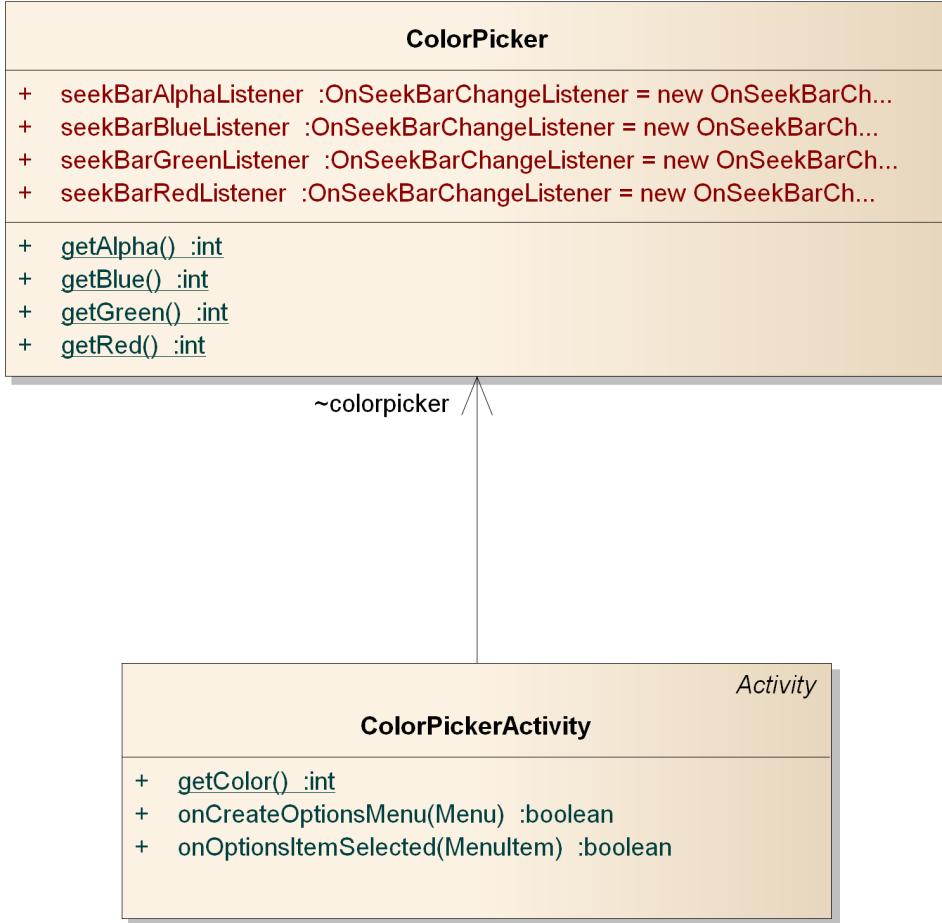


Abb. 22: ColorPickerActivity für die Farbauswahl

Der Farbwert wird in einer separaten Klasse „ColorPicker“ berechnet. ColorPickerActivity dient hier, wie andere Activities auch nur als eine Schnittstelle zwischen dem Benutzer und der Farbauswahl-Funktion.

Weitere mögliche Einstellungen in der SettingsActivity sind:

- Strichstärke für die Zeichnung festlegen
- Option zur Aktivierung und Deaktivierung der Kollisionserkennung
- Sensitivität der Kollisionserkennung (Schwellenwerte: High, Medium und Low)

5. Ergebnis

5.1 Startbildschirm

Im Startbildschirm werden alle, in der Reichweite des Bluetooth-Signals verfügbare Sphero Roboter angezeigt. Wählt der Benutzer einen Roboter aus, verbindet die App den Roboter mit dem verwendeten Gerät.

