ulm university universität
uulm

Fakultät für
Ingenieurwissenschaften
und Informatik
Institut für Organisation und
Management von Informations-
systemen

Dezember 2015

Steuerung eines Fussballroboters mit mobilen Endgeräten

Bachelorarbeit an der Universität Ulm

OMI-2015-12-31

Vorgelegt von:
Patrick Lutz

Gutachter:
Prof. Dr.-Ing. Stefan Wesner
Prof. Dr. Dr.-Ing. Wolfgang Minker

Betreuer:
Dipl.-Inf. Lutz Schubert

Fassung 21. Dezember 2015

Satz: PDF-LaTeX 2ε

Abstract

Diese Bachelorarbeit befasst sich mit dem Thema der Steuerung von eigens entwickelten Robotern unter Zuhilfenahme von mobilen Endgeräten als Steuercontroller. In diesem Kontext war die Aufgabe ein Fussballspiel zu ermöglichen bei dem zwei Roboter die Spielfiguren darstellen. Diese sollen von den Nutzern mit mobilen Endgeräten, zum Beispiel mit Smartphones gesteuert werden können. Um die Rechtzeitigkeit zu unterstützen wurden verschiedene Kommunikationsmethoden untersucht und sich für die am besten geeignetste entschieden. Außerdem beschäftigt sich diese Arbeit mit einigen Technologien, die die Benutzerfreundlichkeit und Bedienbarkeit unterstützen, wie zum Beispiel das selbstständige Auffinden und Anfahren der Ladestation.

Inhaltsverzeichnis

1 Einleitung

Bereits seit einigen Jahren finden sich in der Unterhaltungsindustrie zahlreiche Angebote von ferngesteuerten Robotern. Hierbei kann unterschieden werden zwischen den Arten, bei denen der Mensch die Steuerung übernimmt, oder denen bei denen die Roboter autonom fahren und handeln. 1997 wurde ein Wettbewerb gegründet, bei dem es Ziel ist, eine Art selbstspielende Roboter-Fussballmannschaft zu erstellen. Die Rede ist von RoboCup [4]. Hauptaugenmerk liegt hierbei natürlich in der Entwicklung der Roboter und der autonomen Steuerung.
Bei einer vom Menschen initiierten Steuerung, wie zum Beispiel bei ferngesteuerten Autos, stellt sich gleichzeitig die Frage der Übertragungstechnologie. Soll es möglich sein das Fahrzeug oder den Roboter auch ohne direkten Sichtkontakt zu steuern oder nicht? Was bedeutet Rechtzeitigkeit in diesem Kontext?
Ziel dieser Bachelorarbeit ist es, eine von Menschen initiierte Steuerung eines eigens entwickelten Roboters zu entwickeln, die sich mit diesen Fragen beschäftigt. Hierzu sind noch weitere Rahmenbedingungen zu definieren, um eine sinnvolle Entscheidungsfindung zu gewährleisten.

Auf einer Tischplatte in der Universität sollen zwei Roboter platziert werden. An den beiden Enden der Tischplatte werden Tore montiert, ähnlich wie man es beim Tischfussball kennt. Da seit einiger Zeit Smartphones und Laptops allgegenwärtig sind, sollen diese die Steuercontroller bilden. Mithilfe davon sollen nun die Roboter angesteuert werden können und ein Zwei-Spieler Fussballspiel ermöglicht werden. Die Roboter verfügen über einen Schussapparat, der es möglich macht den Ball zu beschleunigen. Sobald der Ball in ein Tor befördert wurde, wird ein Tor automatisch erkannt und auf den Steuercontrollern angezeigt. Diese Steuercontroller können entweder ein Android-Gerät oder jedes beliebige andere Gerät sein, das über eine Tastatur und einen Webbrowser verfügt. Über diese Steuercontroller ist es möglich die Roboter zu navigieren und einen Schuss zu tätigen. Hierfür werden verschiedene Steuerarten bereitgestellt. Die Roboter bewegen sich vollkommen kabellos, was die Verwendung eines Akkus notwendig macht. Sobald der Akkustand unter einen bestimmten Schwellenwert fällt, wird der Roboter von der Steuerungslogik automatisch an die Ladestation geführt, damit sich der Akku des Roboters lädt, ohne den Roboter manuell dort hin bewegen zu müssen. Über die Kamera, die an den Robotern befestigt ist, soll es möglich sein den Roboter ohne direkten Sichtkontakt zu steuern. Hierzu muss eine Kameraübertragung des Roboters auf die Steuercontroller erfolgen.

Die Fertigung und Implementierung der Roboter ist Teil einer vorangegangenen Bachelorarbeit, auf die hier nur Oberflächlich eingegangen wird [9].

Im ersten Teil der Arbeit werden die verwendeten Technologien näher gebracht. Daraufhin folgt die Diskussion über das Softwarekonzept der Steuerung. Weiter folgt die Implementierung und die tatsächliche Herangehensweise bei der Entwicklung dieser Steuerung und abschließend folgt ein kurzes Fazit und Ausblick auf mögliche Erweiterungen

1.1 Schnittstelle zum Roboter

Die Schnittstelle zum Roboter definiert auch die Abgrenzung dieser Bachelorarbeit zu der von Alexander Ulbrich. Durch den Pool der möglichen Befehle an den Roboter wird klar, was der Roboter bereits kann und was durch die Steuereinheit erreicht werden muss.

Da der Roboter eine Byte-Orientierte Kommunikationsweise erfordert, und keine Key-Value Objekte entgegennehmen und versenden kann, mussten die Pakete in Bezug auf die Befehlsreihenfolge für beide Übertragungsrichtungen definiert werden. Im Folgenden ist die Paketstruktur aufgelistet, die der Roboter vom Server empfängt.

Bytenummer	Befehl	Beschreibung
0	Start Byte	Jeder Befehl beginnt mit 0xFF
1	Kamera	0x01 aktiviert die Kamera, 0x00 deaktiviert sie
2	Schussapparat	0x01 löst einen Schuss aus, zu schnelle Wiederholungen werden vom Roboter ignoriert
3	Motorleistung links	Setzt die Leistung des linken Antriebs auf diesen Wert (zwischen 0 und 100 % in Hex, also 0x00 - 0x64)
4	Motorleistung rechts	Setzt die Leistung des rechten Antriebs auf diesen Wert (zwischen 0 und 100 % in Hex, also 0x00 - 0x64)
5	Checksumme	Dieses Feld ermöglicht die Erkennung von fehlerhaften Paketen. Sowohl dem Server, als auch dem Roboter ist die Berechnung hierfür bekannt.

Tabelle 1.1: Paketstruktur, die der Roboter annimmt

Hier wird ersichtlich, dass selbst für die Beschleunigung nach vorne, also gleichmäßiges Beschleunigen der Räder, keine Funktion oder derartiges bereit steht, lediglich die beiden Motoren können unabhängig voneinander angesteuert werden. Dadurch muss jegliche Logik, die es ermöglicht den Roboter in bestimmte Richtungen zu navigieren, von der Steuereinheit bereitgestellt werden.

In der anderen Richtung, also Pakete die vom Roboter aus gehen, gibt es lediglich einige Statusnachrichten und die Bilddaten. Auch diese sind in Tabelle 1.2 aufgeführt.

Bytenummer	Befehl	Beschreibung
0	Start Byte	Jeder Befehl beginnt mit 0xFF
1	Akkustand	Enthält den aktuellen Ladestand des Akkus (zwischen 0 und 100 % in Hex, also 0x00 - 0x64)
2	Goal-Flag	Enthält Werte zwischen 0x00 und 0x03. Das entspricht den Situationen: Tor in Sicht, Richtungsweisung in Sicht (links und rechts) und kein Tor in Sicht
3 - 9	Nicht-Implementiert	An dieser Stelle waren ursprünglich mehrere Stati eingeplant, jedoch aus Zeit- und Aufwandgründen nicht weiter beachtet
10 + 11	Bildlänge	In diesen beiden Bytes wird die Bildlänge mitgeteilt, die das Paket ab diesem Byte mit sich bringt
12 - ?	Bilddaten	Je nach Bedarf werden noch zusätzliche Bilddaten an das Paket angehängt

Tabelle 1.2: Paketstruktur, die der Roboter versendet

2 Grundlagen

In diesem Kapitel werden die verwendeten Technologien kurz erklärt und begründet, warum sie für diese Arbeit eine Bedeutung haben. Außerdem werden grundlegende Entscheidungen herbeigeführt, auf die sich die weitere Arbeit stützt.

2.1 Wahl der Übertragungsfrequenz

In Bezug auf die Rechtzeitigkeit der Steuerung ist es wichtig, dass für den Nutzer keine Verzögerungserscheinungen bei der Richtungsänderung des Roboters auftreten. Da der Mensch eine gewisse Latenz aufweist, um von der Analyse des Geschehens eine Handlung zu tätigen, hat die minimal notwendige Frequenz für Richtungsänderungen hier etwas Spielraum.
Um eine ausreichende Frequenz zu finden wurden an dieser Stelle einige Testläufe mit verschiedenen Frequenzen durchgeführt. Wie erwartet war die Steuerung mit 1Hz sehr träge und unsensibel. Ein rasches Ändern der Richtung oder Bremsen ist damit unmöglich. Um die Frequenz und damit den Traffic im Netzwerk möglichst gering zu halten wurde sie langsam erhöht. Ab einer Frequenz von 10Hz waren keine Verzögerungen vom Befehl zur Bewegung mehr festzustellen. Deshalb wurde die Befehlsfrequenz auf diese 10Hz gewählt. Die maximale Verzögerung von der Initiierung bis zur Umsetzung der Richtungsänderung beträgt hierbei 100ms. Bei den Testläufen befanden sich die Roboter, der Server und die Steuercontroller im selben Netzwerk. Sollte sich eine der Komponenten außerhalb des Netzes der anderen befinden, würden weitere Verzögerungen auftreten, die jedoch nicht durch die Frequenz beeinflussbar sind, da jedes einzelne Paket diese Verzögerung erfährt. Auch die Statusnachrichten kommen in diesem Intervall, es sei denn, es werden noch Bilddaten angehängt. In diesem Fall sendet der Roboter ohne Pausen zwischen den Paketen, damit die Bilddaten möglichst verzögerungsfrei auf dem Steuergerät angezeigt werden können.

2.2 Wahl der Endgeräte für die Steuercontroller

Für die Wahl der Endgeräte kamen einige Komponenten in Frage. Für die meisten Personen ist heutzutage ein Smartphone der ständige Wegbegleiter. Deshalb bietet es sich an, die Steuerung über ein Smartphone zu realisieren. Jedoch gibt es Unterschiede zwischen den verschiedenen Betriebssystemen der Smartphones. Die am weitesten verbreiteten sind Android und iOS. Um ein breiteres Nutzerspektrum abzudecken fiel die Entscheidung hier auf Android, da die Anzahl von Geräten auf denen Android operiert stetig steigt, wohingegen die von iOS sinken [2]. Um noch

mehr Nutzern die Bedienung zu ermöglichen soll zusätzlich eine Webanwendung entwickelt werden, um die Roboter mithilfe von Laptops oder PCs zu steuern.

2.3 UDP

Beim User Datagram Protocol (UDP) handelt es sich um ein verbindungsloses und in jeglicher Hinsicht ungeschütztes Protokoll, das auf der Transportebene (Schicht 4 im OSI-Schichtenmodell) arbeitet. Es bestehen keinerlei Mechanismen, die das korrekte Übertragen von Paketen gewährleisten.

Ein UDP Header besteht aus 8 Byte. Mit diesen 8 Byte werden lediglich Source-Port, Destination-Port, Länge und die Checksumme übertragen. Dieser vergleichsweise kleine Header (vgl. TCP mit etwa 20 Byte), führt zu einem geringen Overhead während der Übertragung, auch bei kleinen Paketen. Nachdem ein Paket gesendet wurde erfolgt keine Bestätigung des Pakets vom Empfänger. Durch diesen Uni-Direktionalen Sendevorgang entsteht wenig Traffic im Netzwerk. Falls gewisse Sicherheitsmechanismen gewünscht sind, müssen diese in höheren Schichten implementiert werden.
Aufgrund dieser Eigenschaften wird UDP in Bereichen eingesetzt, in denen eventuelle Paketverluste zu verkraften sind, bzw. von höheren Schichten aufgelöst werden und mehr Wert auf kurze Übertragungszeit gelegt wird.

Bei der Kommunikation mit dem Roboter kommt es hauptsächlich auf die Rechtzeitigkeit der Übertragung an. Einzelne Paketverluste sind vertretbar, da das fehlende Paket von dem nächsten ankommenden Paket ersetzt wird. Weil UDP keinen Sicherheitsmechanismen unterliegt, die eventuell Pakete zurückhalten ist dieses Protokoll hierfür gut geeignet.

2.4 TCP

Beim Transmission Control Protocol (TCP) handelt es sich um ein verbindungsorientiertes paketvermittelndes Protokoll. Mithilfe verschiedener Mechanismen wird sichergestellt, dass Pakete in der richtigen Reihenfolge ankommen, es zu keinen Staus kommt und dass Netzwerkknoten nicht überlaufen. Dadurch, dass das Protokoll verbindungsorientiert arbeitet, können beide Teilnehmer der Verbindung Daten ohne Anfragen senden.

