

# Konzeption und Implementierung eines Schwarmverhaltens von mobilen Kleinrobotern anhand eines Verfolgungsszenarios

## STUDIENARBEIT

für die Prüfung zum  
Bachelor of Science  
des Studiengangs Informatik  
Studienrichtung Angewandte Informatik  
an der  
Dualen Hochschule Baden-Württemberg Karlsruhe

18. April 2017

Bearbeitungszeitraum	24 Wochen	
Name	Manuel Bothner	Simon Lang
Matrikelnummer	8359139	6794837
Kurs	TINF14B2	TINF14B2
Ausbildungsfirma	1&1 Internet SE	ifm ecomatic GmbH
	Brauerstr. 48	Im Heidach 18
	76135 Karlsruhe	88079 Kressbronn am Bodensee
Betreuer	Prof. Hans-Jörg Haubner	
Gutachter	Prof. Dr. Heinrich Braun	

## Erklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. 9. 2015)

Ich versichere hiermit, dass ich die Studienarbeit meiner Studienarbeit mit dem Thema: „Konzeption und Implementierung eines Sachwarmverhaltens von mobilen Kleinrobotern anhand eines Verfolgungsszenarios“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

---

Ort, Datum

Unterschrift

---

Ort, Datum

Unterschrift

# Abstract

# Zusammenfassung

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>11</b>
1.1	Ausgangslage . . . . .	11
1.2	Zielsetzung . . . . .	11
1.3	Erwartetes Ergebnis . . . . .	11
<b>2</b>	<b>Technische Grundlagen</b>	<b>12</b>
2.1	Robotik . . . . .	12
2.1.1	Roboter . . . . .	12
2.1.2	Mobile Roboter . . . . .	12
2.1.3	Sensorik . . . . .	13
2.1.4	Sensordatenverarbeitung . . . . .	14
2.1.5	Antriebsarten . . . . .	14
2.2	LEGO MINDSTORMS . . . . .	14
2.2.1	Das EV3-System . . . . .	14
2.2.2	Der EV3-Stein (Steuereinheit) . . . . .	15
2.2.3	Motoren . . . . .	16
2.2.4	Sensoren . . . . .	16
2.2.5	Programmierung . . . . .	18
2.3	Application (App) Entwicklung . . . . .	20
2.3.1	Native Apps . . . . .	20
2.3.2	Web Apps . . . . .	20
2.3.3	Hybride Apps . . . . .	21
2.3.4	Plattformübergreifende Entwicklung . . . . .	21
2.3.5	Xamarin . . . . .	22
2.3.6	Mono . . . . .	22
2.3.7	.NET Framework . . . . .	22
2.4	TCP-Kommunikation . . . . .	23
2.4.1	Gundlegendes . . . . .	23
2.4.2	Nagle-Algorithmus . . . . .	24
2.4.3	Kommunikationsablauf . . . . .	24
2.4.4	Socket-Programmierung . . . . .	26
2.5	Java . . . . .	26
2.5.1	Grundlagen . . . . .	26
2.5.2	Java Runtime Environment . . . . .	26
<b>3</b>	<b>Theoretische Grundlagen</b>	<b>26</b>
3.1	Schwarmverhalten . . . . .	26
3.1.1	Allgemein . . . . .	26

3.1.2	Vorbilder aus dem Tierreich . . . . .	26
3.1.3	Szenarien . . . . .	26
3.1.4	Algorithmen . . . . .	26
<b>4</b>	<b>Projektorganisation</b>	<b>27</b>
4.1	Projektablaufplan . . . . .	27
<b>5</b>	<b>Konzeption</b>	<b>28</b>
5.1	Anforderungsdefinitionen . . . . .	28
5.1.1	Softwarearchitektur . . . . .	28
5.2	Use Cases . . . . .	28
5.2.1	Connect . . . . .	28
5.2.2	Synchronization . . . . .	29
5.2.3	Szenario . . . . .	32
5.2.4	Exception . . . . .	32
5.3	Kommunikation . . . . .	32
<b>6</b>	<b>Lösungsansatz</b>	<b>38</b>
<b>7</b>	<b>Umsetzung</b>	<b>39</b>
<b>8</b>	<b>Evaluation</b>	<b>40</b>
<b>9</b>	<b>Zusammenfassung und Ausblick</b>	<b>41</b>







---

# Abbildungsverzeichnis

1	Zentrale Komponenten des EV3-Systems . . . . .	15
2	App Entwicklung . . . . .	20
3	Xamarin . . . . .	22
4	Mono . . . . .	22
5	TCP Verbindungsaufbau . . . . .	25
6	TCP Datenaustausch . . . . .	25
7	TCP Verbindungsabbau . . . . .	25
8	Connect . . . . .	28
9	Connection . . . . .	29
10	Synchronization . . . . .	30
11	Robot list . . . . .	31
12	Spectator . . . . .	32
13	Spectator . . . . .	33
14	Control . . . . .	34
15	Control . . . . .	35
16	Synchron . . . . .	36
17	Follow . . . . .	37

# Tabellenverzeichnis

1	Einordnung von Sensoren . . . . .	14
2	Eigenschaften der EV3-Motortypen . . . . .	16
3	Eigenschaften der EV3-Motortypen . . . . .	19

# **1 Einleitung**

Heutzutage werden viele Arbeitsschritte in der Produktion, als auch Dienstleistungen von Maschinen verrichtet, da diese effizienter Arbeiten und weniger Kosten als Menschen verursachen. Da jede Maschine auf einen spezifischen Arbeitsschritt konfiguriert ist, müssen die verschiedenen Maschinen untereinander wie ein Schwarm agieren. Diese Verhaltensstrukturen kommen ursprünglich aus dem Tierreich, wie Fischschwärme, Ameisen oder Bienen. Hierbei erledigt jedes Individuum seine zugewiesenen Aufgaben und hält die anderen Parteien auf dem aktuellen Stand.

In diesem Projekt werden diese Verhaltensmuster aus dem Tierreich aufgegriffen und anhand eines Verhaltensszenarios mit Kleinrobotern verwirklicht, die autonom agieren und kommunizieren, um zusammen ihr Ziel zu erreichen. Dabei sollen Konzepte, sowie Algorithmen für Schwarmroboter entstehen, die auch auf andere Szenarien angewendet werden können.

## **1.1 Ausgangslage**

## **1.2 Zielsetzung**

## **1.3 Erwartetes Ergebnis**

## 2 Technische Grundlagen

### 2.1 Robotik

Die Robotik beschäftigt sich mit dem Entwurf, der Konstruktion sowie der Programmierung, Steuerung und dem Betrieb von Robotern

[<http://wirtschaftslexikon.gabler.de/Definition/robotik.html>]. Dabei umfasst die Robotik eine Vielzahl von Fachgebieten wie der Elektrotechnik, dem Maschinenbau, der Informatik sowie der Biologie und der Medizin.

#### 2.1.1 Roboter

Was im Kontext der Robotik unter einem Roboter zu verstehen ist, ist gar nicht so einfach darzulegen, da es in Tat keine allgemein anerkannte Definition dieses Begriffs gibt, die seiner üblichen Verwendung entspricht [Mobile Roboter - 2].

Auch ohne eine vollkommen allgemeingültige und präzise Beschreibung eines Roboters zu sein, soll hier die VDI-Richtlinie 2860 einen Eindruck darüber vermitteln, was in der Robotik mit Roboter gemeint ist.

Die VDI-Richtlinie 2860 von 1990 definiert einen Roboter wie folgt:

*»Ein Roboter ist ein frei und wieder programmierbarer, multifunktionaler Manipulator mit mindestens drei unabhängigen Achsen, um Materialien, Teile, Werkzeuge oder spezielle Geräte auf programmierten, variablen Bahnen zu bewegen zur Erfüllung der verschiedensten Aufgaben.«*

Auch wenn diese Definition einige grundlegenden Eigenschaften eines Roboters darlegt, beschreibt diese Definition den Begriff des Roboters im industriellen Kontext und trifft hauptsächlich auf stationäre Industrieroboter zu, wie sie in der Automobilfertigung beispielsweise als Schweiß- oder Lackierroboter verwendet werden. Die genannten programmierten Bahnen sind dort möglich, weil die Arbeitsprozess und die Umgebung auf den Roboter zugeschnitten und vollständig bekannt sind [MR 2].

Im Gegensatz dazu trifft dieser Aspekt auf mobile Roboter nicht zu. Mobile Roboter wie beispielsweise Service-Roboter bewegen sich in einer unstrukturierten und dynamischen Umgebung.

#### 2.1.2 Mobile Roboter

Mobile Roboter sind in ihrer Position variable, bewegen sich selbständig von Ort zu Ort und interagieren mit ihrer Umgebung. Diese macht es notwendig das die Roboter ihre Umgebung erfassen und wahrnehmen können um auf Veränderungen wie beispielsweise Hindernisse reagieren zu können.

...

