DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Konzeption und Implementierung eines Sachwarmverhaltens 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

20. März 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	

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

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

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Abstract

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Zusammenfassung

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface.

App Application.

Glossar

Application Eine Application ist ein ausführbares Programm für mobile Geräte, wie Smartphones oder Tablets.

Application Programming Interface Eine API ist eine Programmierschnittstelle, die die Anbindung von Software ermöglicht.

Abbildungsverzeichnis

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Tabellenverzeichnis

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

2.1.1 Grundlagen

2.1.2 Mobile Roboter

2.1.3 Antriebsarten

2.1.4 Sensorik

2.2

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

2.2.1 Das EV3-System

Der 2013 erschienene EV3 ist das dritte System der Reihe. Die Bezeichnung setzt sich aus EV für Evolution und 3 für die 3 Stufe der -Serie zusammen [vgl. **?** , 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. **?** , 22].

Die folgende Abbildung X.X zeigt einige der zentralen Komponeten des EV3-Systems, wie die Steuereinheit (EV3-Stein), Motoren und vier Sensoren.
Neben den elektronischen Komponeten gehören auch nicht elektronische Teile wie Verbindungsstücke, Balken und Zahnräder wie sie aus gängigen Produkten bekannt sind, zum EV3-System. Sie bilden die strukturelle und meschanische Grundlage der Roboter.

Im Folgenden wird auf die elektronischen Komponenten des EV3-Systems näher eingegangen. dieses Projekt eine deutlich größere Relevanzu 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. **?** , 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

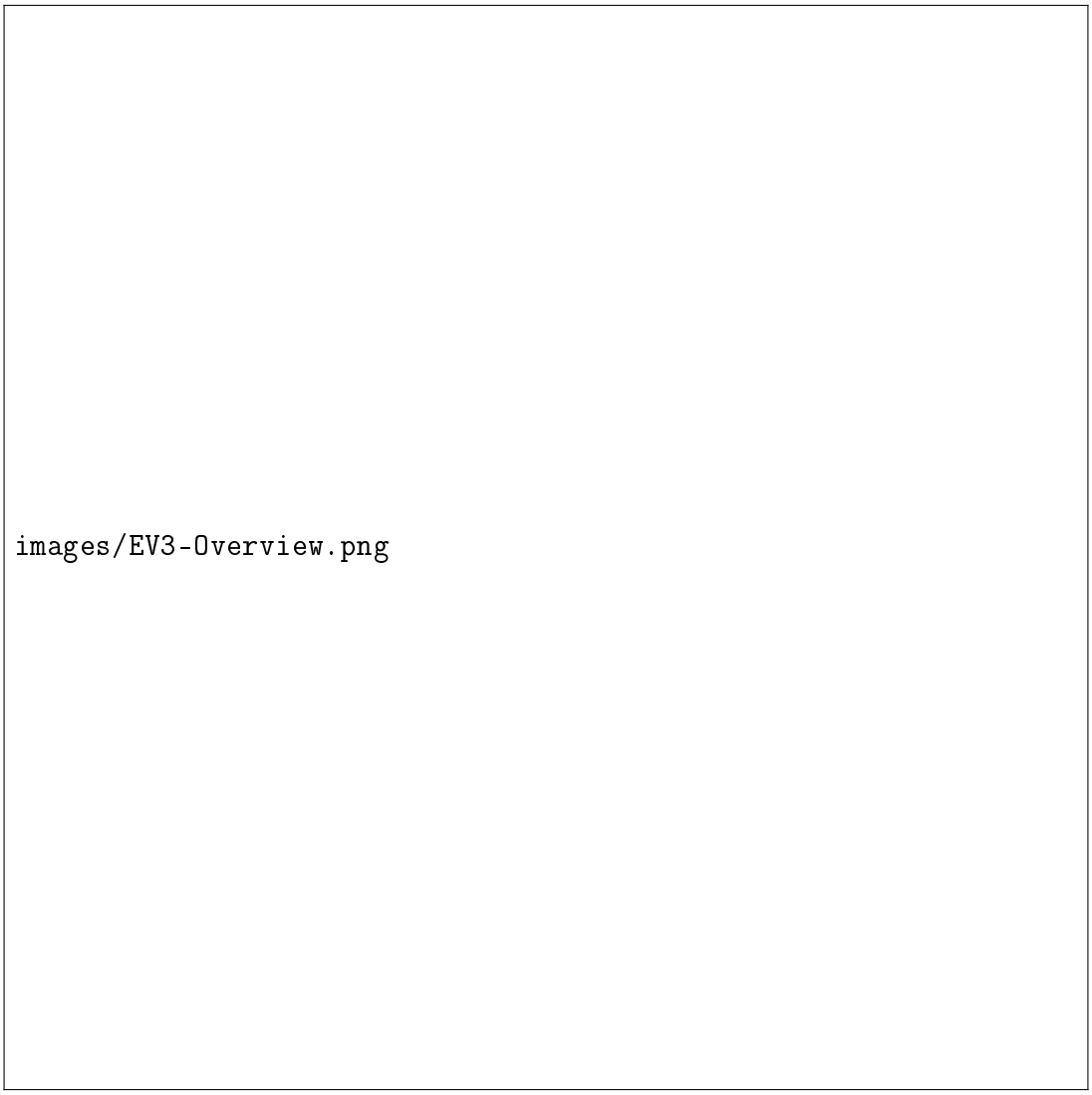


Abbildung 1: Zentrale Komponenten des EV3-Systems

auch für den Zusammenschluss mit einem weiteren EV3-Stein (genannt Daisy Chain) genutzt werden [vgl. **?** , 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. **?** , 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. **?** , 21].
Die folgende Auflistung zeigt einige Leistungsmerkmale des EV3-Steins [vgl.][23]EV3RU[32]Roberta.