Durch den größeren Header und den größeren Traffic, der das Protokoll verursacht, wird TCP für Anwendungen verwendet, bei denen ein Paketverlust ausgeschlossen werden soll, dafür aber eine etwas höhere Latenz in Kauf genommen werden kann. Die Verbindung wird über einen 3-Wege-Handshake hergestellt. Hierbei sendet ein Teilnehmer eine Anfrage (syn), diese wird bestätigt (syn ack), woraufhin die Bestätigung erneut bestätigt wird. In diesem dritten Schritt werden meist bereits die ersten Nutzdaten mitgesendet.

Mithilfe von geordneten Sequenznummern und Acknowledgments werden alle empfangene Pakete bestätigt. Durch das Fehlen einer Sequenznummer erkennt der Empfänger den Verlust

eines Pakets und teilt daraufhin dem Sender mit, dass es nicht angekommen ist. Für den Fall eines Verlusts des Acknowledgments, hat der Sender einen Timer, welcher nach einiger Zeit eine Retransmission des Pakets veranlasst, sofern kein Acknowledgment ankommen sollte.

Für die Kommunikation mit dem Roboter ist TCP aufgrund seiner höheren Übertragungszeit die ungeeignetere Variante als UDP. Auch der höhere Traffic, der vom Roboter verarbeitet werden muss könnte zu Problemen führen.

2.5 HTTP

Das Hypertext Transfer Protocol (HTTP) ist ein zustandsloses Datenübertragungsprotokoll, das auf der Anwendungsschicht arbeitet. Am meisten wird das Protokoll für den Aufbau von Internetseiten verwendet, also von einem Webbrowser, jedoch ist das Aufgabenfeld nicht darauf beschränkt.

HTTP arbeitet mittels zwei Befehlsarten, dem Request und dem Response. Möchte ein Browser eine Datei von einem Server laden, so sendet er einen Request mit dem Namen der Datei.
Einige mögliche Befehle sind:

GET fordert eine Ressource auf dem Server an

POST sendet Daten zur weiteren Verarbeitung zum Server

PUT lädt eine Ressource auf den Server

DELETE löscht eine Ressource auf dem Server (Wird kaum verwendet)

Nachdem der Request beim Server eingegangen ist und verarbeitet wurde, sendet er einen Response. In diesem Response sind Informationen über Server und Datei enthalten, sowie die Nutzdaten, also die angefragte Datei.

2.6 AJAX

Asynchronous Javascript And XML (AJAX) bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Browser und einem Server mittels HTTP. Dies ermöglicht die Veränderung der im Browser geladenen Seite, auch wenn der Server noch nicht geantwortet hat. Generell wäre AJAX ein Ansatz mit dem man die Kommunikation zwischen Webanwendung und Server realisieren könnte, jedoch hat es den Nachteil, dass pro Aufruf eine extra Verbindung aufgebaut wird. Bei häufigen Aufrufen könnte dies zu Problemen führen. Ein weiterer Nachteil ist, dass der eigentliche Sendevorgang vom Client über den Server geregelt wird, wodurch es nicht ohne weiteres möglich wäre, den Absender zu identifizieren. Der größte Nachteil wäre jedoch, dass für diese Art von Kommunikation eine eigene Schnittstelle für Webanwendung und

eine für die Android-Anwendung zur Verfügung stehen muss. Mit anderen Protokollen (siehe 2.7) ist es möglich sich auf eine Schnittstelle zu reduzieren.

2.7 WebSockets

Das WebSocket-Protokoll ist ein Netzwerkprotokoll, das auf TCP und HTTP basiert. In diesem Protokoll ist es möglich, eine bidirektionale Verbindung zwischen einer Webanwendung und einem WebSocket-Server herzustellen. Im Gegensatz zu reinem HTTP ist es hierbei möglich, dass der Server ohne einen vorhergehenden Request des Clients, Daten an den Client sendet. Lediglich den Verbindungsaufbau muss der Client initiieren. Dies wird realisiert, indem die TCP-Verbindung nach dem Verbindungsaufbau nicht sofort geschlossen wird. Eine WebSocket URL wird über die beiden Schemata **wss** und **ws** definiert, was für verschlüsselte und unverschlüsselte Verbindungen steht.

Der Verbindungsaufbau funktioniert über einen Handshake, der wie in HTTP üblich über einen Request und anschließenden Response erreicht wird. Aufgrund der Tatsache, dass die HTTP-Header nur beim Verbindungsaufbau gesendet werden und dadurch der Traffic durch den HTTP-Header gering ist, wird das Protokoll hauptsächlich von Anwendungen verwendet, die regelmäßige Kommunikation zwischen Client und Server verlangen, wie zum Beispiel Online-Spiele.

Für die Kommunikation mit den Steuergeräten stellen die WebSockets eine praktische Lösung dar. Da als Grundlage TCP für die WebSockets verwendet werden ist es möglich eine dauerhafte Verbindung zu halten. Vor allem die Webanwendung schränkt die Übertragungsprotokolle enorm ein, da es Client-Seitig nicht möglich ist eine direkte TCP Verbindung aufzubauen, HTTP jedoch schon. Mit WebSockets ist es möglich eine einheitliche Schnittstelle für die Steuergeräte zu erstellen, unabhängig von welcher Plattform sie zugreifen.

2.8 Jetty Webserver

Jetty [1] ist eine Java-Implementierung eines Webservers. Durch seine geringe Größe ist es leicht, ihn in andere Software zu integrieren. Außerdem unterstützt Jetty die Möglichkeit WebSockets aufzubauen. Damit ist Jetty eine gute Wahl, um die Webanwendung zusammen mit der anderen Software bereitzustellen, ohne einen extra Webserver zu konfigurieren. Im Vergleich zu anderen Java-Implementierungen von Webservern hat Jetty den Vorteil, dass es mithilfe einer einfach Library in das Projekt eingebunden werden kann. Es ist nicht notwendig eine Verzeichnisstruktur zu beachten oder weiteren Installationsaufwand zu betreiben.

2.9 Base64

Base64 ist ein Kodierungsverfahren, bei dem 8-Bit Binärdaten in eine rein aus lesbaren Zeichen bestehende Zeichenkette umgewandelt wird. Bei diesem Verfahren steigt der benötigte Platzbedarf um 33-36 %. Jedoch ist es für eine sehr Hardware-abstrahierte Programmiersprache wie Javascript, die bei der Webanwendung zur Anwendung kommt ungeeignet auf Byte-Ebene zu kommunizieren. Mit der base64 Kodierung wird eine reine Zeichen basierte Übertragung mit den Steuergeräten möglich.

3 Einführung der Komponenten

In diesem Kapitel findet eine kurze Einführung in die verwendeten Komponenten statt. Im restlichen Teil der Arbeit wird immer wieder auf diese Rollen zurück gegriffen.

3.1 Roboter

Wie in Kapitel 2 eingeführt wurde der Roboter im Rahmen einer weiteren Bachelorarbeit erstellt. Hier soll nur kurz auf die für diese Arbeit wichtigen Aspekte des Roboters eingegangen werden. Der Roboter verfügt über zwei Räder, die über jeweils einen Motor beschleunigt und abgebremst werden können. Außerdem befindet sich vorne ein Schussmechanismus, der mit einem Befehl (vgl. 1.1) ausgelöst werden kann. Die beiden Motoren der Räder beschleunigen den Roboter auf bis zu $2\frac{m}{s}$. Dadurch wird ersichtlich, dass bei hohen Geschwindigkeiten eine gute Reaktionszeit notwendig ist, damit sich der Roboter noch steuern lässt. Auf der hinteren Seite des Roboters befinden sich Ladekontakte und knapp darüber ein Infrarot-Empfänger. Ein Bild des Roboters befindet sich im Anhang 8.4. Für diese Arbeit werden zwei Roboter benötigt.

3.2 Server

An dieser Stelle wird die Entscheidung aus Kapitel 5 vorgegriffen um später besser mit den Begrifflichkeiten umgehen zu können.
Der Server ist ein Java-Programm, das die zentrale Kommunikationsstelle darstellt. Er trennt die Controller von der Logik und von der direkten Kommunikation mit den Robotern. Dieses Programm wird auf einem RaspberryPi ausgeführt, der sich mit einer festen IP-Adresse im Universitätsnetz befindet. Über die GPIO-Pins des RaspberryPi's sind die Unterkomponenten des Tors angeschlossen.

3.3 Tor

Das Tor soll mehrere Funktionalitäten in sich vereinen. Da jeder Roboter einem Tor zugeordnet wird, soll der Roboter die Tore unterscheiden können. Nachdem er sie unterschieden hat soll der Server in der Lage sein den Roboter zu diesem Tor zu navigieren. Dadurch ist es nicht notwendig, dass der Nutzer selbst den Roboter zum Tor navigiert, wenn der Akku entladen ist. Der Roboter fährt ganz ohne Nutzerinteraktion zum Tor, auch wenn momentan kein Spieler verbunden ist.

Das Tor wurde unter Verwendung von 3D-Druck erstellt und konnte dadurch den Anforderungen entsprechend entworfen werden. Das verwendete Programm für die 3D-Zeichnung war Solid Edge [5].

3.3.1 Ladestation

Mit der Ladestation sind die Ladekontakte gemeint, die am Tor befestigt sind. Sie befinden sich auf Höhe der Ladekontakte des Roboters, um ein Laden des Roboters zu ermöglichen, nur indem er gegen die Kontakte des Tors fährt.

3.3.2 IR-LED

Die Infrarot-LED (IR-LED) ist die Komponente, die notwendig ist, um dem Roboter zu signalisieren wo sich das Tor und damit die Ladestation befindet. Tatsächlich handelt es sich hierbei um drei Infrarot-LEDs die zu einem Bauteil zusammengefasst wurden. Sie befinden sich am oberen Rand des Tore, auf Höhe des IR-Empfängers des Roboters.

3.3.3 Torerkennung

Die Torerkennung ist der mechanische Schalter, der ein Signal an den Server weiterleitet sobald ein Tor erzielt wurde.

3.4 Steuercontroller

Ein Steuercontroller bezeichnet im Folgenden das Gerät, das der Nutzer verwendet um den Roboter zu steuern. Das kann entweder ein Android-Gerät sein, auf dem die Android-Anwendung ausgeführt wird oder die Webanwendung mithilfe eines Laptops oder PCs. Die Android-Anwendung funktioniert auf allen Geräten, auf denen mindestens Android API 14 installiert ist.

4 Kommunikation

Im Folgenden werden die Übertragungstechnologien untersucht und ausgewählt, welche für die Anforderungen am besten geeignet ist. Dem gesamten Kapitel liegen die Erkenntnisse aus Kapitel 2 zugrunde.

4.1 Übertragungstechnologie

Da die Roboter, sowie auch die Controller über Funk mit dem Server kommunizieren sollen, gab es an dieser Stelle einige Entscheidungen.
Für die Kommunikationswege Roboter-Server und Controller-Server gab es verschiedene Anforderungen. Da der Server in Nähe der Roboter aufgestellt werden soll spielt hierbei die Übertragungsdistanz keine sehr große Rolle, weshalb Bluetooth denkbar wäre, auch WLAN wäre eine gute Lösung, da in den meisten Gebäuden ein drahtloses Netzwerk vorhanden ist. Selbst optische Übertragungsverfahren wären denkbar (und werden sogar verwendet, siehe Kapitel 6.1.1).

Die Hauptanforderungen an die Roboter-Server Übertragung sind Übertragungsgeschwindigkeit und Ausfallsicherheit. Eine energiesparende Lösung ist aufgrund der begrenzten Akkukapazität der Roboter wünschenswert.
Da die Roboter ständig in Bewegung sind und Richtungsänderungen schnell und zuverlässig verarbeitet werden sollen, wurde die optische Übertragung ausgeschlossen. Bei einem solchen Verfahren müsste ein ständiger Sichtkontakt zwischen Sender und Empfänger bestehen, was eine robuste Übertragung beim Fahren um Hindernisse nahezu ausschließt.

Bluetooth würde die Anforderung der Ausfallsicherheit erfüllen, da die Roboter auf einem begrenzten Gebiet zum Einsatz kommen und die Distanz deshalb ausreichend gering wäre.

WLAN bringt eine hohe Ausfallsicherheit, da die Verbindung in Gebäuden als nahezu Konstant anzunehmen ist. Da der Server im gleichen Netzwerk hängt wie der Roboter, ist es bei dieser Übertragung irrelevant wie weit die beiden Geräte tatsächlich voneinander entfernt sind, solange sie sich eben im selben Netzwerk befinden, was gegenüber Bluetooth mehr Spielraum für die Größe des Spielfeldes übrig lässt. Was die Übertragungsgeschwindigkeit angeht, ist WLAN um einiges schneller (stark von Version abhängig) [7]. Das ist vor allem für die Übertragung der Bilddaten wichtig, da dort hohe Datenmengen in kurzer Zeit übertragen werden müssen.