Zu den Vertretern mobiler Roboter zählen zum Beispiel Service-Roboter, Erkundungsroboter und Humanoide Roboter. Im Folgenden sind einige Beispiele für mobile Roboter dargelegt:

- **Shakey:** Shakey war ein mobiler Roboter der von 1966 bis 1972 an der am Stanford Research Institute entwickelt wurde. Seine Entwicklung leistete wichtige Beiträge für die Robotik sowie in der KI-Forschung im Bereich der Handlungsplanung und dem selbständigen Lernen [MR 5f].
- **Spirit & Opportunity:** Spirit & Opportunity sind zwei baugleiche Roboter die im Jahr 2003 von der NASA zum Mars geschickt wurden um den Himmelskörper zu erkunden. Die beiden Erkundungsroboter sind Radfahrzeuge mit flexiblem Fahrgestell, verfügen über eine Panoramakamera sowie Sensoren zur Untersuchung des Erdbodens und Gesteins. Obwohl die Roboter in Bezug auf ihrer grundsätzlichen Aktionen von der Erde aus ferngesteuert werden, ist eine autonome Steuerung welche auf kurzfristige, unerwartet Ereignisse wie das Wegrutschen von Rädern reagiert aufgrund der langen Signallaufzeiten unverzichtbar. Die Roboter waren für eine Lebensdauer von 90 Marstagen ausgelegt, übertrafen diese aber bei weitem (mehr als das 30fache) [MR 8f]
- **Stanley:** Stanley ist ein vollständig autonomer Roboter der 2005 am Grand Challenge Wettbewerb teilnahm und diesem gewann. Bei diesem Wettbewerb mussten Fahrzeuge ohne Eingriff von Menschen eine festgelegte, jedoch nicht markierte Strecke von rund 213 km von einem definierten Start- zu einem definierten Zielpunkt zurücklegen. Die Strecke führte durch die Mojave-Wüste in den USA. Bei Stanley handelt es sich um einen modifizierter VW Touareg, dem Sensoren zur Umgebungswahrnehmung und Bordrechner zur Bearbeitung des Kontrollprogramms eingebaut wurden. Stanley meisterte die 213 km lange Strecke welche unter anderem durch felsige oder sandige Bereiche sowie durch Wasserläufe führte in knapp unter 7 Stunden.
- **Tribot D**

Die Beispiele zeigen wie vielfältig ... Neben der Forschung -> Marktpotenzial erwähnen

### 2.1.3 Sensorik

Um mit der Umgebung interagieren zu können müssen mobile Roboter diese wahrnehmen, dazu dienen Sensoren. Sensoren ermöglichen es dem Roboter Informationen über seine Umwelt und über seinen Zustand zu sammeln. Sensoren lassen sich hinsichtlich ihrer Arbeitsweise und ... wie folgt klassifizieren:

- **Propriozeptive Sensoren** – Diese Art der Sensoren bestimmen eine Messgröße des Roboters selbst und haben keine „Kontakt“ zur Umwelt z.B. Bestimmung der Lage Aufgrund eines Neigungssensors.
- **Exterozeptive Sensoren** – Im Gegensatz zu den propriozeptive Sensoren gewinnen diese Sensoren Informationen aus Messgrößen der Umwelt beispielsweise die Bestimmung der Orientierung in Bezug auf die Umwelt.
- **Aktive Sensoren** – Aktive Sensoren senden aktive Energie in ihre Umwelt aus und Erfassen anschließend die zurückkehrenden Signale wie dies beispielsweise ein Ultraschallsensor tut.
- **Passive Sensoren** – Diese Sensoren senden nicht aktiv aus sondern erfassen ausschließlich die von Natur aus vorhandenen Signale wie z.B. das einfallende Licht durch eine Kamera.

Die folgenden Tabelle zeigt beispielhaft die Einordnung einiger Sensoren:

	Aktive Sensoren	Passive Sensoren
Propriozeptive Sensoren	(Inkrementalgeber (Photoelektrische Abtastung))	Inkrementalgeber, Neigungssensor, Gyroskop
Exterozeptive Sensoren	Ultraschallsensor, Laserscanner, Infrarotsensor, Radar	Kontaktsensor, Kompass, Kamera, GPS

Tabelle 1: Einordnung von Sensoren

#### 2.1.4 Sensordatenverarbeitung

#### 2.1.5 Antriebsarten

### 2.2 LEGO MINDSTORMS

LEGO MINDSTORMS ist eine seit 1988 existierende Produktserie des Spielwarenherstellers LEGO [vgl. 5, 21]. LEGO MINDSTORMS ermöglicht das Bauen, Programmieren und Steuern verschiedener LEGO Roboter. Diese Roboter bestehen dabei aus gängigen LEGO Teilen die auch in anderen LEGO-Produkten Verwendung finden, sowie speziellen LEGO-Komponenten wie einer zentralen Steuereinheit, Motoren und Sensoren.

#### 2.2.1 Das EV3-System

Der 2013 erschienene EV3 ist das dritte System der LEGO MINDSTORMS Reihe. Die Bezeichnung setzt sich aus EV für Evolution und 3 für die 3 Stufe der LEGO MINDSTORMS-Serie zusammen [vgl. 5, Seite 21].

Im Vergleich zu den Vorgängersystemen verfügt das EV3-System über eine modernere und leistungsfähigere Steuereinheit und auch die anderen elektronischen Komponenten des System wurden an den heutigen Stand der Technik angepasst [vgl. 5, Seite 22].

Die folgende Abbildung X.X zeigt einige der zentralen Komponenten des EV3-Systems, wie die Steuereinheit (EV3-Stein), Motoren und vier Sensoren.

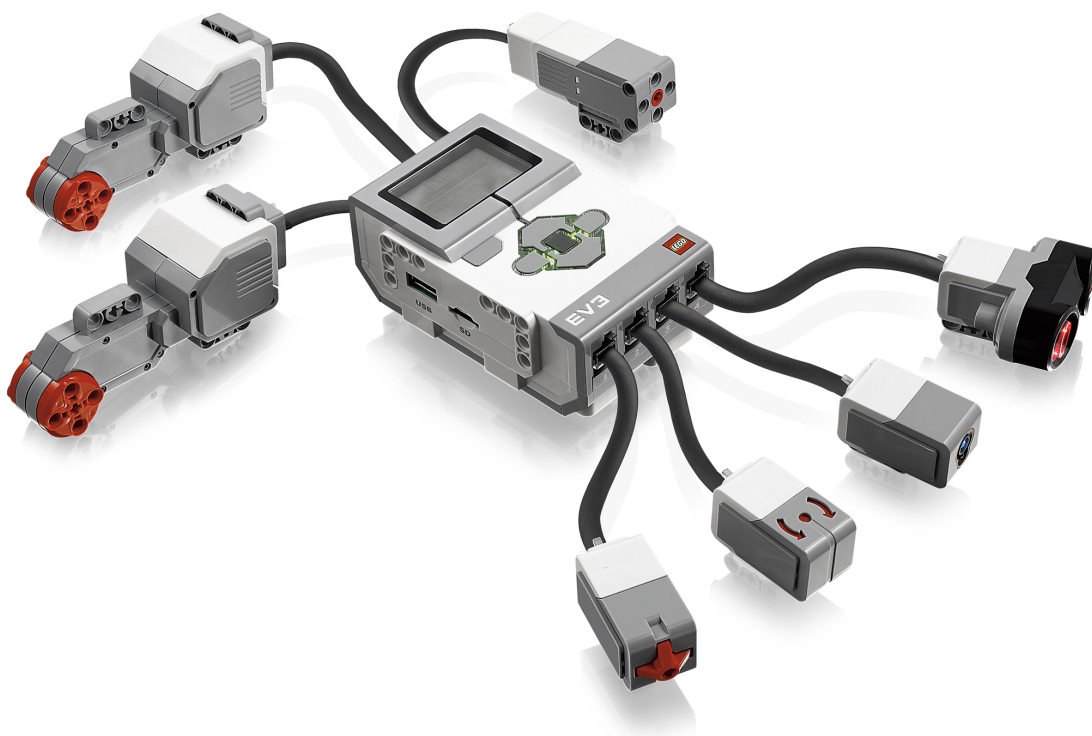


Abbildung 1: Zentrale Komponenten des EV3-Systems

Neben den elektronischen Komponenten gehören auch nicht elektronische Teile wie Verbindungsstücke, Balken und Zahnräder wie sie aus gängigen LEGO Produkten bekannt sind, zum EV3-System. Sie bilden die strukturelle und mechanische Grundlage der Roboter.

Im Folgenden wird auf die elektronischen Komponenten des EV3-Systems näher eingegangen. dieses Projekt eine deutlich größere Relevanz aufweisen.

### 2.2.2 Der EV3-Stein (Steuereinheit)

Die zentrale Komponenten und das Gehirn des LEGO MINDSTORMS EV3-Systems ist die zentrale Steuereinheit kurz (EV3-)Stein oder auch Brick genannt. Bei ihm handelt es sich um eine Computer welcher selbständig Programme ausführen kann. Dazu verfügt der EV3-Stein über ein Linux Betriebssystem und eine spezielle Firmware, die wie die auszuführenden Programme auf einem Flash-Speicher liegen [vgl. 5, 21].

Zur Kommunikation mit dem PC verfügt der EV3-Stein über eine USB- sowie Bluetooth-Schnittstelle. Neben der Kommunikation zu einem Computer kann die USB-Schnittstelle auch für den Zusammenschluss mit einem weiteren EV3-Stein (genannt Daisy Chain) genutzt werden [vgl. 5, Seite 21].

Für den Anschluss von Motoren und Sensoren verfügt der EV3-Stein über 8 Ports, an welche die anderen System-Komponenten müber Kabel mit RJ12-Steckern angeschlossen werden. 4 der Ports dienen für den Anschluss von Motoren, die restlichen 4 Ports für die Abfrage von Sensorwerte [vgl. 5, 21].

Der EV3-Stein besitzt an der Vorderseite ein LCD-Display zur Anzeige von Texten und Grafiken sowie 6 Knöpfe für die Bedienung durch den Benutzer. Display und Knöpfe dienen zur Bedienung der Firmware sowie zur Tätigung von Einstellungen, können aber ebenso durch Programmen angesprochen und ausgewertet werden [vgl. 5, 21].

Die folgende Auflistung zeigt einige Leistungsmerkmale des EV3-Steins [vgl. 5, 4, Seite 23 f., Seite 32].