- 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 Servormotoren mit integriertem Rotationssensor, welche von außen angesteuert und abgefragt werden können [vgl. **?** , 92]. Die Motoren lassen sich sehr exakt steuern und ermöglichen so einen synchronen Betrieb mehrerer Motoren [vgl. **?** , 29].
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 1: 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 Frabsensor 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. **?** , 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 balues Licht aus, je nachdem zu welchem Anteil ein Gegenstand die einzelnen Farben reflektiert wird die Frabe 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 Intesnsität des in das Sensorfenster eindringende Umgebungslichts gemessen.

Eigenschaften:

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

Durch diesen Sensor wird es beispielsweise möglich den Roboter einer frabigen 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 emmitiert dazu Untraschall und misst die Laufzeit der Schallwellen, wenn diese von einem Objekt reflektiert werden, aus der Laufzeit kann dann die Entfernung ermittelt werden. Der Senors verfügt über zwei unterschiedliche Betriebsmodi [vgl. **?** , 32]:

- 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 Untraschall, sondern er reagiert auf »fremden« Ultraschall und kann so einen anderen aktiven Ultraschallsensor erkennen.

Eigenschaften:

- Genauigkeit: +/- 1 cm
- Messbereich: 3 cm bis 250 cm

Berührungssensor Der Berührungssensor ist ein einfacher mechanischer Sensor. Wird der Knopf am Ende des Senors 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. **?** , 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. **?** , 33].

Eigenschaften:

- Genauigkeit: +/- 3° (bei einer 90° Drehung)
- Geschwindigkeit: maximal 440 Grad/Sekunde
- Abtastrate: 1.000 Hz

Rotationssensor (Integiert) Wie bereits im Abschnitt X.X dargelegt verfügen die beiden Motortypen über initgrierte 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°
- Umdrehungen: Motorabhängig

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

2.2.5 Programmierung

Für die Programmierung der Produkte gibt es eine Reihe unterschiedlicher Programmiersprachen und -umgebungen. Die hauseigene -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 [vgl. **?** , 25].
Die Abbildung X.X gibt einen Überblick über verschiedene für den EV3 verfügbare Programmiersprechen 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. **?** , 21].
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. ? , 23].
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 2: Eigenschaften der EV3-Motoren

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

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

2.3 App Entwicklung

Eine App ist ein ausführbares Programm für mobile Geräte, wie Smartphones oder Tablets. Um eine App für ein mobiles Gerät zu entwickeln, müssen wie für andere Anwendungen im Voraus Anforderungen definiert werden, die diese erfüllen soll. Je nach festgelegten Anforderungen, die an das System gestellt werden, besteht eine bestimmte Anzahl von Möglichkeiten der Entwicklung. Allgemein kennt die App Entwicklung drei verschiedene Arten, die native, web und hybride Entwicklung, siehe (2.2.1), (2.2.2) und (2.2.3). Dabei werden verschiedene Frameworks verwendet, um mit unterschiedlichsten Programmiersprachen den Aufbau der Logik zu beschreiben. Eine App besteht immer aus zwei Teile, dem (UI), das meist mit einer (XML) ähnlichen Sprache beschrieben wird und dem Programmcode, der sich auf viele Klassen verteilt und die Funktionalitäten der App beschreiben.

Abbildung 2: App Entwicklung

2.3.1 Native Apps

In der Entwicklung von nativen Apps werden die direkten Ressourcen des Gerätes verwendet. Dazu gehört die Laufzeitumgebung des Betriebssystemes, Bibliotheken und Hardwareschnittstellen. Der Vorteil von einer nativen Entwicklung liegt hauptsächlich darin, dass diese für das Betriebssystem optimiert ist und die vorhandenen Schnittstellen genutzt werden können, um komplexe und rechenintensive Anwendungen zu ermöglichen.[1]
Vertreter diese Entwicklung finden sich für verschiedene Betriebssysteme. Der populärste unter ihnen ist bei weitem Android mit einer nativen Java Entwicklung über Android Studio von Google. Sie besitzt aktuellen den höchsten Marktanteil und eine entsprechende Popularität unter Entwickler und Nutzer.

2.3.2 Web Apps

Die Entwicklung von web Apps arbeitet mit systemübergreifenden Ressourcen und greift dabei auf gängige Webtechnologien, wie (HTML), (CSS) und JavaScript zurück. Die App wird hierbei nicht wie normale Anwendungen direkt auf dem System des Gerätes ausgeführt, sondern kommt in dessen Browser zur Ausführung. Der Vorteil hierbei ist vor allem, dass diese Art von App auf allen Betriebssystemen lauffähig ist und direkt über das Internet veröffentlicht und aktualisiert werden kann, jedoch wird eine stabile Internetverbindung vorausgesetzt.[1]

[1][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 perfomante web Apps entwickeln.

2.3.3 Hybride Apps

Die Entwicklung von hybride Apps vereinigt 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.[2]
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.

[2][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.[3] 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 zur erzeugen, der so entsprechend schnell ausgeführt werden kann.

Abbildung 3: Xamarin

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

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#, VisulF# und C++.