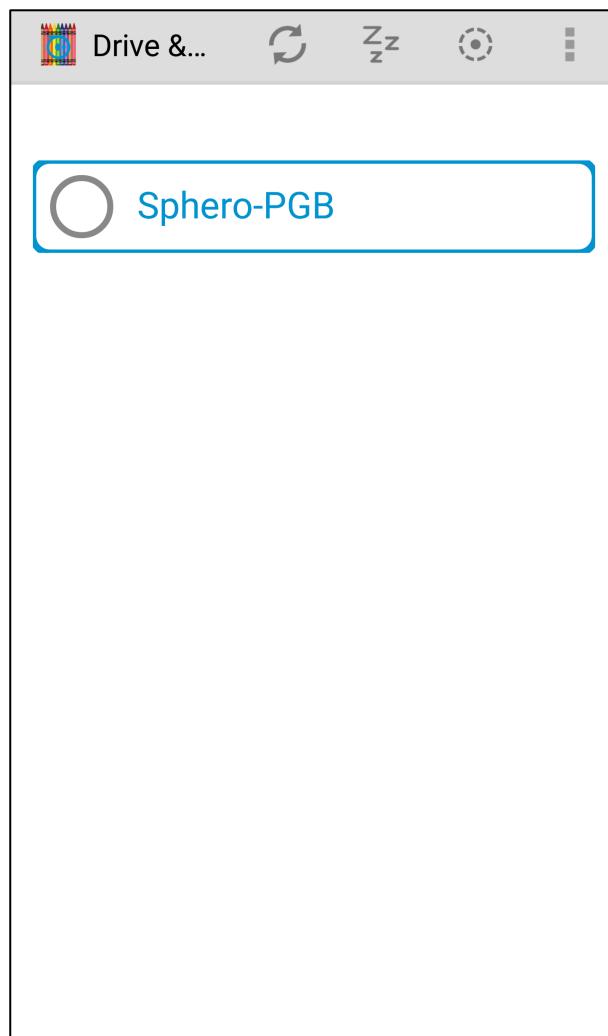


Abb. 23: Startbildschirm der App Drive & Draw

5.2 MainActivity

Nachdem sich der Benutzer mit Sphero verbunden hat, wird die MainActivity angezeigt. In der MainActivity hat der Benutzer Zugriff auf alle Bedienelemente und mehrere Einstellungsoptionen:

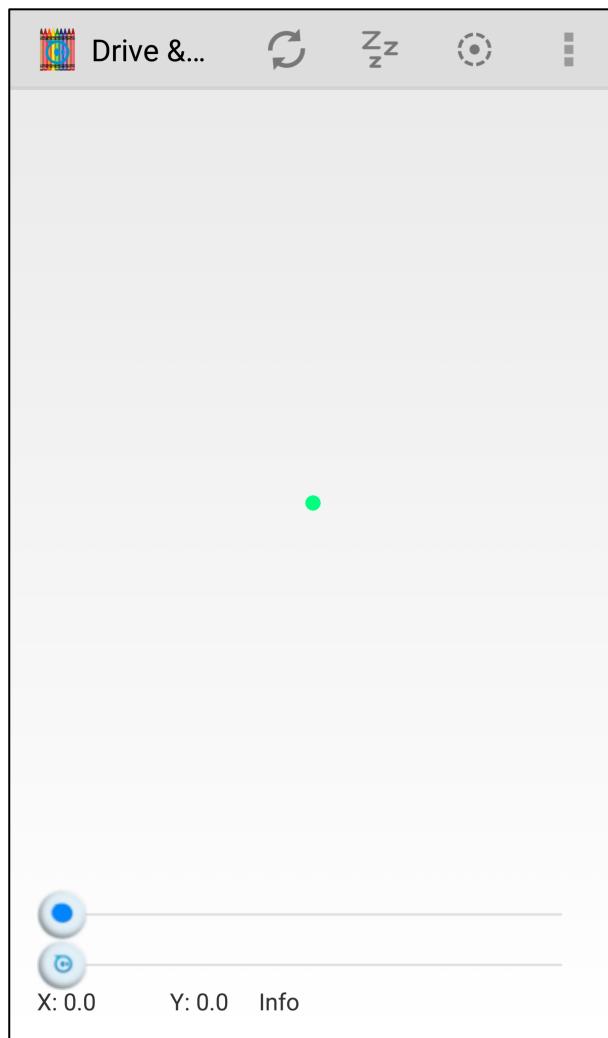


Abb. 24: Zeichenfläche mit Bedienelementen

Der Grüne punkt in der Mitte der Abb. 24 zeigt die aktuelle Position von Sphero an ($x=0$, $y=0$).