- Prozessor: ARM9 32Bit, 300 MHz, 16 MB Flash 64MB RAM
- Betriebssystem: Linux
- Sensoranschlüsse: 4x, Analog / Digital bis zu 460,8 Kbit/s
- USB-Schnittstellen: 2x, für Kommunikation zum PC, Daisy Chain, WiFi-Stick, USB-Speichermedium
- SD-Karten-Lesegerät: 1x, für MicroSD-Karte bis 32 GB
- User-Interface: 6 Knöpfe inkl. Beleuchtung
- Display: LCD Matrix, monochrom, 178 x 128 Pixel
- Kommunikation: Bluetooth v2.1, USB 2.0 (Kommunikation zum PC), USB 1.1 (Daisy Chain)

### 2.2.3 Motoren

Das EV3-System verfügt über zwei unterschiedliche Motoren, einen großen Motor und einen mittleren Motor. Bei beiden handelt es sich um Servomotoren mit integriertem Rotationssensor, welche von außen angesteuert und abgefragt werden können [vgl. 4, 92]. Die Motoren lassen sich sehr exakt steuern und ermöglichen so einen synchronen Betrieb mehrerer Motoren [vgl. 5, Seite 29 f.].

Die folgende Tabelle zeigt die wichtigsten Eigenschaften der beiden Motoren.



Eigenschaft / Motortyp	Großer Motor	Mittlerer Motor
Winkelgenauigkeit	1 °	1 °
Umdrehungen	160 bis 170 U/min	240 bis 250 U/min
Drehmoment Rotation	20 Ncm	8 Ncm
Drehmoment Stillstand	40 Ncm	12 Ncm
Gewicht	76g	36g

Tabelle 2: Eigenschaften der EV3-Motoren

#### 2.2.4 Sensoren

Zum EV3-System gehören eine Reihe von verschiedenen Sensoren die es den Robotern ermöglichen Informationen über ihre Umwelt zu sammeln sowie ihre Eigenbewegungen zu erfassen. Im folgenden Abschnitt werden die wichtigsten Sensoren mit ihren Leistungsmerkmalen beschrieben.

**Farbsensor** Der Farbsensor ist ein digitaler Sensor der dazu dient die Lichtintensität sowie verschiedener Farben zu erkennen. Der Sensor kann sowohl aktiv als auch passiv betrieben werden und verfügt dafür über vier unterschiedliche Betriebsmodi [vgl. 4, 101]:

- Farbmodus (passiv) - In diesem Modus erkennt der Sensor 7 verschiedenen Farben.
- RGB-Modus (aktiv) - In diesem Modus sendet der Sensor nacheinander rotes, grünes und blaues Licht aus, je nachdem zu welchem Anteil ein Gegenstand die einzelnen Farben reflektiert wird die Farbe des Gegenstands ermittelt.
- Rotlicht-Modus (aktiv) - Bei diesem Modus wird Rotlicht ausgesendet und die Intensität des reflektierten Lichts gemessen.
- Umgebungslicht-Modus (passiv) - Bei diesem Modus wird die Intensität des in das Sensorfenster eindringende Umgebungslichts gemessen.

Eigenschaften:

- Erkennung der Farben: keine Farbe, Schwarz, Blau, Grün, Gelb, Rot, Weiß, Braun
- Abtastrate: 1.000 Hz
- Entfernungsreichweite: 15 bis 50 mm

Durch diesen Sensor wird es beispielsweise möglich den Roboter einer farbigen Linie auf dem Boden zu folgen.

**Ultraschallsensor** Diese aktive Sensor verwendet für den Menschen unhörbaren Ultraschall um die Entfernung von Objekten zu ermitteln. Der Sensor emittiert dazu Ultraschall und misst die Laufzeit der Schallwellen, wenn diese von einem Objekt reflektiert werden, aus der Laufzeit kann dann die Entfernung ermittelt werden. Der Sensor verfügt über zwei unterschiedliche Betriebsmodi [vgl. 5, 32 f.]:

- Messen - In diesem Modus sendet der Sensor Ultraschall aus um die Entfernung von Objekten zu ermitteln.
- Scannen - In diesem passiven Modus emittiert der Sensor selbst keinen Ultraschall, sondern er reagiert auf »fremden« Ultraschall und kann so einen anderen aktiven Ultraschallsensor erkennen.

Eigenschaften:

- Genauigkeit:  $\pm 1$  cm
- Messbereich: 3 cm bis 250 cm

**Berührungssensor** Der Berührungssensor ist ein einfacher mechanischer Sensor. Wird der Knopf am Ende des Sensors gedrückt wird dies registriert. Trotz der Einfachheit dieses Sensors ist dieser dennoch sehr nützlich, da er beispielsweise die Kollision des Roboters mit einem Hindernis erkennen kann [vgl. 5, 33].

**Kreiselsensor (Gyroskop)** Der Kreiselsensor ermöglicht es Drehbewegungen um eine Achse über Rotationsgeschwindigkeit und Drehwinkel zu messen. Dadurch wird es möglich die Eigenbewegung des Roboters oder einer Roboterkomponente zu registrieren [vgl. 5, 33].

Eigenschaften:

- Genauigkeit:  $\pm 3^\circ$  (bei einer  $90^\circ$  Drehung)
- Geschwindigkeit: maximal 440 Grad/Sekunde
- Abtastrate: 1.000 Hz

**Rotationssensor (Integriert)** Wie bereits im Abschnitt X.X dargelegt verfügen die beiden Motortypen über integrierte Rotationssensoren die es ermöglichen, die Umdrehungen der Motoren auszulesen. Durch diese Sensoren ist es möglich durch Odometrie Rückschlüsse über die Bewegung bzw. Position des Roboters zu schließen.

Eigenschaften:

- Genauigkeit:  $1^\circ$

- Umdrehungen: Motorabhängig

Neben den hier vorgestellten Sensoren existiert noch ein Infrarotsensor, welcher in Verbindung mit einer Infrarotfernsteuerung dazu dient einen EV3-Roboter fernzusteuern.

### 2.2.5 Programmierung

Für die Programmierung der LEGO MINDSTORMS Produkte gibt es eine Reihe unterschiedlicher Programmiersprachen und -umgebungen. Die hauseigene LEGO-Software zur Programmierung des EV3 richtet sich an Einsteiger. Sie ermöglicht es über eine grafische Oberfläche via vorgefertigter Programmabläufe welche durch grafische Blöcke repräsentiert werden den EV3 zu programmieren.<sup>1</sup>

Die Abbildung X.X gibt einen Überblick über verschiedene für den EV3 verfügbare Programmiersprachen sowie ihre Vor- und Nachteile.

**leJOS** Das LEGO Java Operating System abgekürzt leJOS ist ein Framework, das es ermöglicht den EV3 mit der Programmiersprache Java zu programmieren. Das leJOS-Projekt wurde 1999 gegründet und sämtliche Komponenten (wie auch Java) sind kostenlos verfügbar [vgl. 4, 21 f.].

leJOS bietet eine schlanke Java Virtual Machine (JVM) für den EV3-Stein sowie eine Klassenbibliothek mit welcher die Komponenten des EV3 (Motoren, Sensoren etc.) angesprochen werden können. Installiert wird leJOS auf einer bootbaren microSD-Karte und kann anschließend davon gestartet werden, ohne die auf dem EV3 vorhandene LEGO-Software zu löschen oder zu verändern [vgl. 4, 23 f.].

Durch leJOS ist es möglich den EV3 mit Hilfe der Hochsprache Java zu programmieren womit eine mächtige Programmiersprache zur Verfügung steht und die Vorteile der Objektorientierung für den EV3 genutzt werden können. leJOS bietet eine umfangreiche

Eigenschaft / Programmiersprache	leJOS	EV3-Software	RobotC	NEPO
Installation	+	++	+	+++
Handhabung	+	++	+	++
Kosten	kostenlos	kostenlos	49\$	kostenlos
Einstieg	0	++	+	+++
Funktionsumfang	++	+	++	++

0 = neutral; + = gut; ++ = sehr gut; +++ = hervorragend

Tabelle 3: Eigenschaften der EV3-Motoren

Klassenbibliothek sowie gut dokumentierte API was unter anderem die Integration von weiteren Sensoren etc. erleichtert [vgl. 4, 23 f.]. Im folgenden sind einige Features die leJOS bietet aufgelistet:

<sup>1</sup>[vgl. 4, 25 f.]

- Objektorientierte Programmierung mit Java
- Die meisten Klassen der Pakete `java.lang`, `java.util` und `java.io`
- Rekursion
- Synchronisation
- Multithreading
- Exceptions
- Vollständige Bluetooth unterstützung
- Umfangreiche Klassenbibliothek zum Steuern und Auslesen der EV3-Komponenten
- High-Level-Robotik-Tasks (Navigation, Localization etc.)

<sup>2</sup>[vgl. 2, Unterschiede und Vergleich native Apps vs. Web Apps]

Von dieser Entwicklung finden sich viele Vertreter mit der Unterstützung diverser Frameworks. Das populärste unter ihnen ist aktuell AngularJS von Google, was auf JavaScript basiert. In Kombination mit anderen Webtechnologien, wie glshtml und CSS lassen sich performante web Apps entwickeln.

### **2.3.3 Hybride Apps**

Die Entwicklung von hybride Apps vereint die beiden Entwicklungen von native und web. Sie besteht dabei aus einem nativen Rahmen, in der eine web App zur Ausführung kommt, diese besitzt entsprechende Zugriffsrechte auf Hardwareschnittstellen, um diese mit Application Programming Interfaces (APIs) anzusprechen.<sup>3</sup>

Diese Entwicklung ist aktuell noch sehr jung, jedoch stechen hier bereits verschiedene Vertreter hervor. Der populärste unter ihnen ist Ionic von Drifty, welches auf Apache Cordova als Basis zurückgreift. In Kombination mit AngularJS, TypeScript und anderen Webtechnologien lässt sich die web App entwickeln und auf einem beliebigen Gerät unter einem nativen Browser ausführen. Es unterstützt dabei verschiedenste Betriebssystem, wie Android, iOS und Windows. Diese Entwicklungen können dabei meist nicht nur mobil, sondern unter anderem auf weiteren Systemen, wie stationäre bereitgestellt werden.