[3][vgl. 1, Introduction to Mobile Development - Xamarin]

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 Anforderunsdefinitionen

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

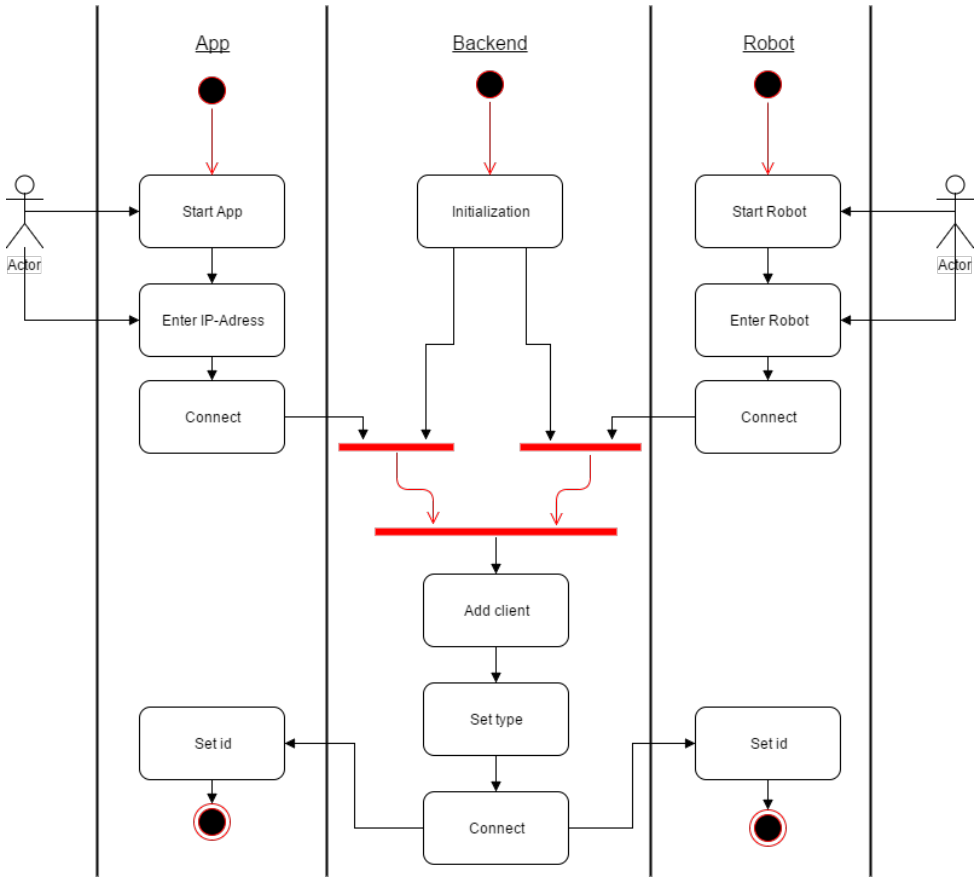
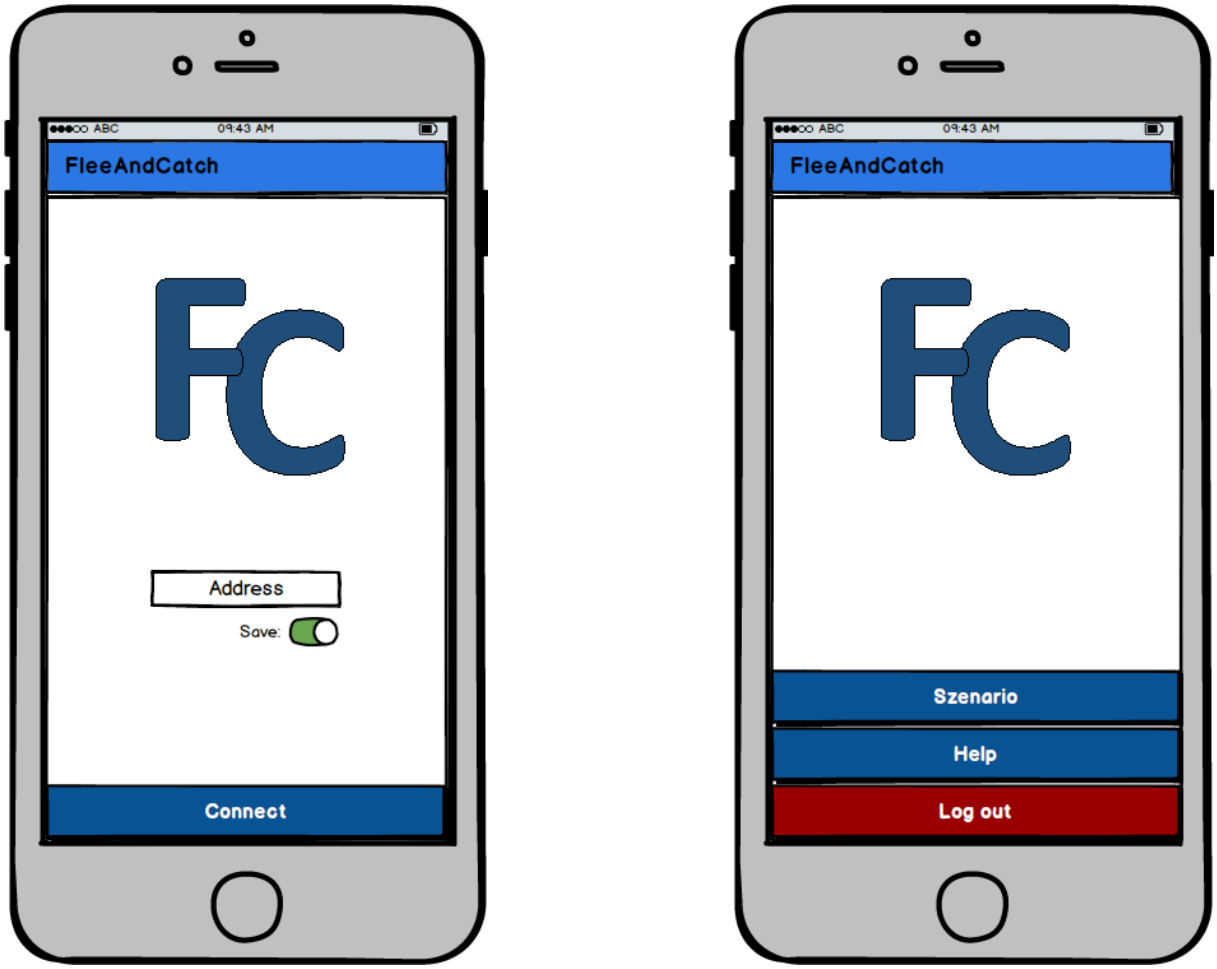