5.3 Zeichnung

Im Abb. 25 ist eine Zeichnung zu sehen, welche mit den Positionsdaten aus der Klasse LocationData erstellt wurde:

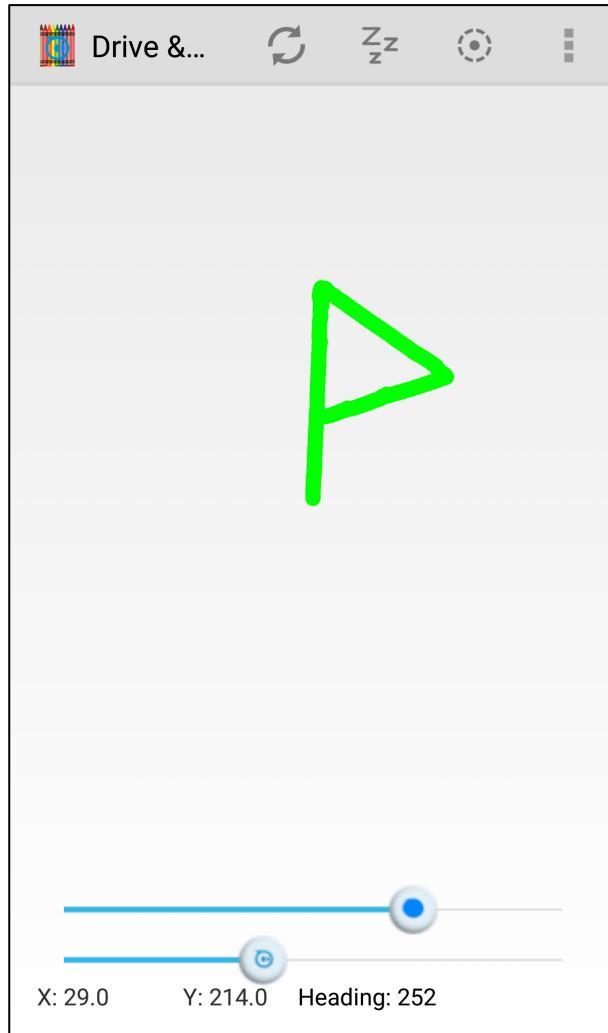


Abb. 25: Mit Sphero erstellte Zeichnung

5.4 Darstellung von Kollisionen

Die Kollisionen werden auf der Zeichenfläche mit roten Punkten markiert:

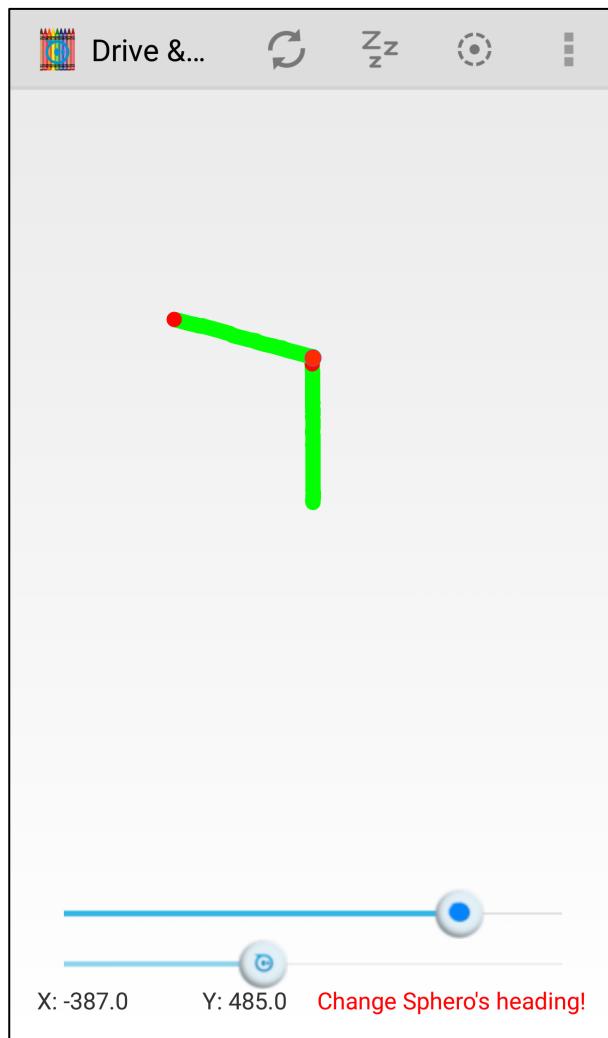


Abb. 26: Darstellung von Kollisionen

Des Weiteren wird bei jeder Kollision der Fortschrittsbalken zum Fahren deaktiviert. Der Benutzer wird aufgefordert die Richtung von Sphero zu ändern. Sobald der Benutzer die Richtung ändert, wird der Fortschrittsbalken zum Fahren wieder freigegeben. Würde der Benutzer bei einer Kollision in die selbe Richtung weiterfahren, würde dies die Kartierung verfälschen.