### **2.3.4 Plattformübergreifende Entwicklung**

Um die Entwicklung von Apps einfach zu halten, verwenden immer mehr Entwickler die Form der plattformübergreifenden Entwicklung. Dadurch lässt sich die App unabhängig des Betriebssystems entwickeln und kann somit eine größere Menge von Nutzern erreichen. Diese Entwicklung greift dabei meist auf plattformübergreifende Konzepte, wie eine native Laufzeitumgebung, oder Browser zurück, um darin die App auszuführen. Der große Vorteil in dieser Entwicklung, liegt in der Wiederverwendbarkeit des Quellcodes und der verbesserten Wartbarkeit, da hier lediglich ein Projekt gewartet werden muss und der Quellcode für viele Betriebssysteme übernommen werden kann. Zur plattformübergreifenden Entwicklung wurden die letzten Jahre viele Ansätze mit verschiedenen Frameworks entwickelt. Beispiele hierfür sind Ionic, Unity, Qt oder Xamarin.

---

<sup>3</sup>[vgl. 3, Native App, Web App und Hybrid App im Überblick]

### 2.3.5 Xamarin

Xamarin ist ein Framework zur Entwicklung von nativen plattformübergreifenden Apps. Dabei baut Xamarin auf Mono, einer opensource Version des .NET Framework, welches auf den .NET ECMA Standards basiert.<sup>4</sup> Um nativen Quellcode auf den verschiedenen Systemen auszuführen, setzt Xamarin auf verschiedene Softwarekomponenten, um aus einem mit .NET entwickelten Projekt nativen Quellcode zu erzeugen.

Für iOS Systeme verwendet Xamarin den AOT (Ahead-of-Time) Compiler, um aus einem Xamarin.iOS Projekt ARM Maschinencode zu erzeugen, der so entsprechend schnell ausgeführt werden kann.

Bei Android nutzt Xamarin die IL, um JIT nativen Quellcode für die entsprechende Hardware zu compilieren und die App auszuführen.



Abbildung 3: Xamarin

### 2.3.6 Mono

Mono ist eine opensource Laufzeitumgebung für Linux Betriebssysteme, um Anwendungen auszuführen, die auf dem .NET Framework basieren. Dabei greift Mono auf Standards des CLI und ECMA von C# zurück. Gestartet wurde das Projekt durch die Firma Novell und aktuell weiterentwickelt von Microsoft und wird dadurch auf gleichem Stand wie .NET gehalten.



Abbildung 4: Mono

### 2.3.7 .NET Framework

Das .NET Framework ist eine Laufzeitumgebung für .NET Anwendungen, die verschiedene Dienste bereitstellt. Es besteht aus zwei Hauptkomponenten, der CLR, die eine Speicherverwaltung und verschiedene Systemdienste bereitstellt, sowie der .NET Bibliothek. Um Anwendungen für .NET zu entwickeln, wird die entsprechende Version von .NET Framework auf dem System benötigt. Als Programmiersprache ist der Entwickler weitgehend unabhängig, der Quellcode muss jedoch die CLI-Spezifikationen erfüllen. Dafür eignen sich unter anderem die Programmiersprachen von Microsoft, wie VisualBasic, C#, VisualF# und C++.

---

<sup>4</sup>[vgl. 1, Introduction to Mobile Development - Xamarin]

## 2.4 TCP-Kommunikation

Das Transmission Control Protocol (TCP) ist ein Transportprotokoll und ermöglicht einen Datenaustausch zwischen kommunizierenden Anwendungsinstanzen in einer Ende-zu-Ende-Beziehung zwischen. TCP ist als Transportprotokoll auf in der 4 Schicht des OSI-Modells angesiedelt und basiert auf dem Internetprotokoll (IP) mit dem zusammen es als Namensgeber der TCP/IP-Protokollfamilie (Internetprotokollfamilie) dient. TCP ist ein offenes, frei verfügbares und weit verbreitetes Protokoll. Als Mitglied der Internetprotokollfamilie ist TCP neben UDP das Transportprotokoll, auf dem die meisten Anwendungen im Internet basieren [GD189].

### 2.4.1 Grundlegendes

Als verbindungsorientiertes Protokoll sorgt TCP für die Erzeugung und Erhaltung einer gesicherten Ende-zu-Ende-Verbindung zwischen zwei Anwendungsprozessen. TCP arbeitet paketvermittelt d.h. überträgt Daten Paketweise und ist ein zuverlässiges Protokoll. Durch diese Eigenschaften stellt TCP sicher dass, Daten

- nicht verloren gehen
- nicht verändert werden
- nicht dupliziert werden
- in der richtigen Reihenfolge eintreffen

Zur Gewährleistung einer vollständigen Übertragung sowie der Integrität der gesendeten Daten nutzt TCP Prüfsummen, Bestätigungen, Zeitüberwachungs- und Nachrichtenwiederholungsmechanismen sowie Sequenznummern für die Reihenfolgeüberwachung und das Sliding Windows Prinzip zur Flusskontrolle [GD].

TCP nutzt prinzipiell folgende Protokollmechanismen:

- Drei-Wege-Handshake-Verbindungsauf- und -abbau
- Positives, kumulatives Bestätigungsverfahren mit Timerüberwachung für jede Nachricht
- Implizites negatives Bestätigungsverfahren (NAK-Mechanismus): Bei drei ankommenden Duplikat-ACK-PDUs wird beim Sender das Fehlen des folgenden Segments angenommen. Ein sog. Fast-Retransmit-Mechanismus führt zur Neuübertragung des Segments, bevor der Timer abläuft.
- Pipelining
- Go-Back-N zur Übertragungswiederholung
- Fluss- und Staukontrolle



### 2.4.2 Nagle-Algorithmus

Nagle-Algorithmus (RFC 896 und RFC 1122) ist ein Algorithmus der der Optimierung dient und der bei allen TCP-Implementierungen verwendet wird. Der versuchte Nagle-Algorithmus aus Optimierungsgründen zu verhindern, dass viele kleine Nachrichten gesendet werden, da dies schlecht für die Netzauslastung ist [GD198].

Dazu werden mehrere Nachrichten zusammengefasst und gebündelt versendet, dies geschieht nach folgendem Prinzip:

- Erhält der TCP-Endpunkt Daten vom Anwendungsprozess wird zunächst nur das erste Datenpaket gesendet und die restlichen Daten werden im Sendepuffer gesammelt.
- Danach werden weiteren Daten so lange im Sendepuffer gesammelt bis alle zuvor gesendeten Datenpakete vom Empfänger bestätigt wurden oder so viele Daten im Sendepuffer liegen das die eingestellte Segmentgröße erreicht ist und ein volles Datenpaket gesendet werden kann.

Dieses Verfahren sorgt zwar für eine gute Netzauslastung da das Verhältnis von Nutzdaten zu Overhead (TCP-Header etc.) steigt jedoch ist dies allerdings nicht für alle Anwendungsszenarien optimal da es die Latenz erhöht. Insbesondere bei Anwendungen die eine unmittelbare Antwort der Gegenstelle benötigen wie SSH- oder Telnet-Anwendung sorgt dies für Verzögerungen. In diesem Fall ist es besser den Nagle-Algorithmus auszuschalten. [GD198 f.]

### 2.4.3 Kommunikationsablauf

Da es sich bei TCP um ein verbindungsorientiertes Protokoll handelt gliedert sich die Kommunikation in die drei Phasen Verbindungsaufbau, Datenaustausch und Verbindungsabbau. Bevor Daten übertragen werden können muss die Verbindung durch den Verbindungsaufbau initiiert und nach Beendigung der Datenübertragung wieder abgebaut werden.

**Client & Server** Der Verbindungsaufbau einer Kommunikation erfolgt bei TCP nach dem Client-/Server-Paradigma, d.h. einer der Teilnehmern agiert als Server und wartet auf einen Verbindungsaufbau durch den Client welchen der andere Teilnehmern darstellt. Nach dem Verbindungsaufbau haben die beiden Rollen jedoch keine Bedeutung mehr und die beide Teilnehmern sind sowohl bei der Datenübertragung als auch beim Verbindungsabbau gleichberechtigt.

#### Verbindungsaufbau

Der Verbindungsaufbau bei TCP basiert auf dem Three-Way-Handshake. Dabei schickt

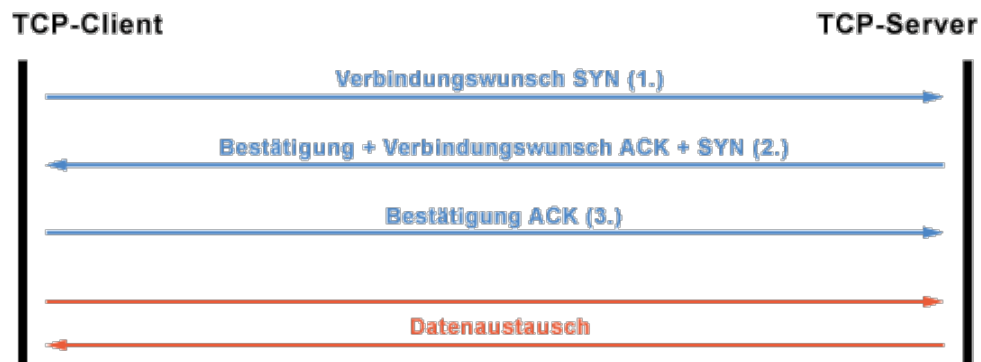


Abbildung 5: TCP Verbindungsaufbau

der schickt der Client einen Verbindungswunsch (SYN) an den Server. Der Server bestätigt den Erhalt der Nachricht (ACK) und äußert seinerseits einen Verbindungswunsch (SYN) welchen der Client nach Erhalt der Nachricht bestätigt (ACK). Nach Ablauf dieses gegenseitigen Anfrage- und Bestätigungsvorgangs ist die Verbindung initiiert und der Datenaustausch zwischen den Teilnehmern kann beginnen [EK].