Abbildung 5: Connect

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



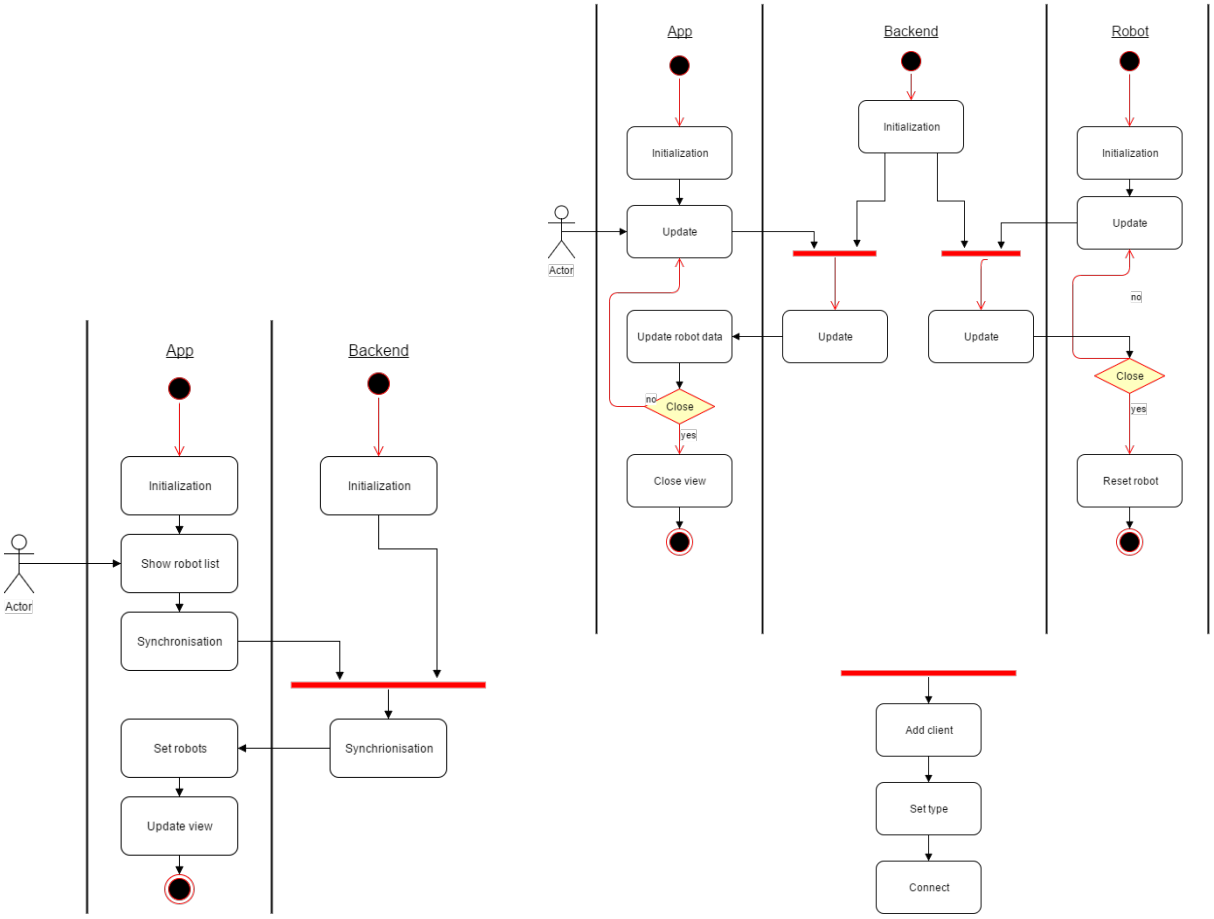
(a) Connect (b) Home

Abbildung 6: 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.