Aufgrund der besseren Flexibilität und der höheren Übertragungsrate fiel an dieser Stelle die Entscheidung auf **WLAN**. Der Vorteil von Bluetooth bestünde hier lediglich in der besseren

Energieeffizienz, jedoch ist die Akkubelastung, die durch die Kommunikation eintritt im Vergleich zu der der Motoren wesentlich geringer. Dadurch fällt die vergleichsweise hohe Belastung von ca. 300mA bei WLAN im Gegensatz zu ca 50mA bei Bluetooth (beim Sendevorgang) [10] nicht so sehr ins Gewicht, wie der Stromverbrauch, der durch die Motoren und den Schussapparat verursacht wird.

Für die Kommunikation zwischen Controller und Server gelten prinzipiell die gleichen Anforderungen, jedoch andere Rahmenbedingungen. Zusätzlich soll es möglich sein den Server mit seinem Controller auch außerhalb der Universität zu erreichen. Da die mobilen Endgeräte in den meisten Fällen ebenfalls über einen Akku mit Strom versorgt werden, spielt hier auch der Aspekt der Energieeffizienz eine Rolle. Weil die Controller den Server auch von weiter entfernten Orten erreichen sollen, die auch außerhalb der Universität liegen können, ist hier lediglich eine Kombination verschiedener Übertragungstechnologien möglich. Deshalb fiel die Entscheidung hierbei ebenfalls auf die Netzwerkvariante, da hiermit das Übertragungsmedium keine entscheidende Rolle spielt. So ist es zum Beispiel möglich sich von zu Hause mittels VPN Zugang zum Universitäts-Netzwerk zu verschaffen. Diese Entscheidung erlaubt es auf Netzwerkebene zu agieren, ohne sich über die Data-Link Layer Gedanken machen zu müssen.

4.2 Übertragungsprotokoll

Nach der Entscheidung über die Übertragungstechnologie blieben für die Wahl des Übertragungsprotokolls grundlegend nur wenige Protokolle zur Auswahl. Diese sind UDP und TCP. Alle anderen denkbaren Protokolle basieren letztendlich auf einem der beiden und stellen nur Erweiterungen dar.

Bei der Roboter-Server Kommunikation gibt es hierbei einige Dinge zu berücksichtigen. In der Richtung Server-zu-Roboter werden Richtungsänderungen sowie der Befehl zum Auslösen des Schussapparats übertragen. Damit Richtungsänderungen nicht auf dem Übertragungsweg verloren gehen wäre ein gesichertes Transportprotokoll die bessere Variante. Mittels TCP wäre hier garantiert, dass keine Pakete verloren gehen und Richtungsänderungen deswegen nicht ausgeführt werden können. Trotzdem wurde für das Übertragungsprotokoll **UDP** gewählt. Genauer betrachtet bedeutet die Ausfallsicherheit des TCP Protokolls keinen großen Vorteil für die zuverlässige Steuerung. Laut der Schnittstellendefinition gilt eine Befehlsfrequenz von 10Hz. Das bedeutet, wenn von zehn Paketen in einer Sekunde hin und wieder eines verloren geht, entsteht kurzzeitig eine Verzögerung von 100ms (2.1). Im Gegensatz zu TCP profitiert UDP allerdings von seiner Eigenschaft wenig Traffic zu verursachen [8]. Da bei TCP jedes Paket bestätigt wird muss der Roboter jedes Paket mehrmals verarbeiten. Die Hardware des Roboters ist jedoch eher schwach bestückt, weshalb ein Paketverlust wegen zu wenig Speicher und die daraus resultierende Sendewiederholung durchaus häufig vorkommen könnte. Deshalb ist es auch zu verkraften, dass gelegentliche Verzögerungen der Richtungsänderungen auftreten. Auch nach Betrachtung der anderen Übertragungsrichtung ist dies die bessere Entscheidung. In Roboter-zu-Server Richtung werden lediglich Statusänderungen, wie zum Beispiel der Akkustand, und Bilddaten der Kamera übertragen. Weder die Statusnachrichten noch die Bilddaten sind hier den Mehrtraffic wert, den TCP verursachen würde. Zumal allein die Größe der Bilddaten die Hardware des Roboters in kürze auslasten würde.

Die Anforderungen für die Controller-Server Kommunikation sind im Grunde die gleichen wie die der Roboter-Server Kommunikation. Diese werden jedoch noch darum erweitert, dass das Protokoll mit mehreren verschiedenen Clients kommunizieren soll. Da unter anderem eine Webanwendung (siehe Abschnitt 2.2) einen Kommunikationspunkt darstellt, muss dieser Punkt bei der Entscheidung berücksichtigt werden. Im Gegensatz zu der Roboter-Server Kommunikation haben wir an dieser Stelle jedoch keinen leistungsschwachen Teilnehmer, weshalb TCP die bessere Variante ist. Weil mit der Webanwendung bereits ein hohes Maß an Abstraktion der niederen Datenstrukturen besteht, ist es nicht ratsam dennoch ein Protokoll auf Byte-Ebene zu wählen, nicht zuletzt, da Javascript ohnehin keine direkte Client-Seitige Unterstützung für TCP oder UDP bietet. Die Lösung an dieser Stelle ist die Verwendung von **Websockets**. Diese basieren auf HTTP, was wiederum auf TCP basiert. Mit ihnen ist es möglich, innerhalb des Client-seitigen Codes mit Javascript eine bidirektionale Verbindung zu einem Socket aufzubauen.

5 Software Architektur

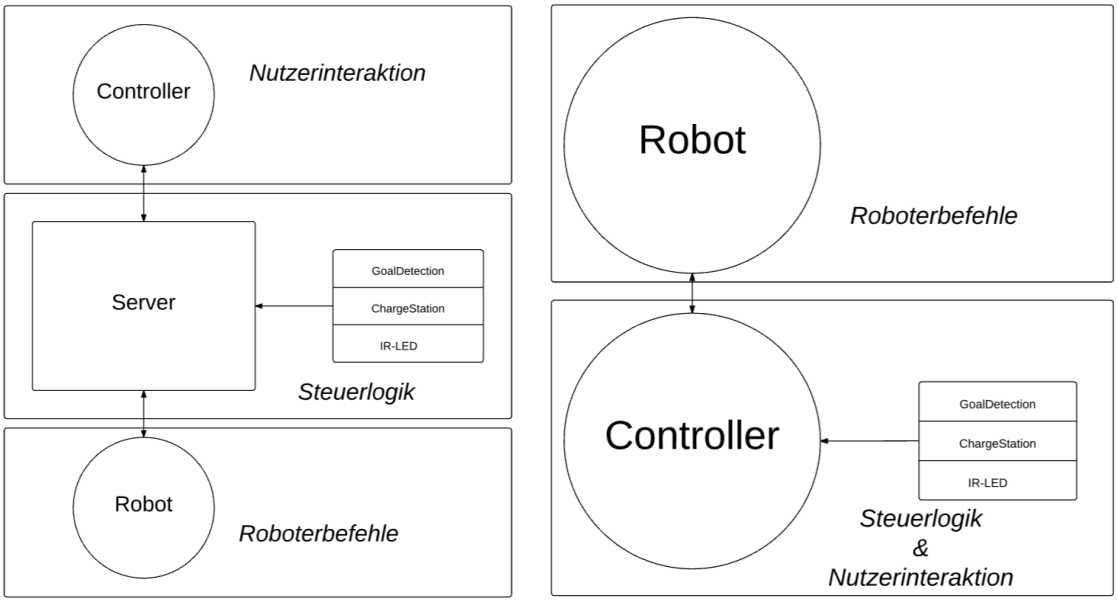
Die Software-Architektur des Projektes definiert die Möglichkeiten bei der Implementierung, die man zum Erfüllen der Anforderungen zur Verfügung hat. Deshalb war es wichtig, sich ausreichend Gedanken zu machen, um später im Projekt nicht feststellen zu müssen, dass die gewählte Architektur für die Anforderungen ungeeignet ist. Die Hauptanforderung an die Architektur ist selbstverständlich die Bereitstellung eines Kommunikationskanals zwischen Controllern und Robotern. Es muss möglich sein Richtungs- und Geschwindigkeitsänderungen seitens der Anwender rechtzeitig an die Roboter mitzuteilen. Interaktivität spielt hierbei eine große Rolle. Die getätigten Steuerungen sollen möglichst zeitnah umgesetzt werden, damit der Benutzer ein gutes Feedback auf seine Steuerung erhält, auch wenn der Roboter eine hohe Geschwindigkeit hat. Der Status des Roboters soll von der Steuerlogik jederzeit ausgewertet und entsprechend darauf reagiert werden können. Hierzu gehören das automatische Anfahren der Ladestation bei geringem Akkustand und die Übertragung der Bilddaten. Außerdem sollen Zustandsänderungen des Spiels, die nicht vom Roboter oder Controller direkt ausgehen erkannt und korrekt verarbeitet werden. Der Nutzer soll lediglich die Informationen Akkustand, Spielstand und die Kameraübertragung zugeteilt bekommen. Alles weitere ist Aufgabe der Steuerlogik und soll vom Nutzer abstrahiert werden. Dadurch entsteht eine klare Rollenverteilung: Der Roboter nimmt lediglich Befehle für die Motoren und den Schussapparat an und der Steuercontroller bekommt Informationen, um mit dem Nutzer interagieren zu können.

Die betrachteten Architekturen waren folgende:

Eine **Client-Server-Architektur** bei der ein Server zentral die Kommunikation verwaltet. Hierbei findet keine Kommunikation zwischen den Steuergeräten und den Robotern direkt statt. Die Torerkennung kommuniziert direkt mit dem Server, ohne dass Roboter und Controller zwangsweise etwas davon mitbekommen. Sowohl die Roboter, als auch die Controller nehmen die Rolle eines Clients ein. Dadurch ist es Möglich, dass eine Steuerung des Servers stattfinden kann, ohne dass ein Controller verbunden ist. Der Server verwaltet zentral alle Informationen von Steuercontroller und Roboter und leitet nur diejenigen weiter, die für den anderen relevant sind. Dadurch wird ein Steuercontroller wirklich nur zu einem Gerät, mit dem der Roboter gesteuert werden kann und das sonst keine weitere Logik enthält. Zusätzlich ist es in dieser Architektur möglich beliebig viele Clients anzubinden. Durch die Trennung von Client und Logik können die Controller beliebig erweitert und ausgetauscht werden, ohne die eigentliche Spiellogik zu ändern.

Eine Architektur, die Roboter und Controller **paarweise** verbindet. Bei dieser Architektur wird jedem Roboter ein Controller zugewiesen, der genau diesen steuert. Die peripheren Geräte senden direkt an die Controller. Auch die gesamte Spiellogik ist in den Controller gekapselt. Ein Vorteil dieser Architektur ist die geringe Latenz bei der Befehlsübermittlung. Im Vergleich zur Client-Server-Architektur gibt es hier keinen dritten Knoten, über den die Übertragung

abgehalten wird, wodurch sich die benötigte Zeit halbiert. Dies stellt für die Rechtzeitigkeit der Steuerung einen erheblichen Vorteil dar. In Bezug auf die Rollenverteilung hat diese Architektur jedoch gravierende Nachteile. Wenn die gesamte Steuerlogik in die Steuercontroller integriert ist, kann der Roboter auch nur dann agieren, wenn ein Steuercontroller verbunden ist. Das Anfahren der Ladestation wäre dadurch erst möglich, sobald sich ein Nutzer verbindet.



(a) Client-Server Architektur **(b)** Paarweise Architektur

Abbildung 5.1: Mögliche Architekturen

Nach Abwägung der Vor- und Nachteile beider Architekturen schien es vernünftiger sich für die Client-Server-Architektur zu entscheiden. Die Hauptgründe hierfür sind die hervorragende Erweiterbarkeit in Bezug auf die Controller und die klare Rollenverteilung. Da keine Logik in den Controllern ist, muss sie auch nicht für jeden Controller separat entwickelt werden, beziehungsweise nicht einmal vorhanden sein. Das spielt unter Betrachtung von Abschnitt 2.2 eine wichtige Rolle um die Webanwendung gut integrieren zu können. Für das Auffinden der Ladestation wäre es nicht sinnvoll, solange mit dem Laden zu warten, bis ein Controller verbunden ist, nur um dann festzustellen dass der Akku leer ist und der Roboter erst geladen werden muss, bevor gespielt werden kann. Auch in Anbetracht der Informationskapselung, die bei der paarweisen Architektur entstehen würde, ist es sinnvoller die Client-Server-Architektur zu bevorzugen. In Anbetracht der Rechtzeitigen Steuerung bietet die gewählte Architektur einen Nachteil gegenüber der paarweisen, da eine dritte Komponente in der Übertragungskette vorhanden ist und deshalb doppelt so viele Pakete versendet werden müssen. Jedoch wird sich später bei der Implementierung herausstellen, dass die Übertragungsgewschwindigkeit auch bei der Client-Server Architektur hoch genug ist, um eine für den Benutzer verzögerungsfreie Steuerung zu gewährleisten.

6 Implementierung

In diesem Kapitel werden die Vorgehensweisen und Verwendungen der Technologien beschrieben. Hierbei wird zuerst auf die Ladestation eingegangen, da dort die Funktionsweise der peripheren Geräte erklärt wird, die für die Implementierung der Server-Logik wichtig sind. Anschließend folgt der Server und zum Schluss die Steuercontroller.
Der Quellcode des Projekts ist unter der Referenz [6] zu finden.