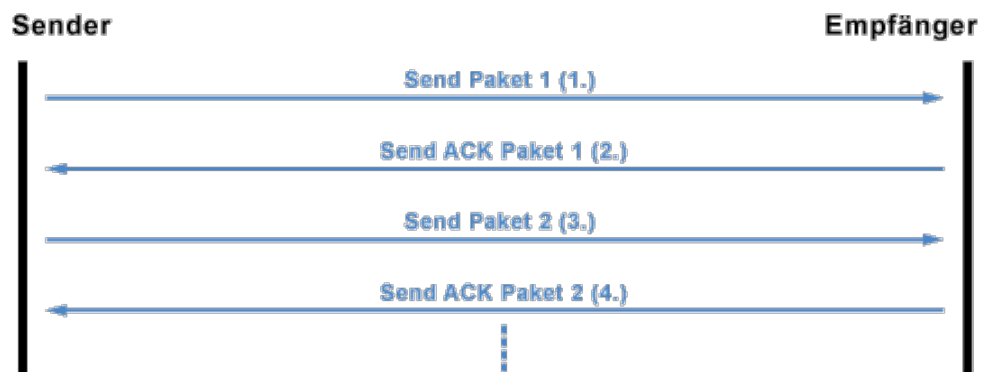


Abbildung 6: TCP Datenaustausch

## Datenaustausch

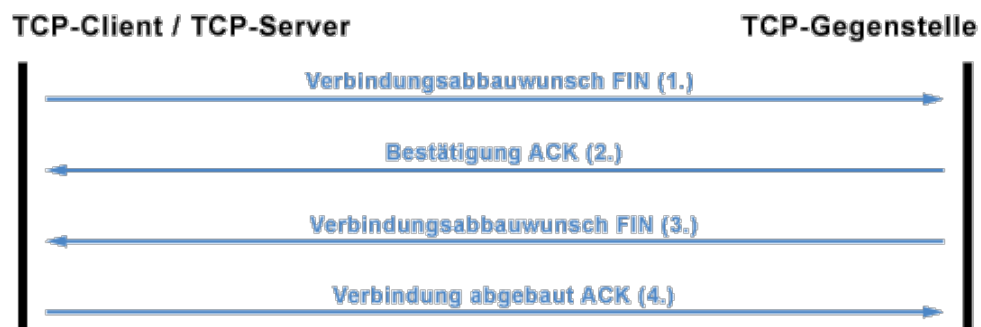


Abbildung 7: TCP Verbindungsabbau

**Verbindungsabbau** Nach Abschluss der Datenübertragung wird von einer Seite (egal von welcher) ein Verbindungsabbau initiiert. Dazu dient einen etwas modifizierten Drei-

Wege-Handshake-Mechanismus. Jede der beiden Verbindungsrichtungen der Vollduplex-Verbindung wird abgebaut, d.h. beide Seiten bauen ihre „Senderichtung“ ab. Die initiiierende Seite schickt zuerst einen Verbindungsabbauwunsch (FIN). Die Gegenstelle bestätigt den Erhalt der Nachricht (ACK) und schickt ebenfalls einen Verbindungsabbauwunsch (FIN) woraufhin sie von der Gegenstelle noch mitgeteilt bekommt, dass die Verbindung abgebaut ist (ACK).

#### **2.4.4 Socket-Programmierung**

Als Transportzugriffsschnittstelle für die TCP-basierte Kommunikation dient die Socket-Schnittstelle. Obwohl es sich bei TCP ein paketvermittelltes Protokoll handelt der Anwendung eine Strom-orientierte Kommunikation, die Daten werden also von einem Anwendungsprozess Byte für Byte in einem Bytestrom geschrieben und TCP sorgt anschließend um den Aufbau von Segmenten, die dann übertragen werden. Andere Transportdienste erwarten ihre Daten in festen Blöcken [GD].

### **2.5 Java**

#### **2.5.1 Grundlagen**

#### **2.5.2 Java Runtime Environment**

## **3 Theoretische Grundlagen**

### **3.1 Schwarmverhalten**

#### **3.1.1 Allgemein**

#### **3.1.2 Vorbilder aus dem Tierreich**

#### **3.1.3 Szenarien**

#### **3.1.4 Algorithmen**

## 4 Projektorganisation

### 4.1 Projektablaufplan

## 5 Konzeption

In diesem Kapitel werden die Anforderungsdefinitionen des Projektes, mit Spezialisierung auf die verschiedenen Use Cases beschrieben.

### 5.1 Anforderungsdefinitionen

In diesem Abschnitt wird auf die Funktionalitäten und Use Cases des Projektes eingegangen.

#### 5.1.1 Softwarearchitektur

### 5.2 Use Cases

#### 5.2.1 Connect

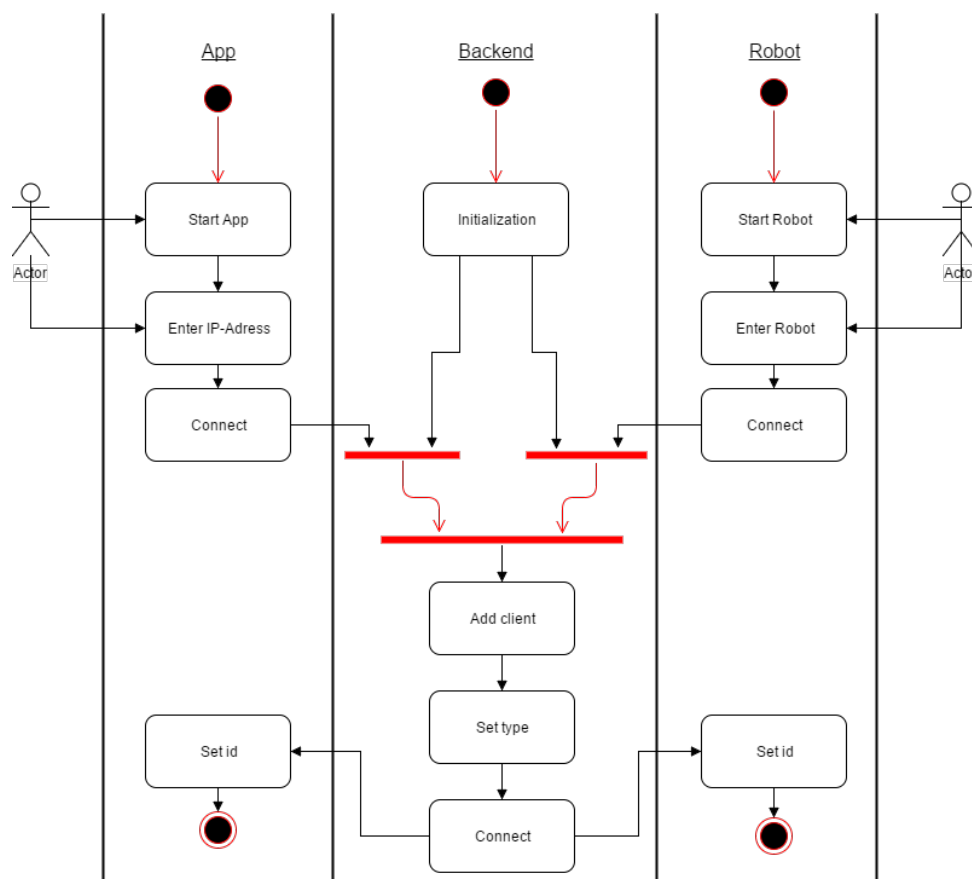


Abbildung 8: Connect

Im Use Case Connect wird eine erste Verbindung durch die Eingabe der IP-Adresse zum Backend aufgebaut. Dabei sendet die Komponente, ob Roboter oder App eine Abbildung seiner selbst als Objekt dem Backend. Daraufhin startet das Backend die Verbindung indem es der Komponente entsprechende Verbindungskommandos zusendet. Sobald eine