(a) Update all (b) Update

Abbildung 7: Synchronization

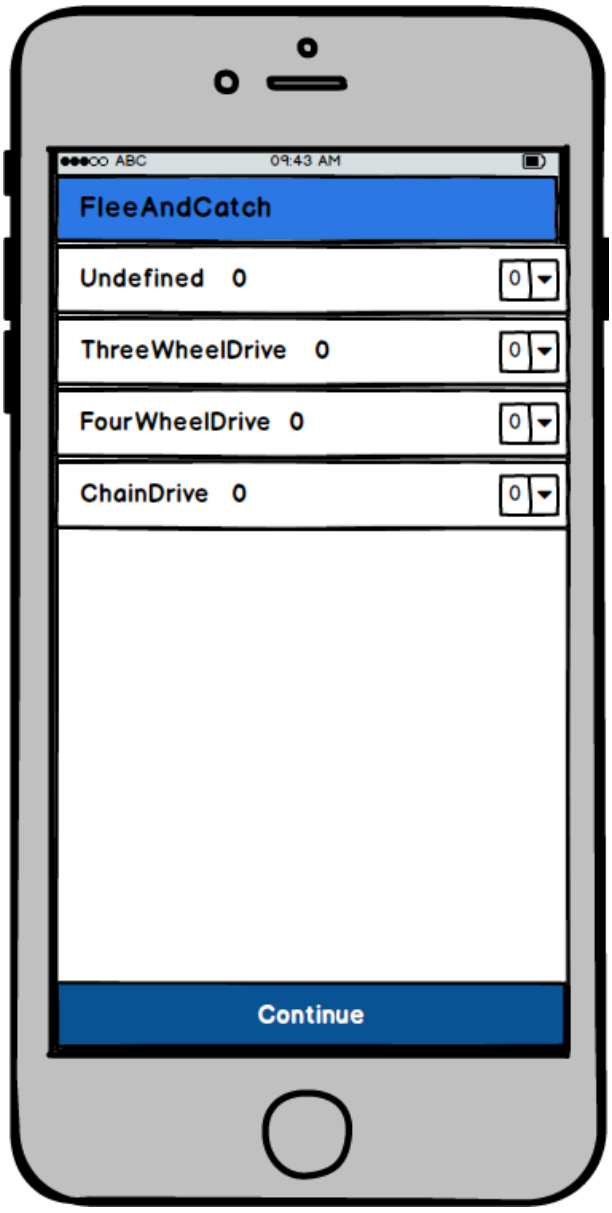


Abbildung 8: Robot list

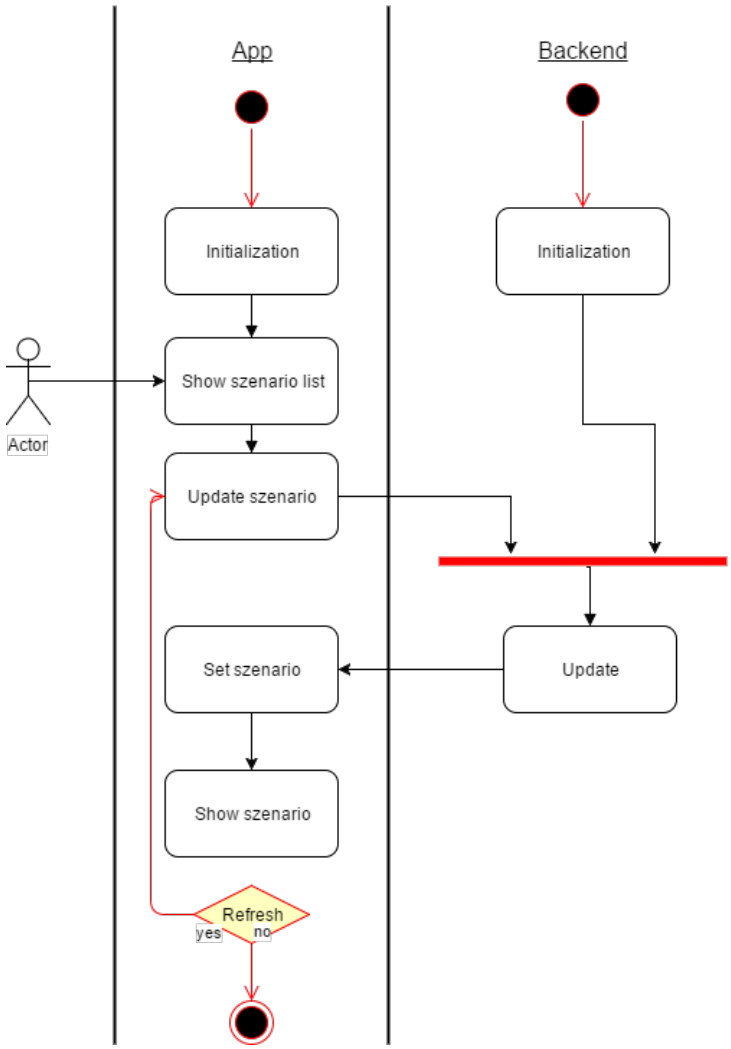