5.5 SettingsActivity

In der SettingsActivity sind alle verfügbaren Optionen in einer Liste dargestellt. Durch das anklicken einer Option wird ein separates Fenster geöffnet, in der die Einstellungen vorgenommen werden können:

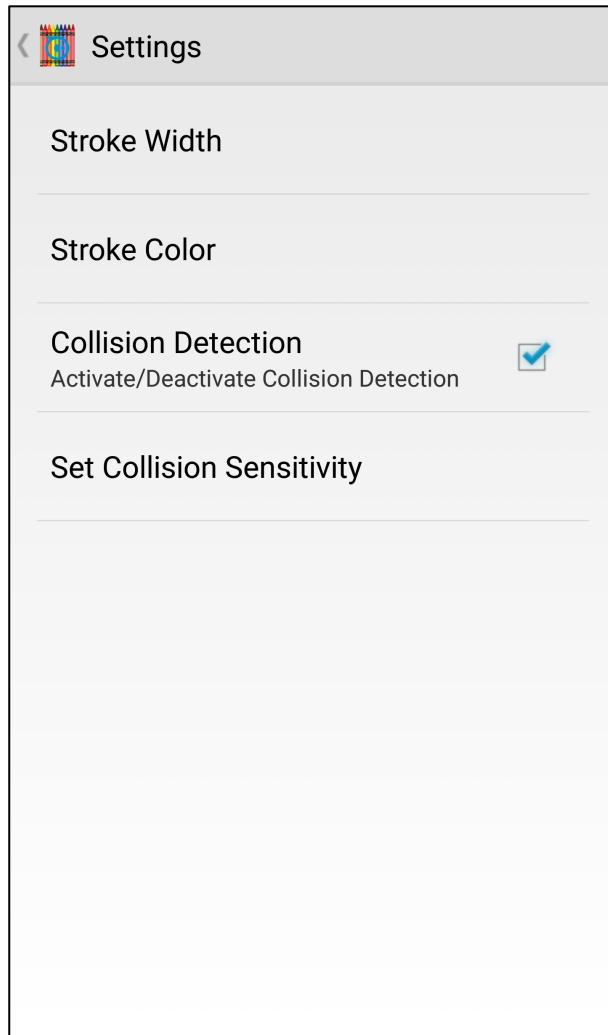


Abb. 27: SettingsActivity mit Einstellungen

5.6 Einstellungsoption: Strichstärke

Die Einstellungsoption „Set Stroke Width“ stellt dem Benutzer 5 Strichstärken zur Auswahl:



Abb. 28: Einstellungsänderung für die Strichstärke

Die Strichstärke wird in „dp“ (engl. Density-independent pixel)¹¹ angegeben. Density-independent pixel ist ein virtuelles Pixel, welches von der Punktdichte des Bildschirms abhängt. Durch die Verwendung von *dp* wird die Darstellung von graphischen Elementen auf unterschiedliche Bildschirmgrößen unterstützt.

¹¹ Vgl. [15]

5.7 Einstellungsoption: Farbauswahl

Die Farbauswahl des Zeichenstifts und Spheros RGB-LED erfolgt in der ColorPickerActivity. Dafür wird der Farbraum RGBA (Rot, Grün, Blau und Alpha) verwendet:

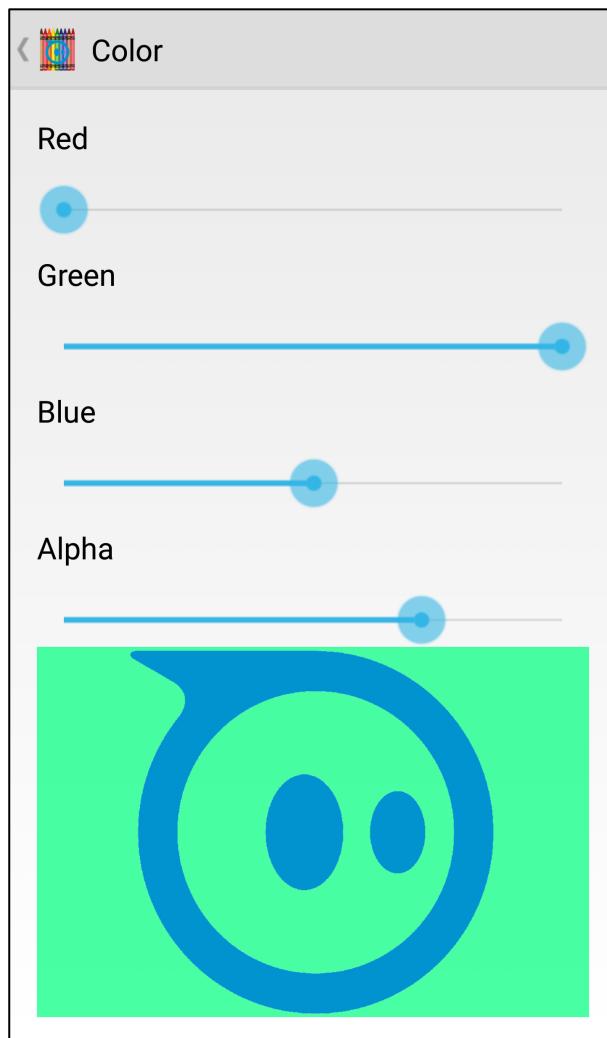


Abb. 29: Farbauswahl in der ColorPickerActivity

Die Farbänderung wird dem Benutzer bei der Änderung der Farbwerte simultan, in der App und auch am Sphero angezeigt.

5.8 Einstellungsoption: Kollision Sensitivität

Bei der Einstellungsoption „Set collision sensitivity“ hat der Benutzer 3 Schwellenwerte zur Auswahl:

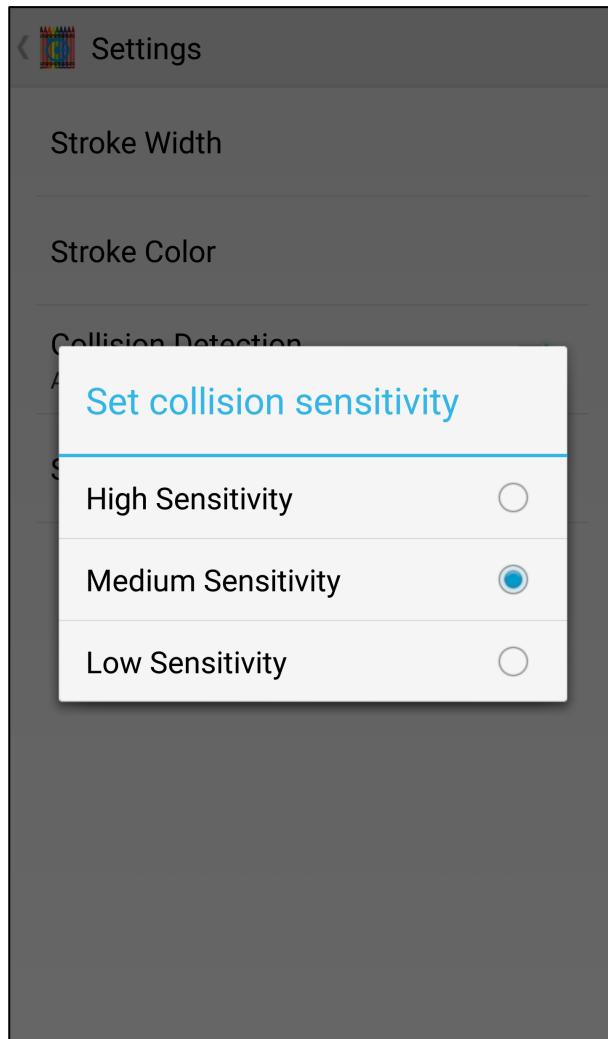


Abb. 30: Schwellwertänderung für die Kollisionserkennung

Der Schwellenwert bei der Option „High Sensitivity“ ist am niedrigsten, sodass damit auch leichte Kollisionen erkannt werden. Bei „Low Sensitivity“ schlägt die Kollisionserkennung erst bei einem stärkeren Aufprall Alarm. Die Option „Medium Sensitivity“ ist als Defaultwert eingestellt.