6.1 Das Tor

Das Tor an sich beinhaltet keine Funktionalität, außer ein Ziel für die Spieler darzustellen. Die eigentlichen Funktionen befinden sich in den folgenden Komponenten, die in das Tor integriert wurden.

6.1.1 Infrarot-LED

Die Infrarot-LED dient dazu, dem Roboter die Richtung zu vermitteln, in der seine Ladestation, also sein Tor zu finden ist. Bei dieser Art der Kommunikation gibt es nur einen Sender und einen Empfänger. Die Infrarot-LED nimmt hierbei die Rolle des Senders ein und der Roboter die des Empfängers. Um die Funktionalität der Kommunikation besser zu vermitteln, wird im Folgenden zunächst auf die technischen Grundlagen bei der IR-Kommunikation eingegangen. Der Infrarot Empfänger besteht aus einer Photodiode, einem bei einer gewissen Frequenz geregeltem Verstärker und einem Demodulator. Bei unserem Empfänger beträgt die Verstärkerfrequenz etwa 38 kHz. Nun ist es möglich, Signale mit der IR-LED zu erzeugen, die dann vom Empfänger verarbeitet werden. Hierbei beträgt die Trägerfrequenz des modulierten Signals genau die Frequenz, auf der der Verstärker des Empfängers arbeitet, also 38 kHz. Das Signal besteht aus zwei Werten, entweder aus der 0, also die IR-LED ist aus, oder aus einer logischen 1, indem die IR-LED mit diesen 38 kHz blinkt. Gemessen werden kann nun die Frequenz, die aus der Dauer der logischen 1 und der logischen 0 resultiert.

Realisiert wurde diese Frequenz mit dem Timerbaustein NE555, an den die drei LEDs parallel zueinander geschaltet sind. Diese lassen sich mithilfe von Transistoren vom RaspberryPi aus unterschiedlich lang an und ausschalten. Der Schaltplan inklusive Bauteilwerte befindet sich im Anhang 8.1b

Der Aufbau ist wie in Abbildung 6.2 zu sehen folgendermaßen: Am oberen Endes Tores sind drei Infrarot-LEDs befestigt. Die äußeren beiden haben einen Abstrahlwinkel von 40°und die

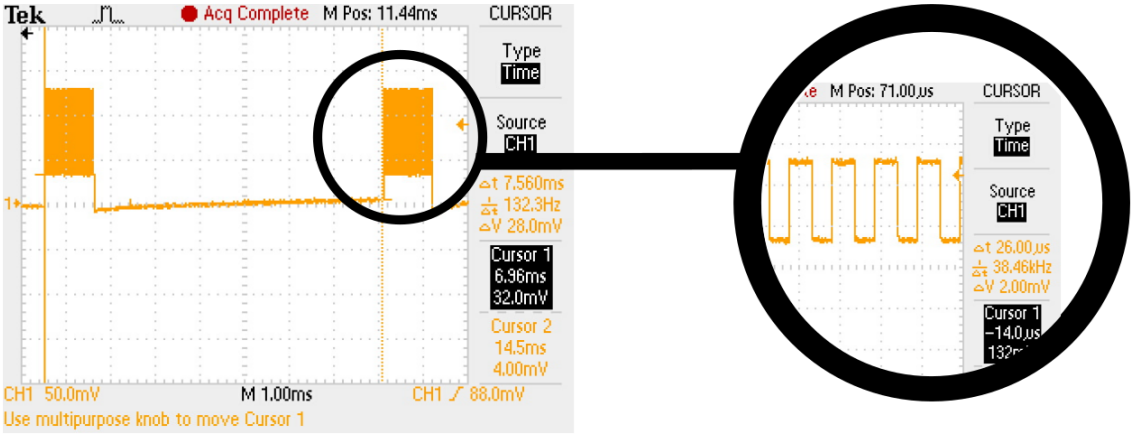


Abbildung 6.1: Beispiel Burst mit einer postivien Pulslänge von ca. 1ms und ca. 6,4ms Pause

mittlere einen von 20°. Diese sind so angeordnet, dass sich die Signale nicht überschneiden. Auf dem Roboter ist der Empfänger auf derselben Seite wie die Ladekontakte platziert. Hierbei muss beachtet werden, dass der Empfänger seitlich abgeschirmt ist, das bedeutet er bekommt nur Signale die tatsächlich frontal auf ihn eintreffen.

Um den Ladeprozess möglichst fehlerfrei zu starten ist es notwendig, dass der Roboter möglichst frontal auf die Ladestation zu fährt, deshalb hat die mittlere LED nur einen Abstrahlwinkel von 20°(maximal 10°Abweichung). Diese ist somit die LED, die signalisiert, dass der Roboter nun korrekt platziert ist und sich nur noch gerade aus auf die LED zu bewegen muss. Die anderen beiden LEDs dienen dem Zweck, den Roboter in genau diese Position zu bringen. Da die LEDs unterschiedlich moduliert sind, kann der Roboter erkennen in welchem Signal er sich nun befindet. Ist es die mittlere, fährt er gerade aus auf das Tor, ist es eine der äußeren, bewegt er sich weiter zur Mitte. (vgl. Abschnitt 6.2.4)

6.1.2 Ladestation

Die Kontakte der Ladestation wurden mithilfe einfacher Kupferstreifen realisiert, die am Tor angebracht wurden. Diese sind direkt mit einem Ladegerät verbunden. (Näheres hierzu in Referenz [9])

6.1.3 Torerkennung

Für die Torerkennung waren mehrere Technologien denkbar. Möglichkeit 1 wäre gewesen das Tor mit einer Lichtschranke zu versehen. Möglichkeit 2 wäre die Entwicklung eines intelligenten Balls, der selbstständig erkennt ob er sich im Tor befindet und Möglichkeit 3 wäre die Konstruktion eines simplen Tasters, der durch den eintreffenden Ball betätigt wird. Möglichkeit 2 wurde schon zu Beginn verworfen, da der Ball selbst eine Komponente darstellen würde und deshalb nicht leicht austauschbar wäre und vor allem ebenfalls mit Strom versorgt werden müsste, was wieder eine Ladestation erfordert. Man müsste ihn so konzipieren, dass er durch die auf ihn

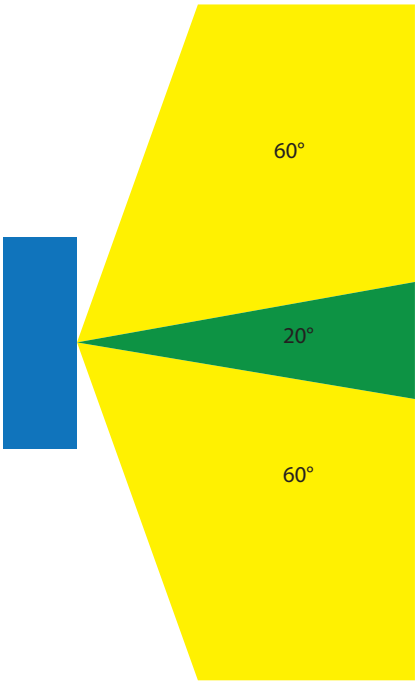


Abbildung 6.2: Schematischer Aufbau der Infrarot LED mit Abstrahlwinkel

einwirkenden Kräfte nicht zerstört würde, da sobald er mal nicht mehr funktionsfähig wäre, man nicht mehr spielen könnte bis ein neuer Ball gefertigt wurde.
Das Problem bei Möglichkeit 1 und 3 ist folgendes: Beide erkennen lediglich eine Bewegung im Tor, es kann aber nicht sichergestellt werden, ob die Bewegung tatsächlich von dem Ball aus geht. So ist es deshalb möglich, einfach mit dem Roboter an das Tor zu fahren und zu hoffen, dass der Roboter weit genug in das Tor hinein ragt. Die Entscheidung fiel schließlich auf die Variante mit dem mechanischen Taster. Im Gegensatz zu der Lichtschranke, ist dieser nicht anfällig gegen andere Lichteinwirkungen, die nur mit erheblichem Mehraufwand gefiltert werden könnten.
Der mechanische Schalter ist eine simple „Klappenkonstruktion“. Im Tor befindet sich eine Messingstange an der eine Klappe hängt. An der Rückseite dieser Klappe befinden sich Kupferstreifen, die, sobald ein Ball die Klappe nach hinten drückt, eine Verbindung zwischen zwei

Kontakten herstellen. Das bewirkt eine Zustandsänderung auf einem GPIO-Pin des RaspberryPi's, die vom Server erkannt wird.

6.2 Server

Der Server hat die Aufgabe, die gesamte Kommunikation zwischen Controllern und Robotern zentral zu verwalten und zu koordinieren. Auch die Spiellogik und Interpretation der Steuerbefehle ist Aufgabe des Servers. Außerdem müssen die Eingabe und Ausgabe der Peripheriegeräte verwaltet, bzw. gesteuert werden. Diese Peripheriegeräte sind die Ladestation, die gefunden werden muss, die Torerkennung, die verarbeitet werden muss und die Infrarot-LED, mithilfe der die Ladestation gefunden werden kann. Die Bilddaten, die vom Roboter versendet werden, müssen hier zwischengespeichert werden und anschließend an die Controller weiter gegeben werden.

6.2.1 Verbindung

Die Verbindungslogik besteht im Wesentlichen aus fünf Klassen, die in Abbildung 6.3 dargestellt sind. Verwaltet werden die Verbindungen in der Klasse *ConnectionManager*, der die Verbindungen für die Spiellogik bereit stellt, sobald ein Controller-Roboter paar verbunden wurde. Für die Verbindungen der Controller gibt es *WebsocketSocket*. Ein Objekt dieser Klasse dient als Schnittstelle der Kommunikation zwischen einem Controller und dem Server. Hierbei werden die Nachrichten im JSON-Format ausgetauscht, da dies eine Key-Value Kommunikation ermöglicht und für abstrakte Sprachen wie Javascript leichter zu verwenden ist als eine Byte-Orientierte Übertragung. Aufgerufen wird die Schnittstelle über die URL: `ws://ip_des_servers:8080/control/`. Für die Verbindungen der Roboter gibt es *UDPConnectionHandler*. Auch ein Objekt davon steht für genau eine Verbindung mit einem Roboter, jedoch werden die Befehle hier im Gegensatz zu den WebSockets Byte-Orientiert ausgetauscht. Um eine Verbindung mit einem Roboter herzustellen, wird in der Klasse *UDPSocketProvider* auf Port 44044 auf eingehende Pakete gewartet und anschließend ein Socket zu diesem Endpunkt bereit gestellt. Diese Klasse implementiert das *Runnable* Interface und wird als eigener Thread gestartet [6]. Auch die Authentifizierung wird in dieser Klasse abgehandelt, indem überprüft wird, ob die eingehenden Pakete einer der IP-Adressen der Roboter zugeordnet werden können. Da die Roboter statische IP-Adressen bekommen sollen ist diese Methode überhaupt möglich. Zusätzlich gibt es für den *UDPConnectionHandler* eine Hilfsklasse, die *ConnectionControl*. In dieser wird festgelegt, wie lange die Verbindung als offen gekennzeichnet werden soll, obwohl kein Paket ankommt. Auf der Controllerseite wird eine solche Implementierung nicht benötigt, da der WebSocket auf HTTP und damit auf TCP basiert. Ein Verbindungsabbruch ist dort auch ohne Timeout erkennbar.