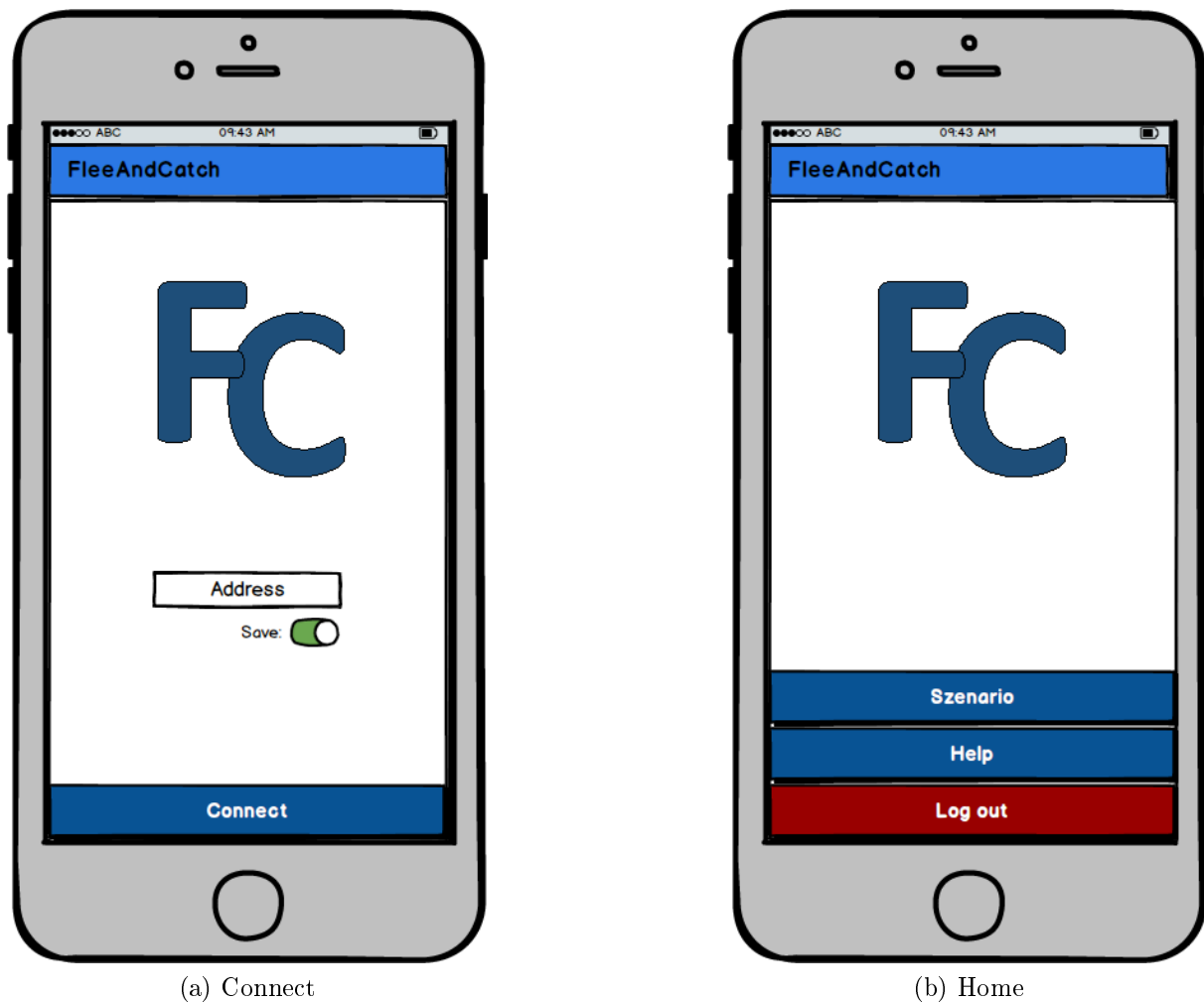


Abbildung 9: Connection

Reaktion in einer festgelegten Zeit erfolgt, akzeptiert das Backend die Verbindung und sendet die entsprechende Id für die Komponente. Ab diesem Moment ist die Komponente verbunden und ein Robot für Aktionen entsprechend verfügbar. Die Verbindungsinitialisierung dient hierbei der Verkürzung der Reaktionszeit, die bei einem Roboter sonst entsprechend hoch wäre.

### 5.2.2 Synchronization

Im Use Case Synchronization werden Daten entsprechend des gesetzten Typen zwischen den Komponenten übertragen. Dabei können einerseits die Roboter als Objekte, oder ganze Szenarien übertragen werden. Dies dient zur Gegenseitigen Synchronisierung der Daten.