6. Ausblick

Die App Drive & Draw kann um viele Funktionen erweitert werden. Momentan bekommt der Benutzer eine Warnung von der Kollisionserkennungsfunktion, sobald eine Kollision erkannt wird. Als Alternative könnte ein Algorithmus entwickelt werden, der bei der Kollisionserkennung ein autonomes ausweichen des Hindernisses ermöglicht.

Die Kalibrierung von Sphero muss in der aktuellen Version der App manuell durchgeführt werden. In Zukunft könnte eine automatische Kalibrierung implementiert werden, die dem Nutzer mehr Komfort bietet.

Drive & Draw wurde in der Entwicklungsumgebung Eclipse implementiert. Für die Android App-Entwicklung musste ein Android-SDK in Eclipse integriert werden, da Android keine eigene Entwicklungsumgebung hatte. Google hat am 08. Mai 2015 eine offizielle Entwicklungsumgebung für Android namens „Android Studio“ veröffentlicht. Für weitere Entwicklungen an der App ist Android Studio zu empfehlen.

7. Fazit

Während des Projektverlaufs konnten alle gestellten Anforderungen erfolgreich umgesetzt und tiefgehendes Wissen in der Entwicklung von Android-Apps erlangt werden.

Des Weiteren konnten mithilfe von Sphero die Grundlagen der Robotik erlernt werden.

Die Entwicklungsarbeiten an der App wurden regelmäßig auf GitHub protokolliert und dokumentiert. Der gesamte Code der App ist online, auf GitHub verfügbar¹². Die App wurde außerdem im Google Play Store veröffentlicht¹³.

Rückblickend betrachtet war der Verlauf der Studienarbeit sehr positiv und sämtliche Probleme konnten mit dem Betreuer zügig behoben werden.

¹² siehe [16]

¹³ siehe [17]

Literaturverzeichnis

- [1] Roland Siegwart and Illah R. Nourbakhsh, Introduction to Autonomous Mobile Robots, 2004
- [2] Georg Stark, Robotik mit MATLAB, 2012
- [3] Thomas Christaller, et al., Robotik, Perspektiven für menschliches Handeln in der zukünftigen Gesellschaft. Springer, Berlin/Heidelberg/New York, 2001
- [4] Addison-Wesley, Android Wireless Application Development 2nd, Developer's Library, 2011
- [5] Orbotix Inc: Sphero,
<http://www.gosphero.com/de/sphero/>, Abruf 10.02.2015
- [6] Orbotix Inc: Sphero-Docs (2013),
<http://orbotixinc.github.io/Sphero-Docs/>, Abruf 10.02.2015
- [7] Orobotik Inc: Sphero API (2013),
<http://orbotixinc.github.io/Sphero-Docs/docs/sphero-api/index.html>,
Abruf 10.02.2015
- [8] Orbotix Inc: Sphero-Android-SDK,
<https://github.com/orbotix/Sphero-Android-SDK>, Abruf 10.02.2015
- [9] KUKA AG: Industrieroboter (2005),
<http://www.kuka.com/>, Abruf 22.02.2015
- [10] Bootic: iRobot Roomba 581,
http://static.bootic.com/_pictures/485814/irobot-roomba-581-vacuum-cleaning-robot_3.jpg, Abruf 22.02.2015
- [11] Aldebaran Robotics: NAO,
<https://www.aldebaran.com/en/humanoid-robot/nao-robot>, Abruf 23.02.2015
- [12] Spacenews: Mars Rover,
http://spacenews.com/wp-content/uploads/2014/11/MSL_NASA4X3.jpg, Abruf 23.02.2015
- [13] Robotplatform: Introduction to Robots & Robotics,
http://www.robotplatform.com/knowledge/Introduction/Introduction_to_Robots.html, Abruf 23.03.2015

- [14] Wikipedia: Lokalisierung (Robotik),
[http://de.wikipedia.org/wiki/Lokalisierung_\(Robotik\)](http://de.wikipedia.org/wiki/Lokalisierung_(Robotik)), Abruf 05.03.2015
- [15] Android Developer: Supporting Multiple Screens,
http://developer.android.com/guide/practices/screens_support.html, Abruf 11.04.2015
- [16] GitHub: Drive-Draw,
<https://github.com/IrtazaS/Drive-Draw>, 2015
- [17] Google Play: Drive & Draw,
<https://play.google.com/store/search?q=irtaza&hl=de>, 2015