6.2.2 Steueralgorithmen

Da die verschiedenen Steuervarianten unterschiedliche Steuerparameter erzeugen ist es für jede einzelne davon notwendig, diese zu interpretieren und in eine Form zu bringen, die mit der

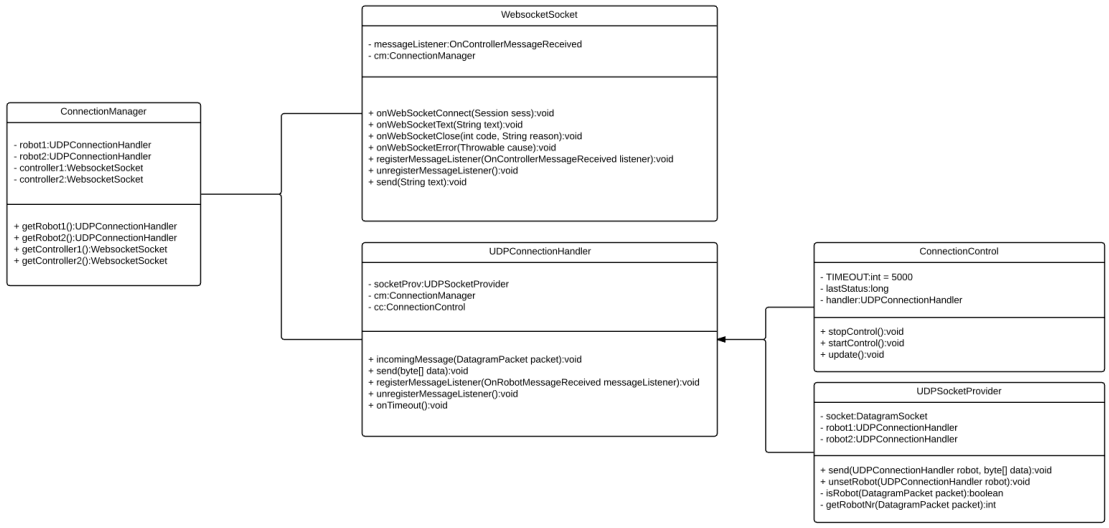


Abbildung 6.3: Klassendiagramm der Verbindungslogik

Schnittstelle zum Roboter kompatibel ist. Auf diese wird in den nächsten Abschnitten genauer eingegangen. Das heißt, am Ende der Übersetzung müssen die Geschwindigkeiten der Motoren links und rechts und ob ein Schuss ausgelöst werden soll. Unabhängig der Steuervariante muss der Ausgangsbefehl anhand des vorherigen Befehls nachkorrigiert werden. Da die Motoren des Roboters eine im Vergleich zur Haftreibung der Räder hohe Beschleunigung aufweisen, müssen die Beschleunigungen begrenzt werden, um durchdrehende Räder und damit ein unkontrollierbares Verhalten der Steuerung zu vermeiden. Dies wird realisiert, indem ein maximal möglicher folgender Motorwert anhand des vorhergehenden Motorwertes errechnet wird. Die Berechnungsformel ist relativ simpel, der neue Wert darf um maximal einen festen Wert x größer sein als der alte Wert. Das ist dadurch zu begründen, dass für die Überschreitung der Haftreibung der Räder lediglich die dort wirkende Kraft ausschlaggebend ist, die wiederum aus der Beschleunigung der Räder resultiert. Dieser Wert x wurde nach einigen Testversuchen auf 10 festgelegt. Für den Fall, das beide gewünschten Motorenwerte über dem errechneten erlaubten Maximum liegen, muss anschließend das Verhältnis von linkem zu rechtem Motor angepasst werden, da sonst beide werte gleichgestellt werden, auch wenn diese sich zuvor unterschieden hätten. Zur Veranschaulichung dieser Korrektur ist in Tabelle 6.1 ein Beispiel aufgeführt. Wie man sieht wünscht der Benutzer durch seine Eingabe einen leichten Bogen nach rechts zu fahren, jedoch würden nach der Maximalwertkorrektur beide Motoren gleich schnell bewegt werden. Erst nach der Anpassung des Verhältnisses fährt der Roboter tatsächlich den gewünschten Radius.

Differential Steuerung

Da die eingehenden Steuerparameter direkt die Motorenbefehle für Links und Rechts beinhalten (vgl. Abschnitt 6.4.1), ist hier keine weitere Interpretation notwendig.

	letzte Motorbefehle	neue Motorbefehle	nach Korrektur (ohne Verhältnis)	nach Korrektur (mit Verhältnis)
Motor Links	50	80	60	06
Motor Rechts	50	70	60	52

Tabelle 6.1: Motor Werte nach Durchlaufen des Korrekturalgorithmus

RC-Remote Steuerung

Die eingehenden Steuerparameter sind der Vorschub und die Richtung. Der Vorschub lässt sich direkt auf die Motoren übertragen, lediglich die Richtung muss mittels einer Formel berechnet werden. Der Interval indem sich der Wert der Richtung befindet ist [-50, 50]. An dieser Stelle bietet sich eine Exponentialformel an. Im Allgemeinen lautet diese:

$$ratio_{LR} = a^{\frac{x}{50}}$$

, wobei der Parameter a bestimmt, wie groß das Verhältnis bei maximaler Auslenkung ist und x den momentanen Richtungswert angibt. Nach einigen Testläufen wurde der Parameter a zu 2 bestimmt. Dadurch bewegt sich der linke Motor mindestens halb so schnell und maximal doppelt so schnell wie der rechte Motor. Der zugehörige Code ist in Listing 6.1 zu sehen.

```
double ratioLR = Math.pow(2,((double) dir/50));
if (ratioLR > 1) {
        robot.setMotorRight(
                (int) ((double) robot.getMotorLeft() / ratioLR)
        );
}
if (ratioLR < 1) {
        robot.setMotorLeft(
                (int) ((double) robot.getMotorRight() * ratioLR)
        );
}
```

Listing 6.1: Umrechnung der RC Parameter in Motorbefehle

Pfeiltasten Steuerung

Diese Steuervariante hat einen bedeutenden Nachteil. Da die Eingangsparameter die momentan gedrückten Tasten sind (siehe Abschnitt 6.3), gibt es keine Zwischenwerte für die Beschleunigung oder Richtung. Die Taste nach vorne bedeutet hierbei einen Motorschub für beide Seiten von 100%. Für jede zusätzliche Richtungstaste, also links oder rechts, werden auf der dementsprechenden Seite 25% abgezogen. Wird nur eine Richtungstaste ohne eine Beschleunigungstaste gedrückt, werden die Motoren auf -15%/15% bzw. 15%/-15% gesetzt, was einer Drehung auf der Stelle entspricht. Diese „Alles oder Nichts“Steuerung führt im Spiel zu einer nicht flüssigen, jedoch

durch die Möglichkeit der Drehung auf der Stelle zu einer annehmbaren Steuerung. Der Code dafür ist in Listing 6.2 gezeigt.

```
if (JSONUtil.jsonObjToBoolean(command.get("up"))) {
        motorLeft += 100;
        motorRight += 100;
}
if (JSONUtil.jsonObjToBoolean(command.get("down"))) {
        motorLeft -= 100;
        motorRight -= 100;
}
if (JSONUtil.jsonObjToBoolean(command.get("right"))) {
        motorRight *= 0.75;
}
if (JSONUtil.jsonObjToBoolean(command.get("left"))) {
        motorLeft *= 0.75;
}

if (motorRight == 0 && motorLeft == 0 &&
JSONUtil.jsonObjToBoolean(command.get("right"))) {

        motorLeft = 20;
        motorRight = -20;
}

if (motorRight == 0 && motorLeft == 0 &&
JSONUtil.jsonObjToBoolean(command.get("left"))) {

        motorLeft = -20;
        motorRight = 20;
}
```

Listing 6.2: Umrechnung der Pfeiltasten in Motorbefehle

6.2.3 Bildübertragung

Wie in der Schnittstellenbeschreibung in Kapitel 1.1 beschrieben, werden die Bilddaten vom Roboter an die Statusnachrichten angehängt. Hierbei kennzeichnen die Bytes FF D8 den Beginn eines neuen Frames. Diese Kombination tritt in JPEG-kodierten Bildern immer nur am Anfang des Bildes auf, wodurch es möglich ist, Bilddaten über mehrere Pakete zu verteilen. Diese Daten werden in der *Player*-Klasse gepuffert, bis das Bild vollständig ist. Anschließend wird es noch base64 kodiert und sofort an die Controller weiter gesendet und nicht mit den regelmäßigen Statusbefehlen, um keine unnötige Zeitverzögerung einzubauen. Die base64 Kodierung wurde im Gegensatz zur einfachen Byte-Orientierten übertragen bevorzugt, damit mit den Controllern in einer höheren Datenabstraktionsebene kommuniziert werden kann. Das vereinfacht den Code vor allem in der Webanwendung, da diese in Javascript implementiert ist. Ein großer Nachteil dieser Kodierung ist der größere Speicherbedarf, der die Übertragung aufgrund höheren

Traffics langsamer macht und dadurch zu Verzögerungen führt, die letztendlich im Verlust der Steuerbarkeit resultieren können. Nach ersten Tests ist die Bildrate vom Roboter jedoch so gering, dass diese wenigen Frames rechtzeitig übertragen werden können. Zusätzlich sind die Frames in einer Größenordnung, in der 33 % Mehrbedarf noch zu keiner Netzwerküberlastung führen.
Der Servercode für die Pufferung der Bilddaten des Roboters ist in Listing 6.3 aufgeführt.

```
private void handlePictureData(byte[] data) {
        for (int i = 0; i < data.length; i++) {
                if (data[i] == (byte) 0xFF &&
                        i < data.length -1 &&
                        data[i+1] == (byte) 0xD8 &&
                        pictureBuf.size() != 0) {

                        sendPicture();
                        pictureBuf.reset();
                }
                pictureBuf.write(data[i]);
        }
}
```

Listing 6.3: Logik für die Pufferung der Bilddaten

6.2.4 Torfindung

Mit der Torfindung ist der Vorgang gemeint, um die Ladestation, die im Tor integriert ist zu finden und selbstständig anzufahren. Über die Statusmeldungen des Roboters erfährt der Server wie hoch der Akkustand des Roboters ist. Sobald der Roboter ein Minimum des Akkustands unterschreitet, greift die Torfindung ein. Diese wird in der Klasse *EnergyManager* implementiert (siehe Abb. 6.5). Dabei wird die Torfindung sofort aktiviert, wenn sich ein Roboter mit dem Server verbindet und nicht erst wenn noch ein Controller hinzu kommt. Tritt der Fall ein, dass der Akku den Schwellenwert unterschreitet, werden die Infrarot-LEDs über die Klasse *BlinkTask* angeschaltet. Über das „seeGoal“Flag (vgl. Schnittstellenbeschreibung, Kapitel 1.1) kann erkannt werden, ob das zum Roboter gehörende Tor in Sicht ist oder ob sich der Roboter neben der zum Tor führenden LED befindet. Bekommt der Server vom Roboter drei aufeinander folgende Stati, in denen der Roboter meldet, dass eine LED in Sicht ist, dann fährt der Roboter in die dementsprechende Richtung. Ist der Roboter vom eigenen Tor abgewandt, beginnt sich der Roboter zu drehen, solange bis entweder mindestens ein Status mit positiver Rückmeldung kommt, oder bis ein Timeout abgelaufen ist. Um die Drehung zu unterbrechen genügt deswegen bereits ein Status mit einer Sichtung, um die Möglichkeit einer korrekten Sichtung nicht durch zu schnelles drehen sofort zu verwerfen. Der Timeout bewirkt, dass sich der Roboter nicht endlos um sich selbst dreht, falls er außerhalb des Abstrahlwinkels der LEDs ist. Tritt ein Timeout auf, fährt der Roboter wenige Sekunden in die aktuelle Richtung, um seine Ausgangsposition zufällig zu ändern. Nach diesem Prinzip wird früher oder später eine LED erkannt und das Tor und damit die Ladestation kann angefahren werden.
Umgesetzt wurde die Torfindung mithilfe eines Zustandsautomaten (Abbildung 6.4). Da der Code einige Zeilen umfasst wird an dieser Stelle auf das Projekt verwiesen [6].

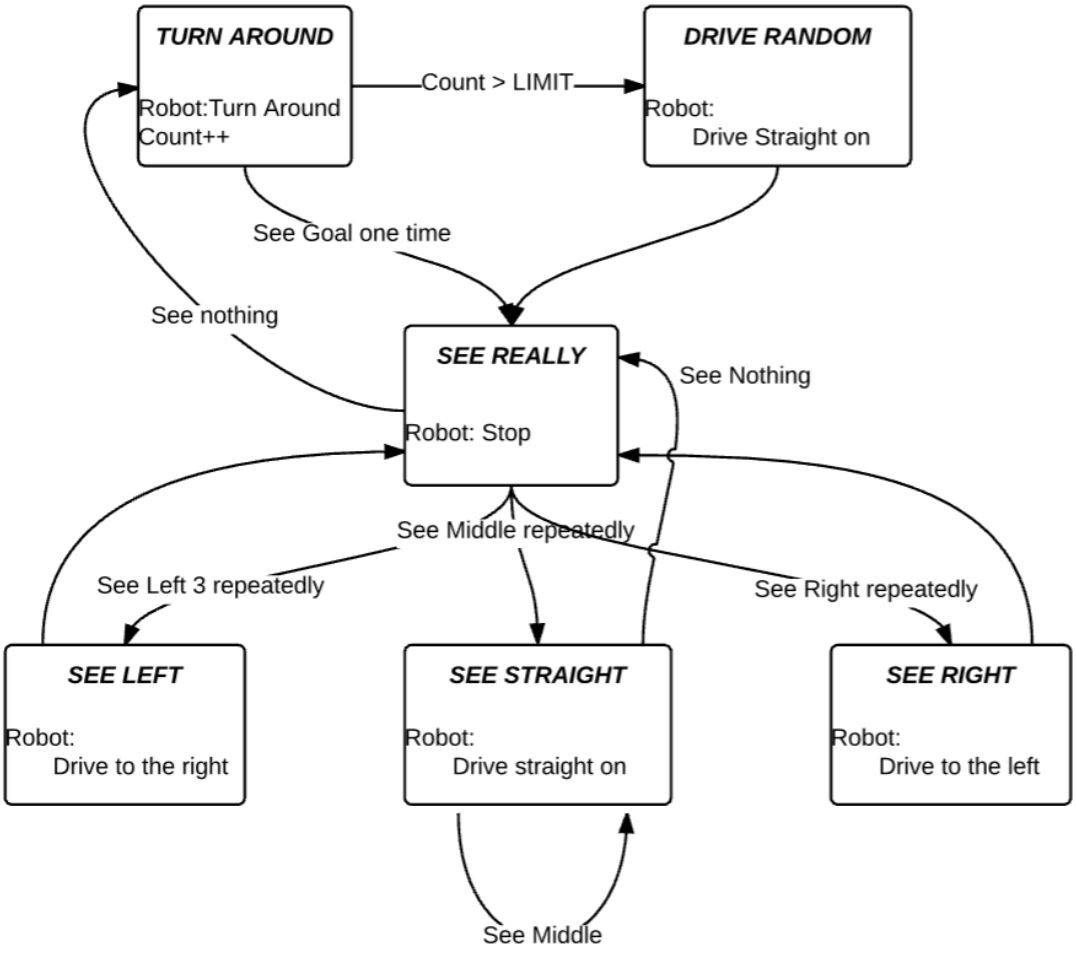


Abbildung 6.4: Zustandsdiagramm der Torfindung

Diese Methode hier gut geeignet, da mit jedem Roboter Status dieselbe Routine aufgerufen wird. Mithilfe der Zustände lässt sich hier gut auf die unterschiedlichen Anforderungen reagieren. Auf den gleichen Status vom Roboter muss unterschiedlich reagiert werden, je nachdem in welchem Zustand sich der Roboter befindet. Der Code hierzu befindet sich in der Klasse *EnergyManager* [6] Der Nachteil bei dieser Implementierung ist natürlich die unsichere Verhaltensweise des Roboters, wenn er außerhalb der LED-Reichweite ist. Diese führt im Worst-Case zu einer kompletten Entladung des Akkus bis zur Bewegungsunfähigkeit, bevor die Station gefunden wurde. Da der Großteil des Spielfelds jedoch von den LEDs abgedeckt wird, ist dieser Fall unwahrscheinlich. Ein Vorteil dieser Methode im Gegensatz zu zum Beispiel einer Kamera basierten Positionsbestimmung ist die Unabhängigkeit des Spielfelds. Lediglich die LEDs müssen am Tor vorhanden sein, jedoch keine speziellen Konturen oder andere feste Muster, an denen der Roboter sich mithilfe der Kamera orientieren kann. Abgesehen davon ist die Torfindung mittels IR-LEDs vergleichsweise einfach zu realisieren.