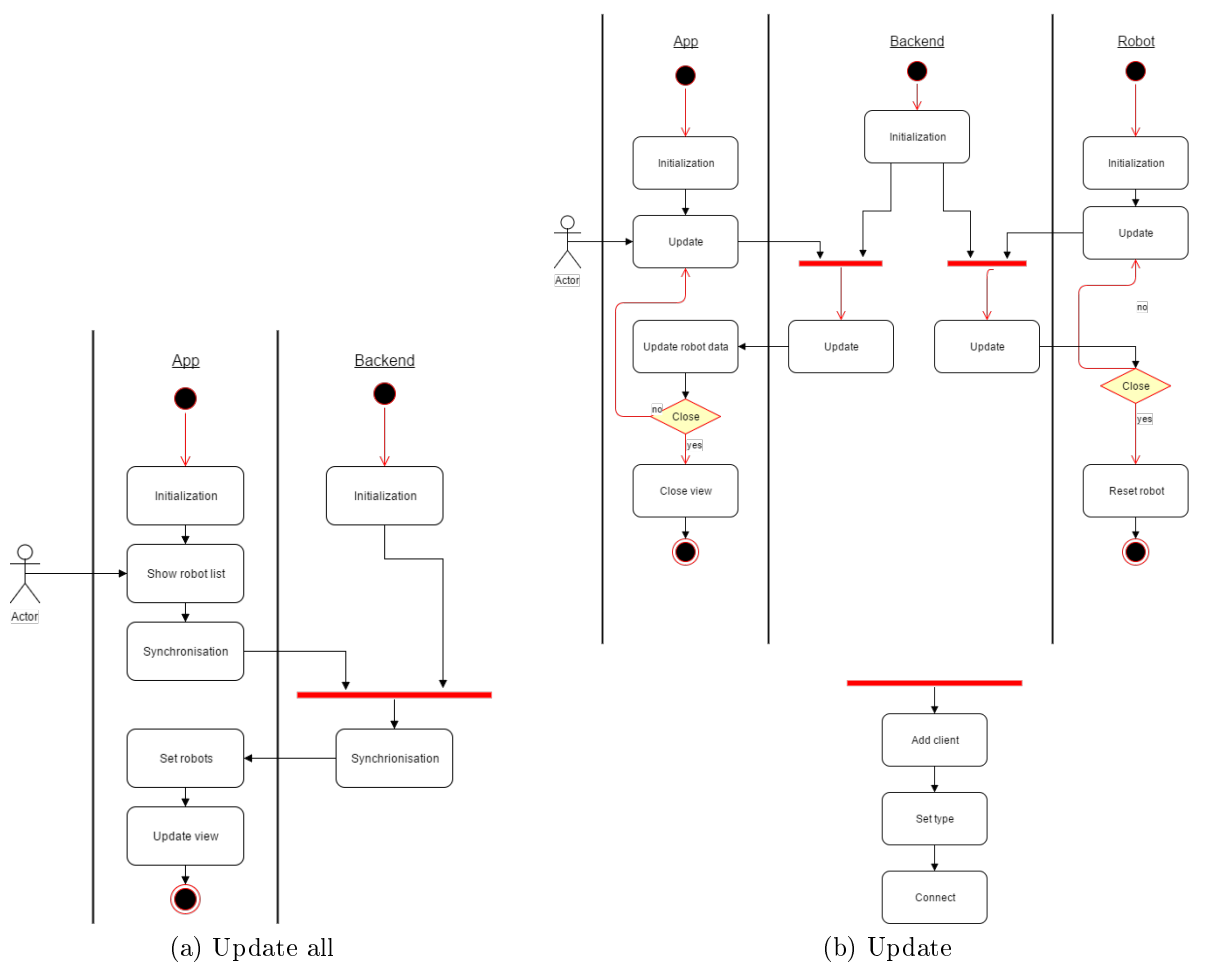


Abbildung 10: Synchronization

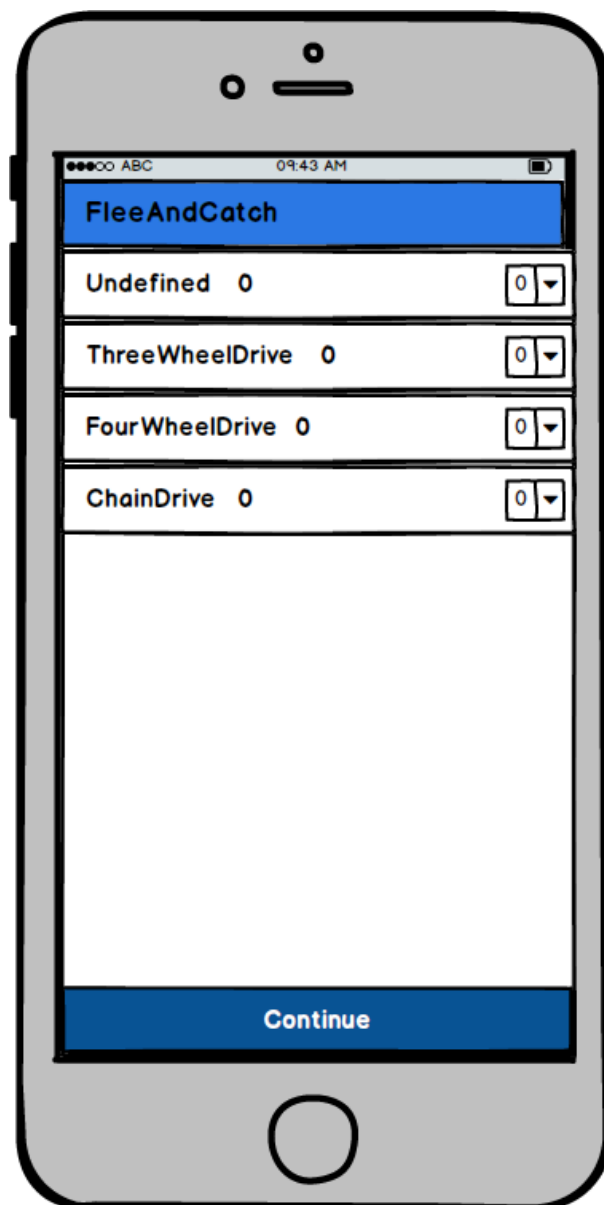


Abbildung 11: Robot list



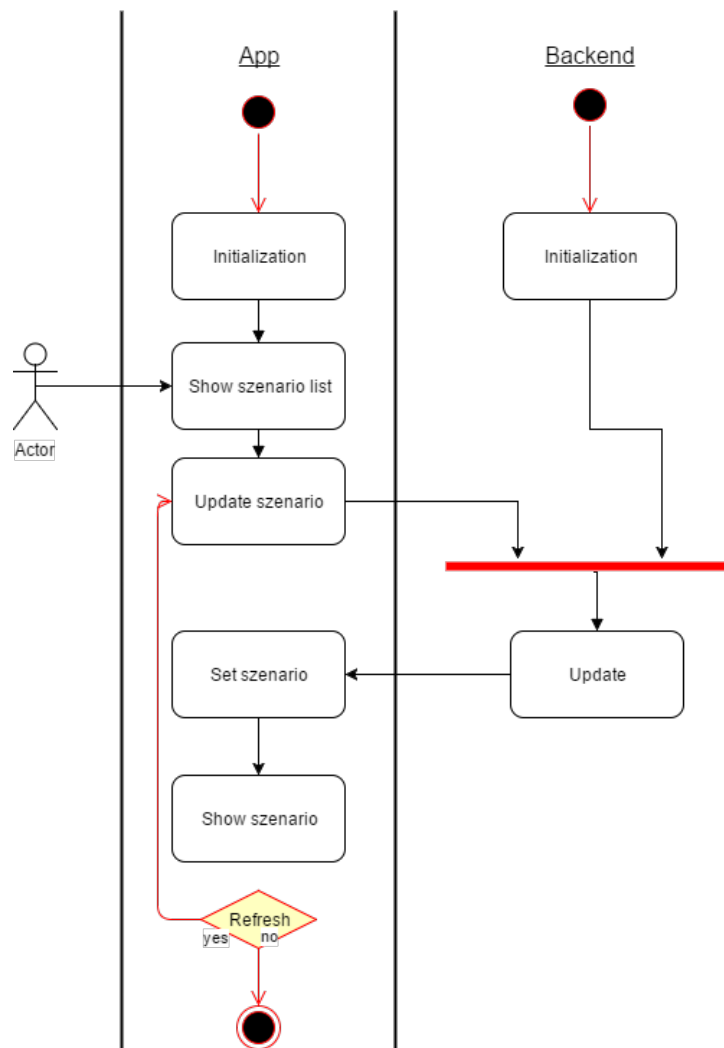


Abbildung 12: Spectator

### 5.2.3 Szenario

### 5.2.4 Exception

## 5.3 Kommunikation

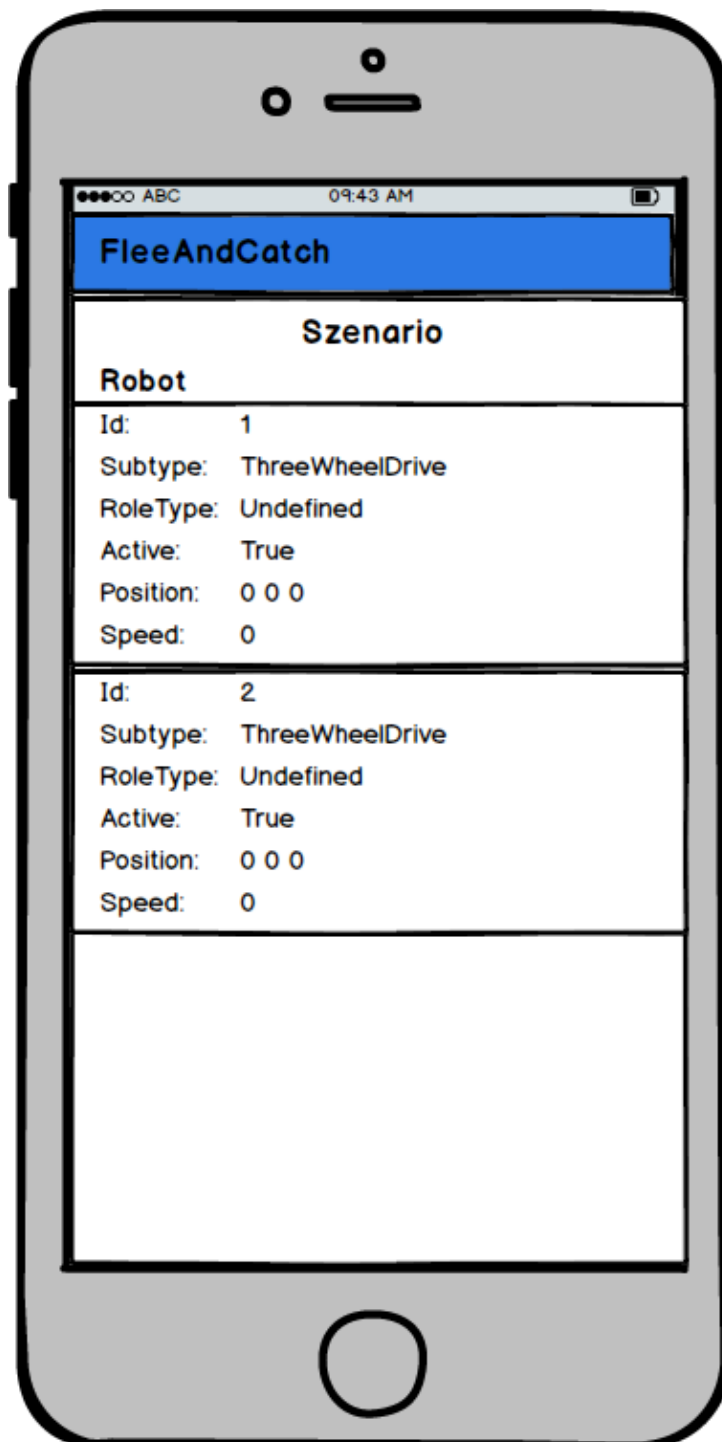


Abbildung 13: Spectator

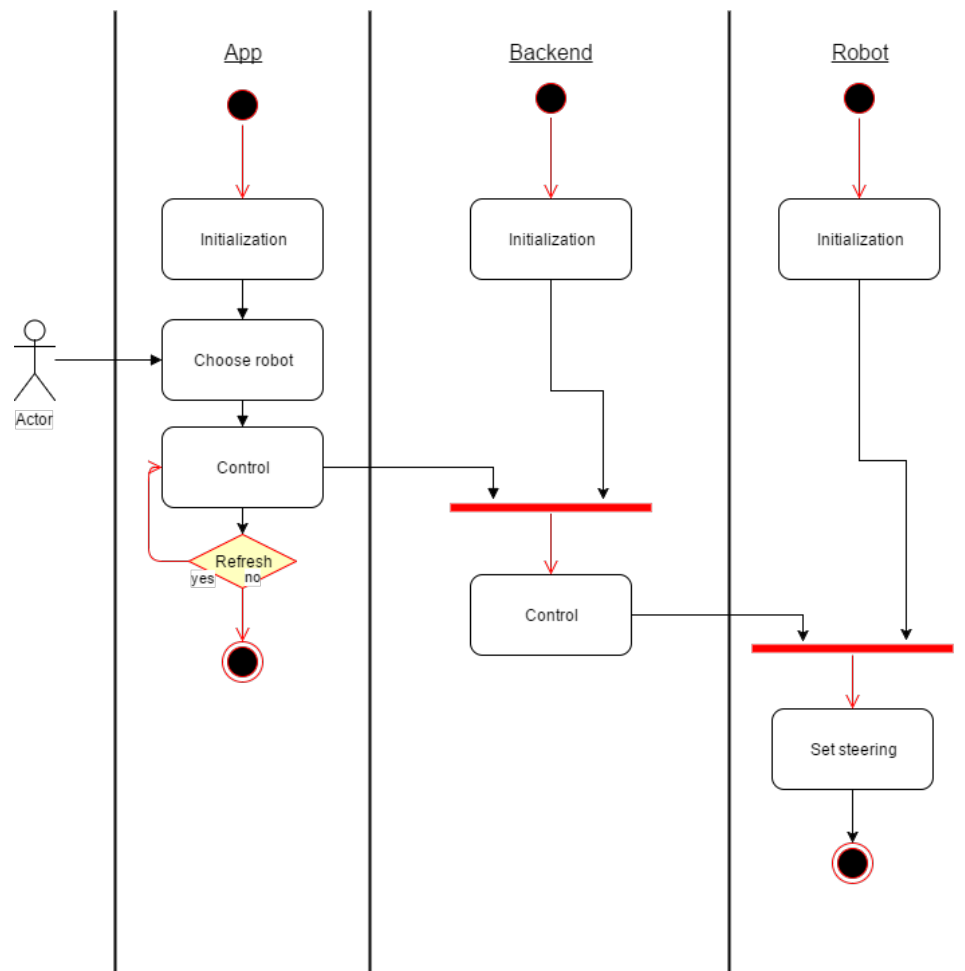


Abbildung 14: Control

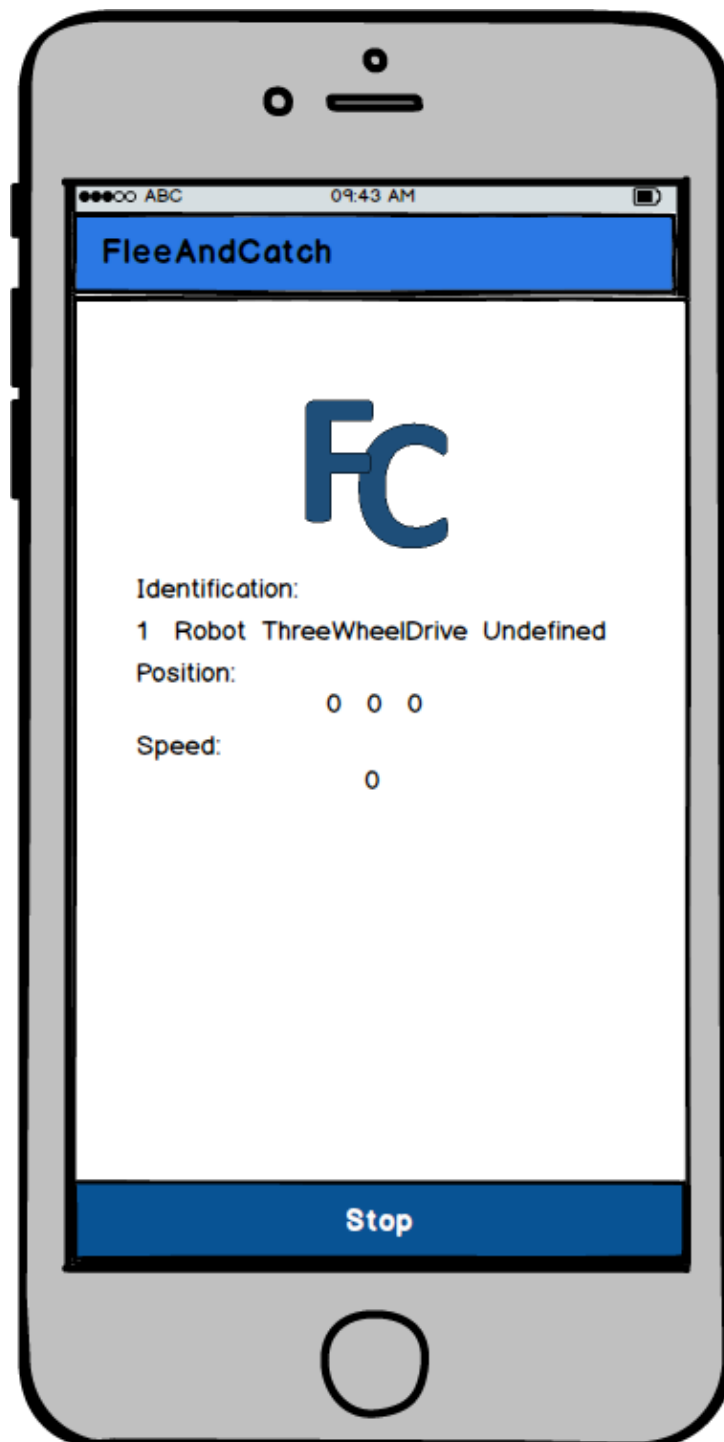


Abbildung 15: Control

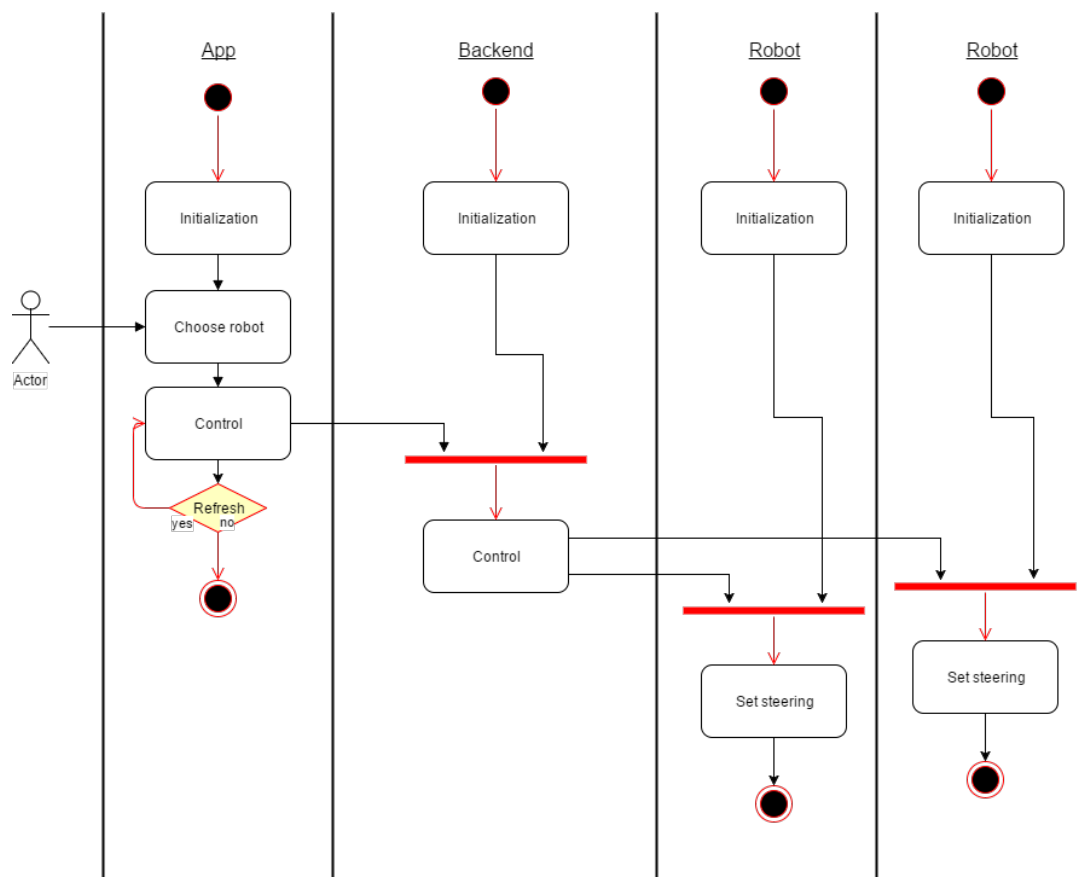


Abbildung 16: Synchron

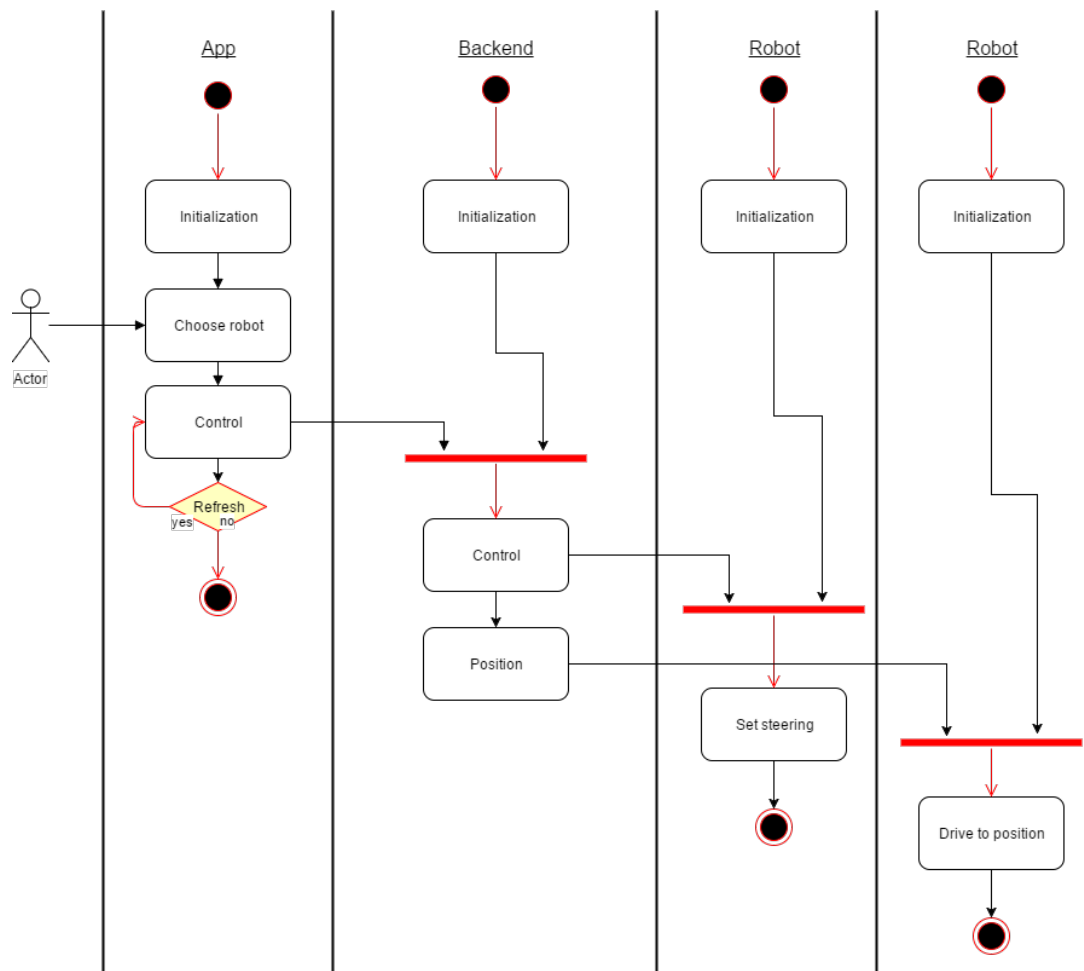


Abbildung 17: Follow

## 6 Lösungsansatz

## 7 Umsetzung



## 8 Evaluation

## 9 Zusammenfassung und Ausblick

---

## Literatur

- [1] Introduction to mobile development - xamarin. URL [https://developer.xamarin.com/guides/cross-platform/getting\\_started/introduction\\_to\\_mobile\\_development/](https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/).
- [2] Daniel Würstl. Unterschiede und vergleich native apps vs. web apps. URL <http://www.app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/107-unterschiede-und-vergleich-native-apps-vs-web-apps>.
- [3] Petra Riepe. Native app, web app und hybrid app im überblick: Warum native wenn es auch hybrid geht? URL <http://www.computerwoche.de/a/warum-native-wenn-es-auch-hybrid-geht,3096411>.
- [4] Maximilian Schöbel, Thorsten Leimbach, and Beate Jost. *Roberta - EV3-Programmieren mit Java*. Roberta - Lernen mit Robotern. Fraunhofer-Verl., Stuttgart, 2015. ISBN 9783839608401.
- [5] Matthias Paul Scholz, Beate Jost, and Thorsten Leimbach. *Das EV3 Roboter Universum: Ein umfassender Einstieg in LEGO MINDSTORMS EV3 : mit 8 spannenden Roboterprojekten*. 1. auflage edition, 2014. ISBN 3-8266-9473-2.

## Anhang