Abbildung 9: Spectator

5.2.3 Szenario

5.2.4 Exception

5.3 Kommunikation

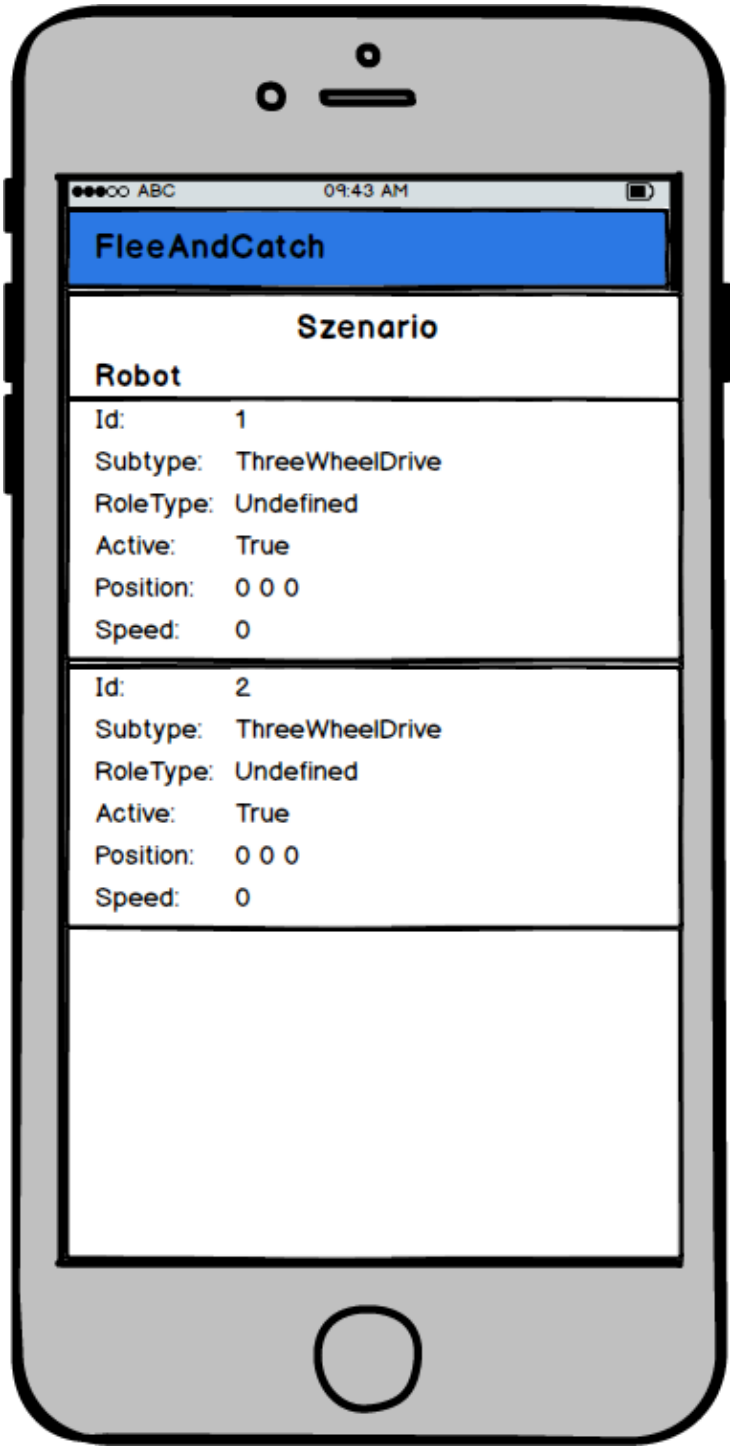


Abbildung 10: Spectator

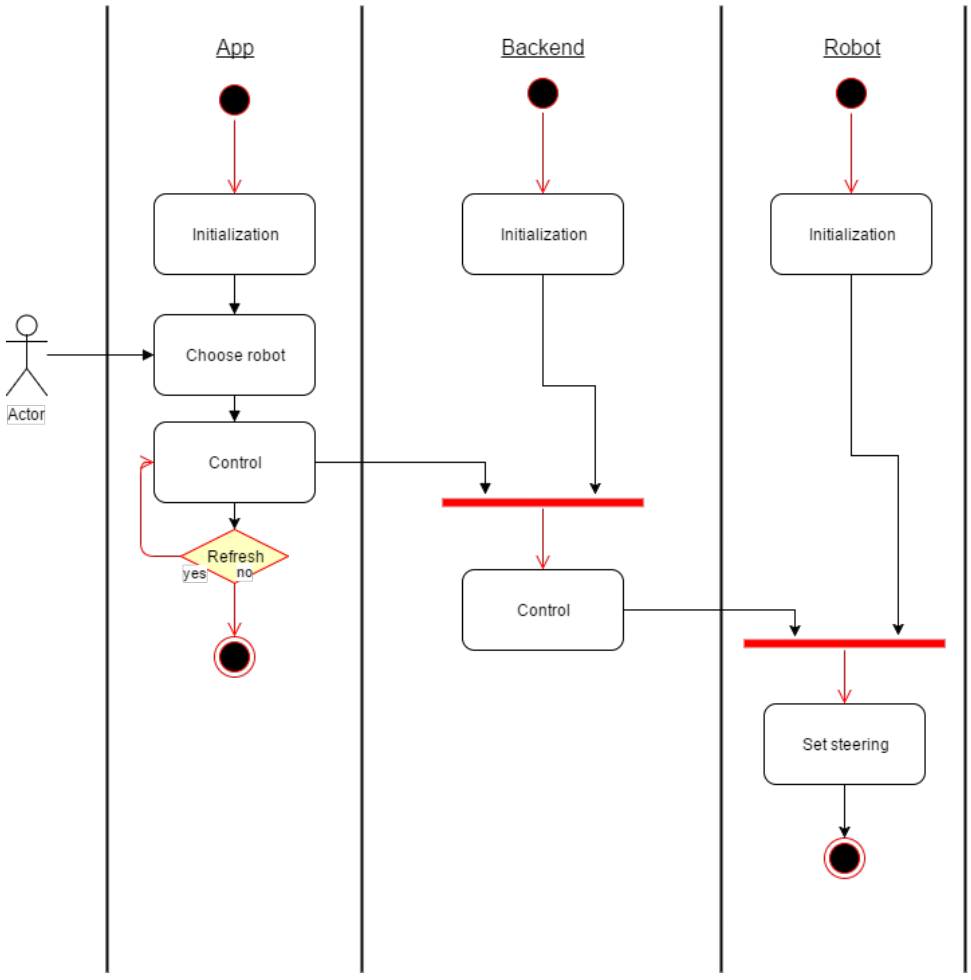