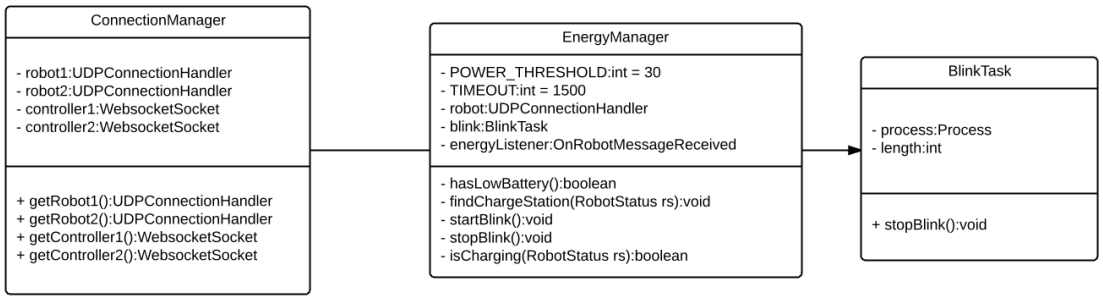


Abbildung 6.5: Klassendiagramm der Torfindung

6.3 Webanwendung

Die Webanwendung stellt eine Steuerungsvariante dar, die von überall aus genutzt werden kann, wo eine Tastatur und ein Browser vorhanden sind. Sie ist in Javascript implementiert und wird direkt von einem ebenfalls auf dem Server installierten Jetty-Webserver gehostet. Aufgerufen werden kann er über die IP-Adresse des Host-Systems unter Port 8080. Dabei funktioniert die Kommunikation genau wie bei der Android-Anwendung über einen WebSocket, der die gegebene Kommunikationsschnittstelle für die Controller auf dem Server nutzt. Die Oberfläche besteht lediglich aus einem Container für die Bilddaten. Über die gewohnte Pfeiltasten-Steuerung kann der Roboter gesteuert werden. Die gedrückten Tasten werden hierbei nur an den Server weitergegeben, erst dort werden sie verarbeitet (siehe Abschnitt 6.2.2).

6.4 Android-Anwendung

Als mobile Anwendung für Endgeräte wurde Android ausgewählt, da diese Plattform einige Vorteile bietet. Im Gegensatz zu anderen mobilen Betriebssystemen ist es für Android möglich, ohne Lizenzen oder andere Absprachen zu treffen, Anwendungen zu entwickeln. Außerdem kann hiermit auch in Java entwickelt werden und der Zugriff auf die Sensorik des Geräts ist bereits sehr gut vorbereitet. Die Anwendung vereint mehrere Steuerarten, die sich in ihrem Abstraktionsgrad unterscheiden. Diese werden in den folgenden Abschnitten beschrieben.

Die Software ist wie in Anhang 8.3 dargestellt konzipiert. Den Mittelpunkt der Android-Anwendung stellt die abstrakte Klasse *ControlActivity* dar. Jede Steueractivity erbt von dieser Klasse. Damit ist es möglich, das gesamte gemeinsame Verhalten der Steuerungen (Anzeigen des Spielstands und der Bilddaten, Reaktion auf Verbindungsabbrüche etc..) in einer Zentrale zu implementieren. Jede Subklasse implementiert hierbei lediglich die Art, wie Befehle akquiriert werden.

Der Schussauslöser wird in jedem Fall über die Beschleunigungssensorik des Geräts implementiert. Hierbei wird die auf das Gerät wirkende Beschleunigung gemessen und ab einem gewissen Grenzlevel der Schuss herbei geführt, was darin resultiert, dass der Schuss über ein kurzes Schütteln des Geräts erreicht wird. Wie auch in der Serverimplementierung gibt es hier eine Klasse, die aktuelle Befehlsänderungen puffert und mit einer Frequenz von 10Hz über den

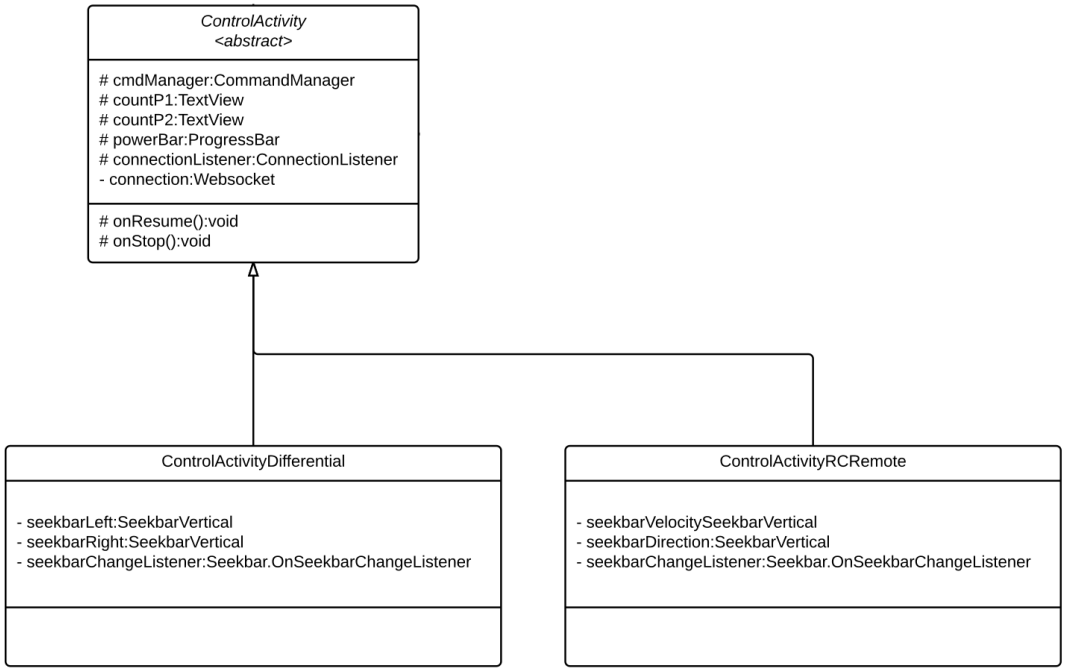


Abbildung 6.6: Klassendiagramm ControlActivities

Websocket an den Server sendet. Die Klassen *Websocket*, *CommandManager* und *ControlActivity* bilden im Klassendiagramm eine Art Regelkreis der wie folgt interpretiert werden kann. In der *ControlActivity* wird der Befehl akquiriert, im *CommandManager* dann in einen Befehl umgewandelt und über den Websocket versendet. Über den *Websocket* werden anschließend die Bilddaten vom Server empfangen und an die *ControlActivity* weitergegeben, worauf der Benutzer reagieren kann und seine Steuerbefehle dementsprechend anpasst.

6.4.1 Differential Steuerung

Diese Steuerung ist die am wenigsten abstrahierte Steuerung. Es gibt zwei vertikal ausgerichtete Slider, die jeweils für den linken oder den rechten Motor stehen und einen Wertebereich von -100 - +100 abdecken. Bewegt man den Slider nach oben, beschleunigt der Roboter vorwärts, nach unten dementsprechend rückwärts. Wie oben bereits erwähnt, wird der Schuss über die Sensorik des Geräts ausgelöst. Mit dieser Steuerung sind die genauesten Bewegungen möglich, da man das volle Potenzial ausschöpfen kann. Im Gegensatz zu den anderen Steuerungen werden keine Befehle limitiert. Jedoch wird die Steuerung dadurch sehr schwierig und für Anfänger nur schwer benutzbar.

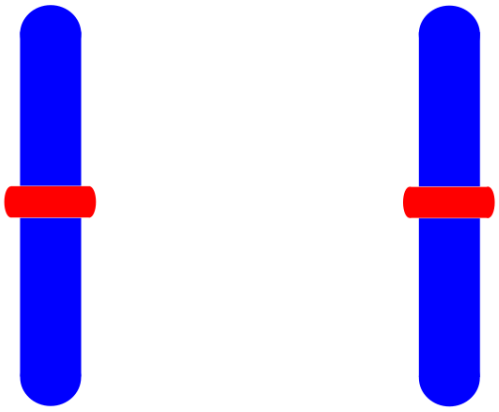


Abbildung 6.7: Schematische Darstellung der Differential Steuerung

6.4.2 RC-Remote Steuerung

Ähnlich wie in der Differential Steuerung kommen auch hier Slider zum Einsatz. Der vertikale Slider steht hierbei für die Beschleunigung des Roboters und der horizontale für die Winkel, den der Roboter einschlägt, ähnlich wie man es von einer Fernbedienung von ferngesteuerten Spielzeugautos kennt. Auch hier wird der Schuss über die Sensoren ausgelöst. Der Beschleunigungsregler hat einen Wertebereich von -100 bis +100. Dadurch bildet er genau die Leistung ab, die jeder Motor annehmen soll (solange keine Richtung angegeben wird). Der Richtungsregler hat einen Wertebereich von -50 - +50, wobei der tatsächliche Intervall keine Rolle spielt, da der Winkel, der eingeschlagen wird relativ zum Gesamtintervall berechnet wird (siehe 6.2.2). Ein großer Vorteil gegenüber den Differential Steuerung ist die leichtere Bedienbarkeit. Da nicht mehr jeder Motor separat angesteuert werden muss ist es zum Beispiel sehr einfach, den Roboter nur gerade aus zu steuern. Jedoch ist der maximale Winkel den man einschlagen kann begrenzt und auch die Möglichkeit sich auf der Stelle zu drehen fällt bei dieser Steuerung weg. Die Umsetzung erfolgt in der Serverlogik und ist in Abschnitt 6.2.2 beschrieben.

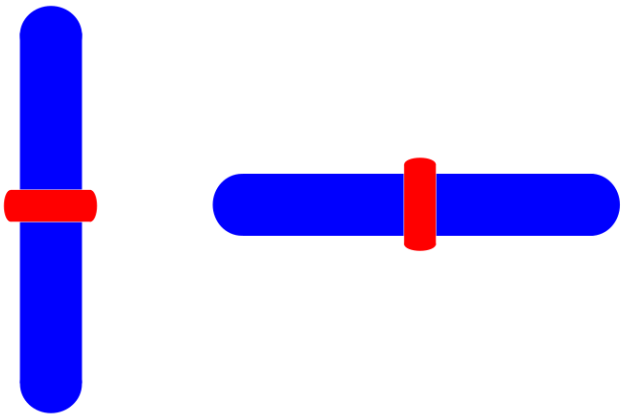


Abbildung 6.8: Schematische Darstellung der RCRemote Steuerung

6.5 RobotSimulator

Einige Komponenten im Code des Servers benötigen gewisse Informationen vom Roboter, wie zum Beispiel die Bildübertragung. Da die Roboter gleichzeitig zur Steuerung entwickelt wurden und deswegen nicht rund um die Uhr zur Verfügung standen war es hilfreich einen Simulator zu entwickeln, der Befehle im exakt selben Format versendet. So wurde die gesamte Roboter-Server Kommunikation simuliert. Hierzu wurde ein Paket mit festen Werten erstellt, das genau wie der Roboter auch, alle 100ms per UDP an den Server gesendet wird.
Um die Bilddatenpufferung des Servers testen zu können, wurden zwei Beispielbilder von der Festplatte in ein großes Byte-Array geladen. So war es möglich die Eigenschaft des Roboters zu simulieren, Bilddaten ohne Trennung, also keine separierten Bilder zu senden (vgl. Listing 6.4).

```
puffer = new byte[puffer1.length + puffer2.length];
System.arraycopy(puffer1, 0, puffer, 0, puffer1.length);
System.arraycopy(puffer2, 0, puffer, puffer1.length, puffer2.length);
                                .
                                .
                                .
private static void fillBuffer() {
        // puffer is byte[] with two pictures concatenated
        // data is the packet to send
        for (int i = 0; i < 1400; i++) {
                data[i + 12] = puffer[pointer];
                pointer++;
                if (pointer == puffer.length) {
                        pointer = 0;
                }
        }
}
```

Listing 6.4: Quellcode des RobotSimulators zum Testen der Kameraübertragung

Ein weiterer Vorteil dieser Methode, abgesehen von der ständigen Verfügbarkeit, lag an der klaren Abtrennung von auftretenden Problemen. Wurde der Simulator erst einmal richtig konfiguriert, war die Schnittstelle für beide Seiten definiert. Funktionierte hinterher im Test etwas bei der Kommunikation nicht, konnten lediglich die Pakete des Simulators mit denen des Roboters verglichen werden. Unterschieden sie sich, war der Fehler auf der Roboter Seite.

7 Fazit und Ausblick

Zum Ende der Arbeit erfolgt nun ein Fazit, indem auf die in der Einleitung definierten Ziele noch einmal eingegangen und diskutiert werden soll, ob die Vorgehensweise zieldienlich war. Anschließend gibt es einen Ausblick auf mögliche Erweiterungen und Verbesserungen der bestehenden Komponenten.

7.1 Fazit

Das Ziel der Arbeit war es, eine Steuerung für einen eigens entwickelten Roboter zu entwerfen, die es dem Benutzer ermöglicht mit seinem Smartphone oder Laptop den Roboter vor Ort, oder mithilfe der Kameraübertragung zu steuern, damit ein Fussballspiel ermöglicht werden kann. Dabei spielt es eine tragende Rolle, dass die Rechtzeitigkeit im Sinne der Übertragungszeit für Kamerabild und Befehlslatenz sicher gestellt ist. Hierfür war es notwendig eine geeignete Übertragungstechnologie zu finden und die Schnittstelle zum Roboter effizient zu gestalten. Auch die Aufgabe, den Roboter wartungsfrei zu halten was den Ladevorgang angeht, galt es zu bewältigen.
Die verzögerungsfreie Bedienbarkeit des Roboters über die Steuercontroller bestätigt die Rechtzeitigkeit in Bezug auf die Steuerung. Dies war zu erwarten, da die maximale Verzögerung von 100ms (vgl. 2.1) zwischen dem Wunsch und dem Eintritt einer Richtungsänderung zu keiner merklichen Beeinträchtigung des Spielflusses führen. Die Kameraübertragung jedoch hat aufgrund der niedrigen Framerate des Roboters gewisse Verzögerungen. Diese Verzögerungen führen bei einer hohen Geschwindigkeit des Roboters zu erschwertem Handling, die jedoch nicht von der Übertragung, sondern von der Roboterhardware zu verantworten sind.
Die Aufgabe den Roboter selbstständig an die Ladestation zu führen wurde meiner Meinung nach nur teilweise erfüllt. So ist es zwar möglich die Ladestation zu finden und auch anzufahren, jedoch hängt der Erfolg noch am Treffen der Ladekontakte. Da dies jedoch nur äußere Umstände sind, die von einer geschickteren Ladestation ausgemerzt werden könnten und die Station ja tatsächlich gefunden wird, halte ich diesen Punkt für wenig relevant, da er sich außerhalb des Interessengebiets befindet.
Somit lässt sich abschließend sagen, dass es durchaus möglich ist einen eigens entwickelten Fussballroboter ohne für den Nutzer erkennbare Verzögerungen zu steuern. Lediglich die Bildübertragung bringt eine gewisse Unrechtzeitigkeit ins Spiel, die aber vermutlich nicht an der Übertragung von Server zu Controller hängt, sondern an der Übertragungskapazität des Roboters. Wäre diese höher ausgefallen, hätte man sich sicherlich für ein Streamingprotokoll entscheiden müssen.
Die Anforderungen für diese Arbeit wurden jedoch zufriedenstellend erfüllt. Es wurde eine Steuerung für ein Roboter Fussballspiel entwickelt, bei dem die Roboter mithilfe von mobilen Endgeräten gesteuert werden.

7.2 Ausblick

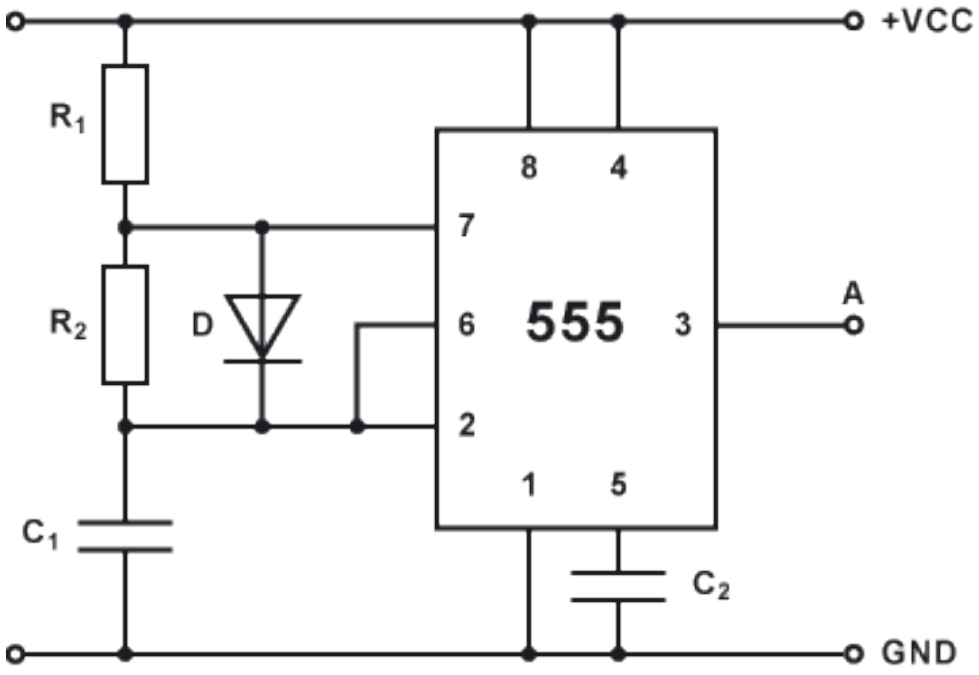
Das Thema Steuerung von Robotern bietet generell viel Spielraum wenn es um Erweiterungen geht. Zunächst wird jedoch auf die bereits verwendeten Komponenten eingegangen.
Auf die Ladestation in Bezug auf das Design der Kontakte wird im Nachfolgenden nicht weiter eingegangen, da dies den Rahmen des eigentlichen Themas zu sehr übertreten würde. Jedoch wäre für die Torfindung an sich eine Positionsbestimmung des Roboters sehr hilfreich. Der Roboter müsste sich nicht ständig neu orientieren und könnte auf Anhieb den richtigen Weg zur Station finden. Dies ließe sich zum Beispiel mit einer über dem Spielfeld hängenden Kamera realisieren.
Auch die Torerkennung bietet noch Verbesserungspotenzial. So könnte man die Tore mit Torkameras ausstatten, die den Ball erkennen. Das schließt Fehlerkennungen durch Rempler oder Fremdkörper aus.

Führt man diesen Gedanken Positionsbestimmung weiter, so wäre es dadurch auch möglich eine zentrale Steuerung seitens des Servers zu entwickeln. Hierbei würde der Server alle Steuerbefehle anhand der aktuellen Position des Roboters errechnen und diese an den Roboter senden, also eine Art Computergegner, um das Spiel auch ohne zweiten menschlichen Gegner spielen können.
Die Funktion den Roboter von außerhalb des Universitätsnetzwerks zu steuern ist zwar vorhanden, jedoch benötigt der Server hierfür auch eine öffentliche IP-Adresse. Wird ihm eine solche zugeteilt, wäre die Teilnahme am Spiel von jedem Ort mit Internetzugang möglich.

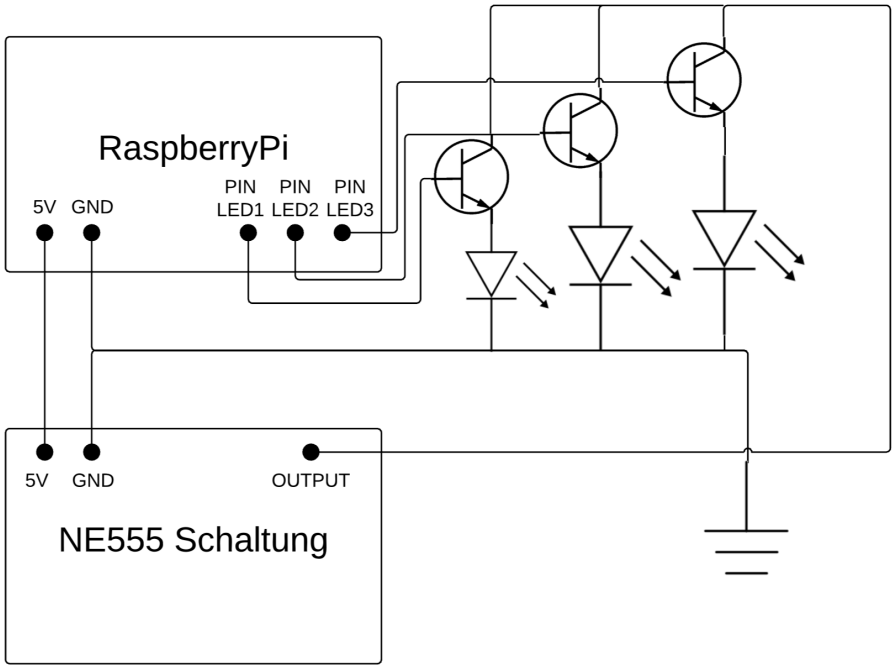
8 Anhang

Bauteil	Wert
R1	50 $k\Omega$
R2	50 $k\Omega$
C1	380 pF
C2	10 nF

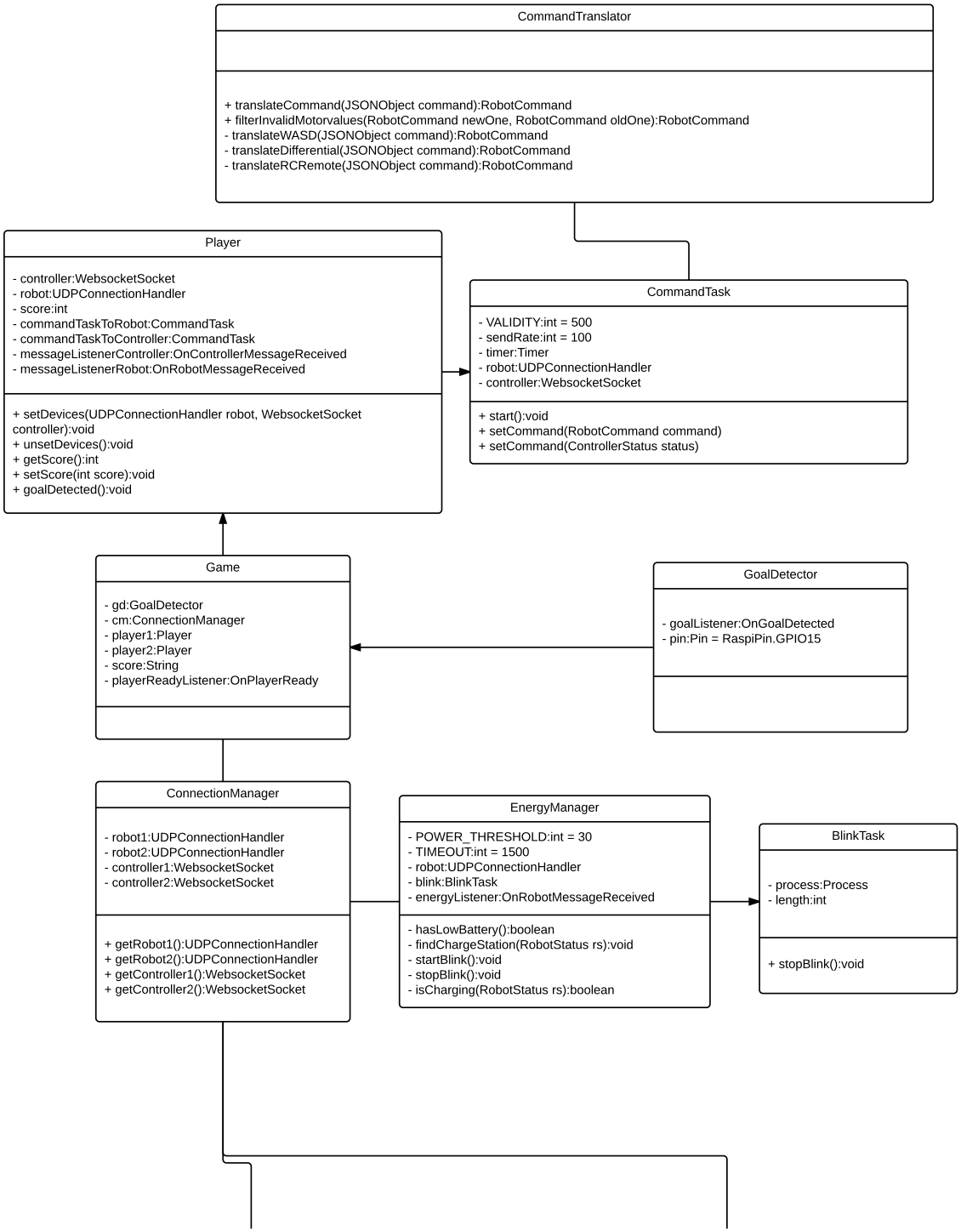
(a) Wertetabelle für die Timerbauteile in Schaltung 8.1b laut Berechnungsformel aus Quelle [3]