Abbildung 11: Control

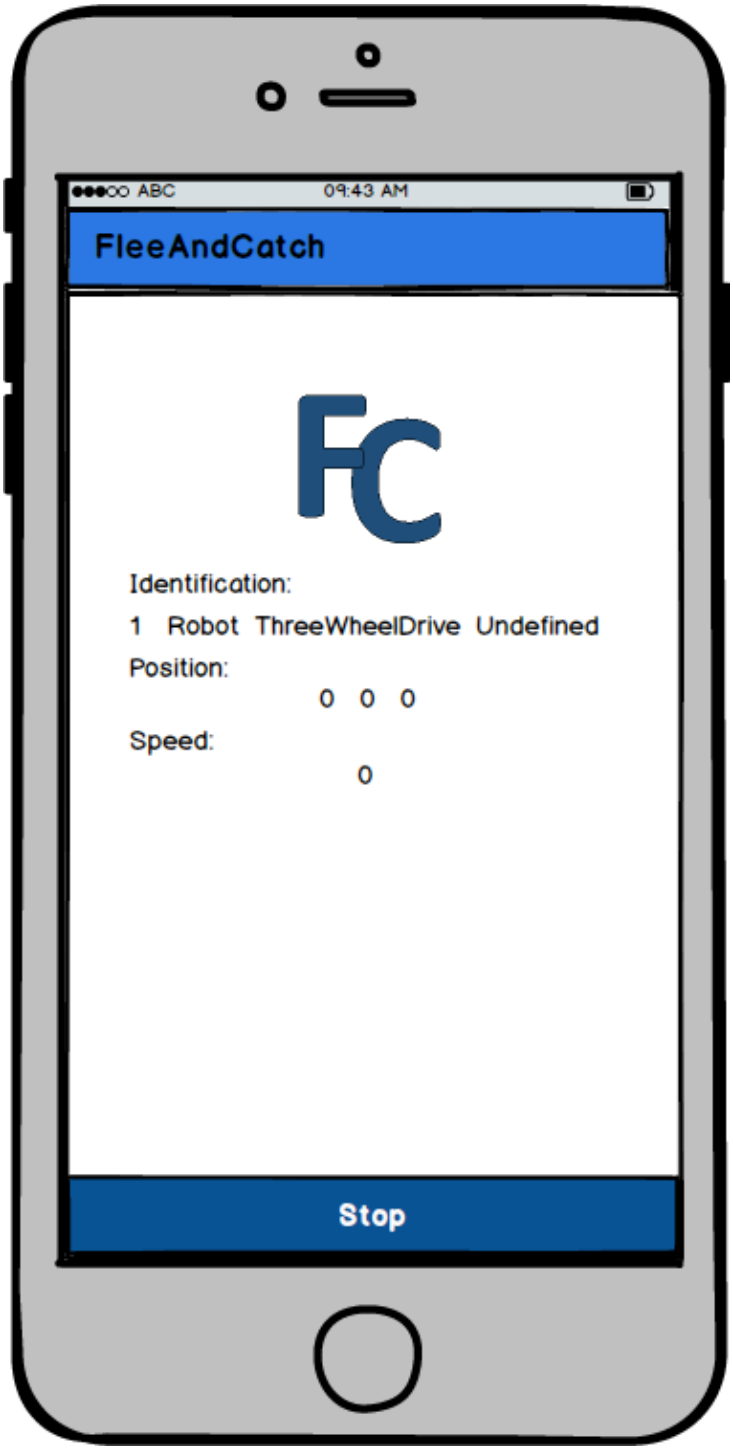


Abbildung 12: Control

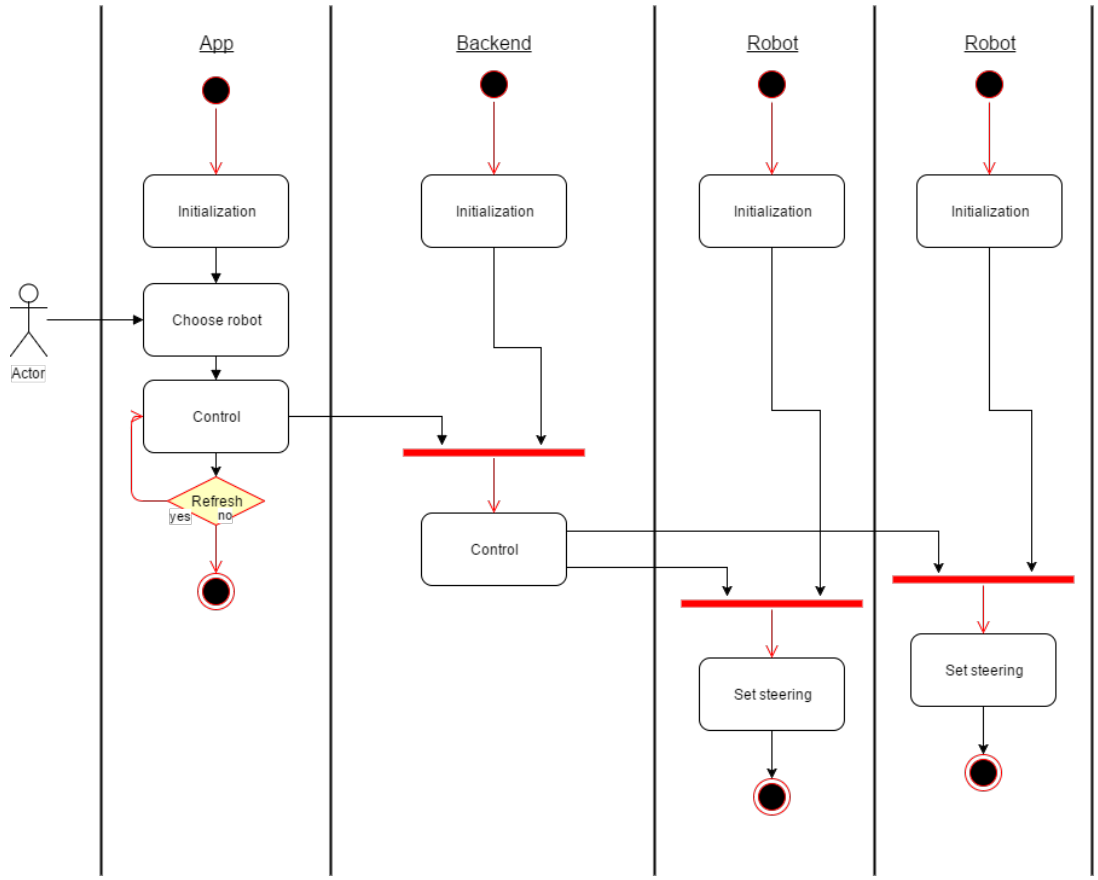


Abbildung 13: Synchron

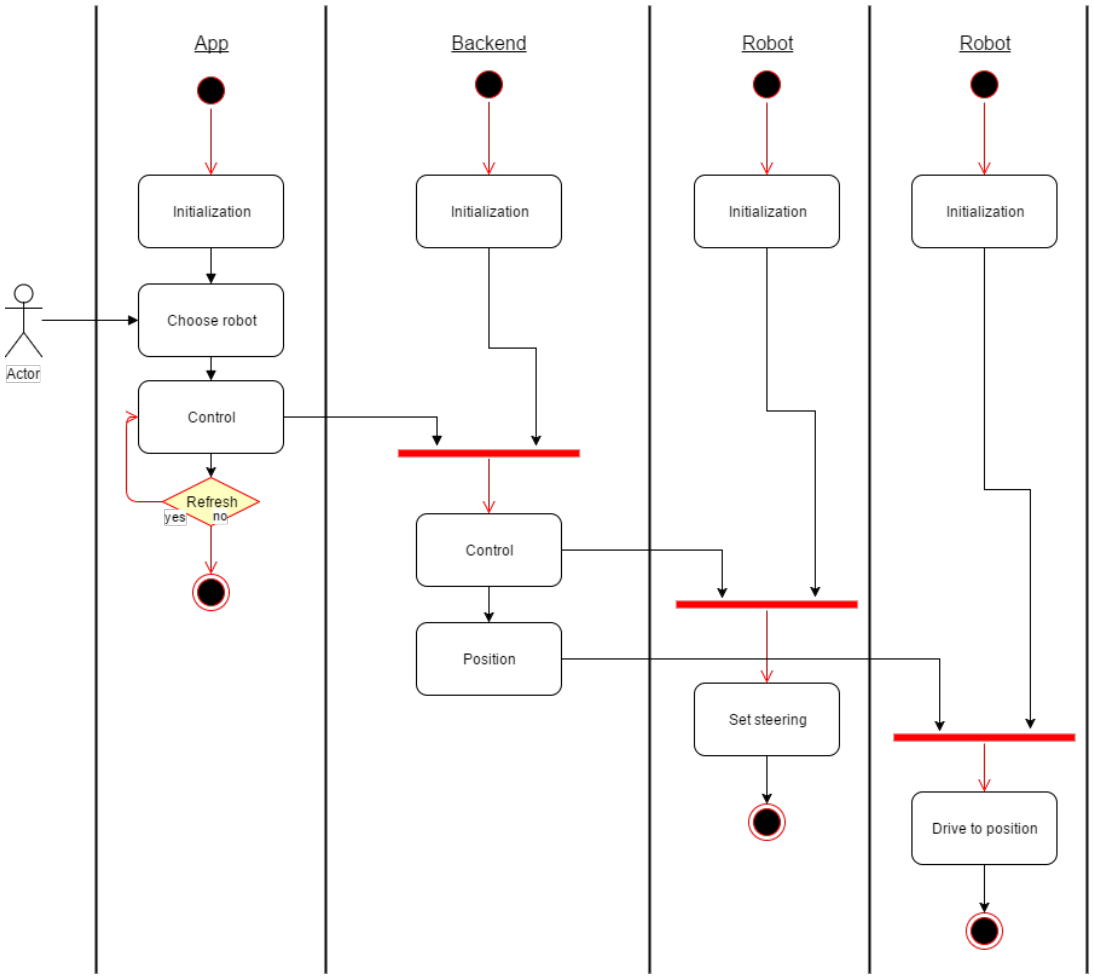


Abbildung 14: Follow

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/.

[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.

Anhang