(b) Schaltplan des Timerbausteins [3]



(c) Gesamter Schaltplan der Infrarot-LEDs


CommandTranslator
+ translateCommand(JSONObject command):RobotCommand
+ filterInvalidMotorvalues(RobotCommand newOne, RobotCommand oldOne):RobotCommand
- translateWASD(JSONObject command):RobotCommand
- translateDifferential(JSONObject command):RobotCommand
- translateRCRemote(JSONObject command):RobotCommand
Player
- controller:WebsocketSocket
- robot:UDPConnectionHandler
- score:int
- commandTaskToRobot:CommandTask
- commandTaskToController:CommandTask
- messageListenerController:OnControllerMessageReceived
- messageListenerRobot:OnRobotMessageReceived
+ setDevices(UDPConnectionHandler robot, WebsocketSocket controller):void
+ unsetDevices():void
+ getScore():int
+ setScore(int score):void
+ goalDetected():void
CommandTask
- VALIDITY:int = 500
- sendRate:int = 100
- timer:Timer
- robot:UDPConnectionHandler
- controller:WebsocketSocket
+ start():void
+ setCommand(RobotCommand command)
+ setCommand(ControllerStatus status)
Game
- gd:GoalDetector
- cm:ConnectionManager
- player1:Player
- player2:Player
- score:String
- playerReadyListener:OnPlayerReady
GoalDetector
- goalListener:OnGoalDetected
- pin:Pin = RaspiPin.GPIO15
ConnectionManager
- robot1:UDPConnectionHandler
- robot2:UDPConnectionHandler
- controller1:WebsocketSocket
- controller2:WebsocketSocket
+ getRobot1():UDPConnectionHandler
+ getRobot2():UDPConnectionHandler
+ getController1():WebsocketSocket
+ getController2():WebsocketSocket
EnergyManager
- POWER_THRESHOLD:int = 30
- TIMEOUT:int = 1500
- robot:UDPConnectionHandler
- blink:BlinkTask
- energyListener:OnRobotMessageReceived
- hasLowBattery():boolean
- findChargeStation(RobotStatus rs):void
- startBlink():void
- stopBlink():void
- isCharging(RobotStatus rs):boolean
BlinkTask
- process:Process
- length:int
+ stopBlink():void

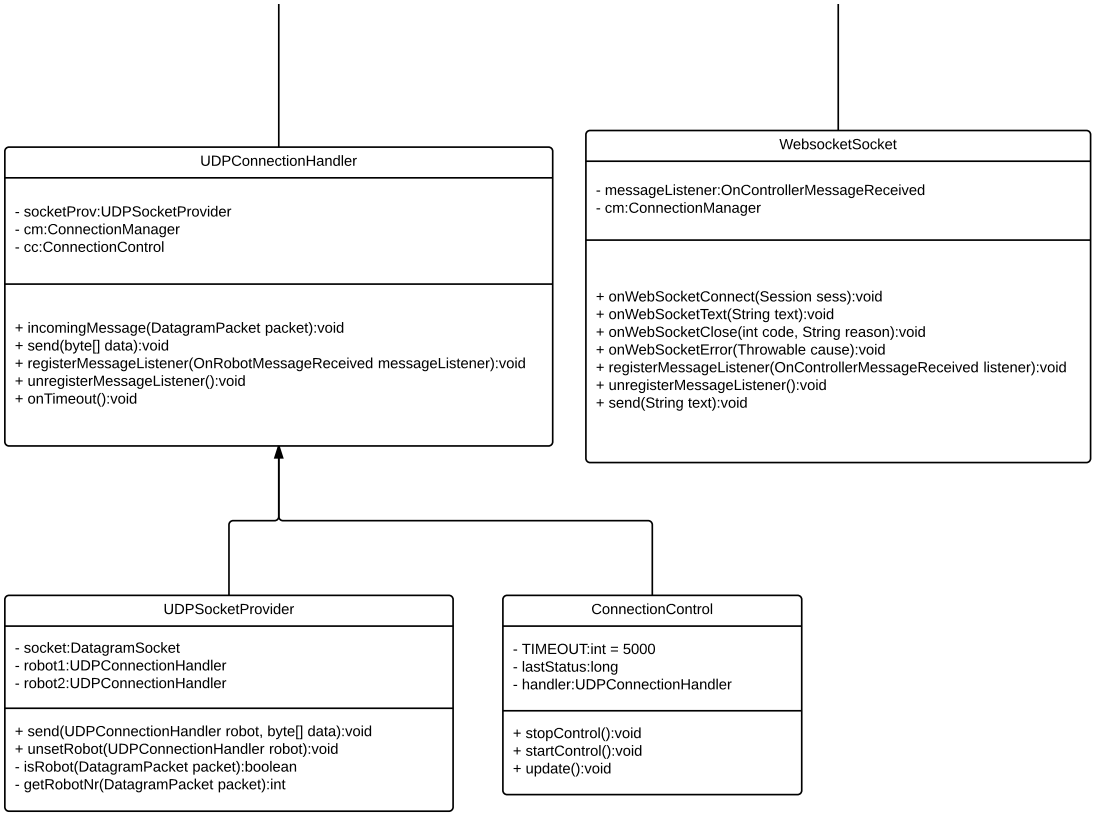


Abbildung 8.2: Klassendiagramm des Servers

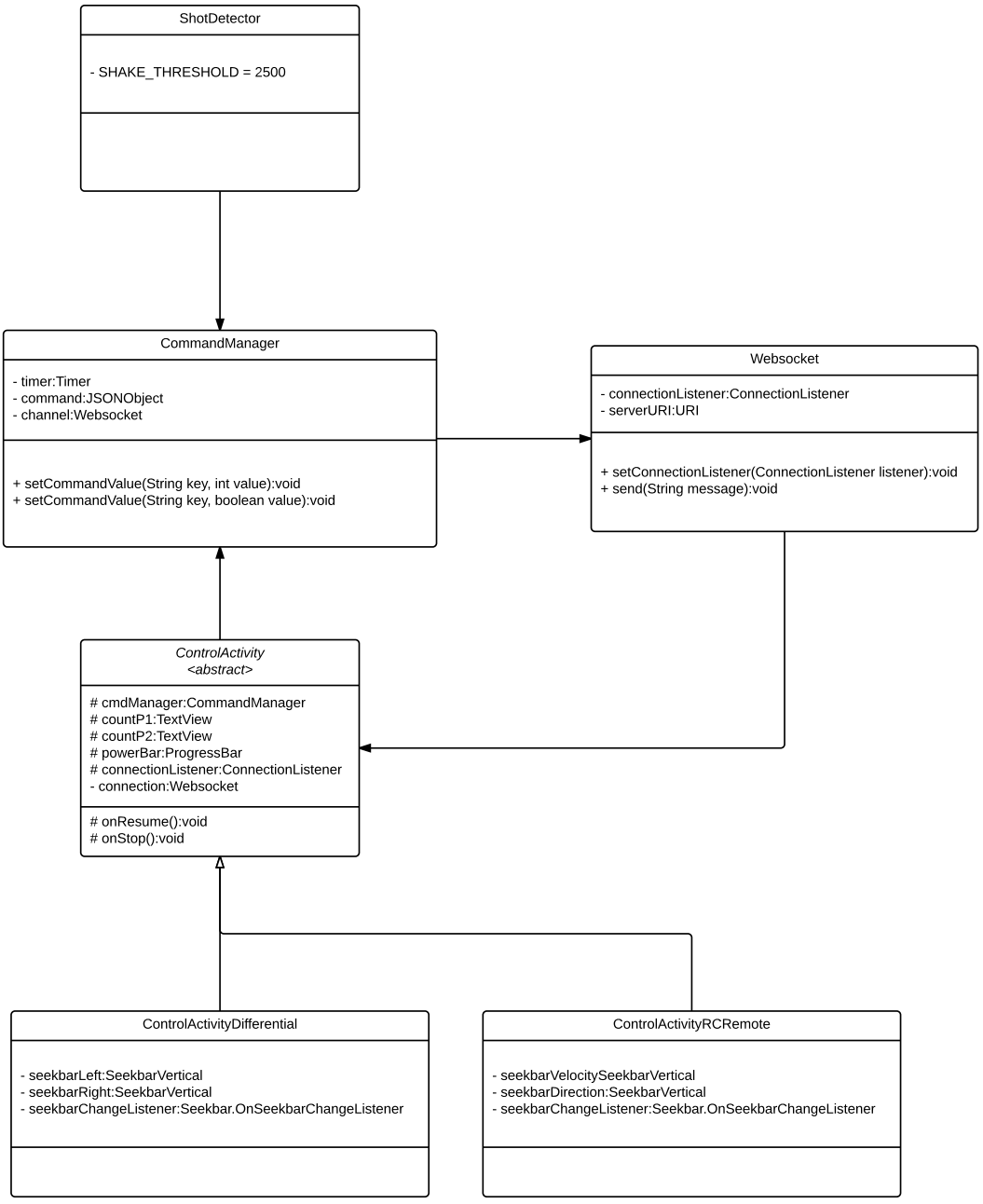


Abbildung 8.3: Klassendiagramm der Android Anwendung

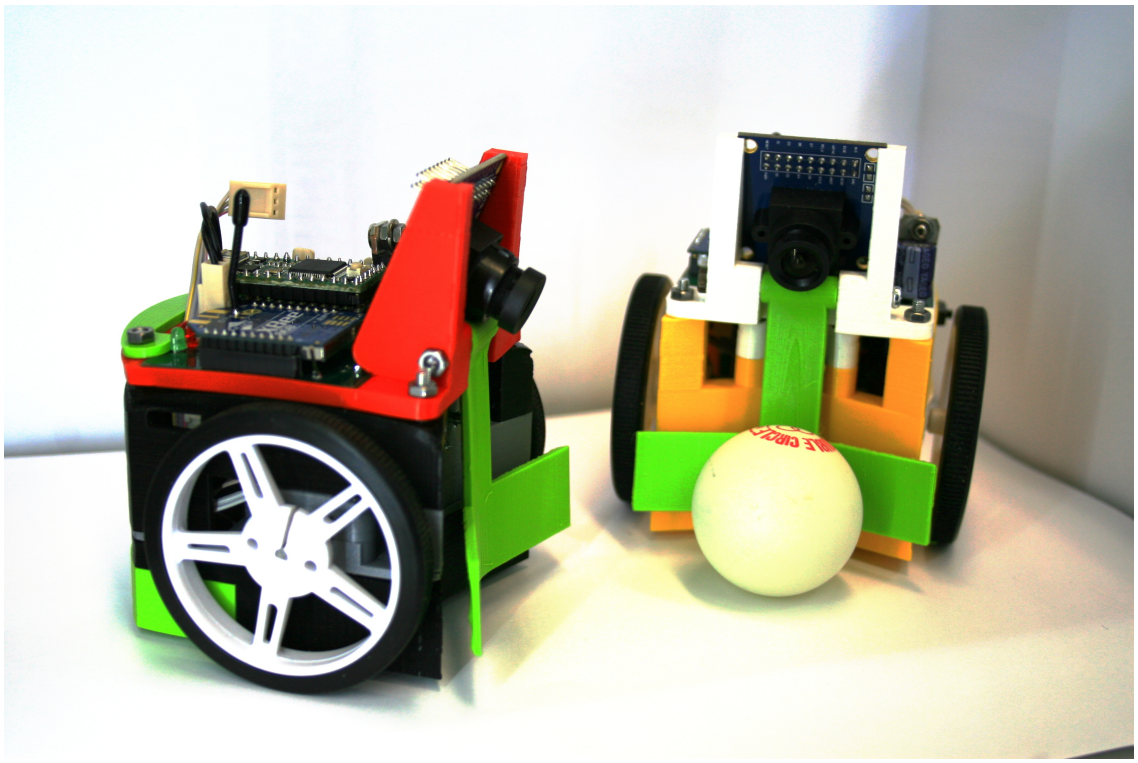


Abbildung 8.4: Die Roboter

Literaturverzeichnis

[1] : *Jetty*. – URL http://www.eclipse.org/jetty/

[2] : *Mobile/Tablet Top Operating System Share Trend*. – URL http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1&qpct=4&qpsp=175&qpnp=12&qptimeframe=M&qpcustomd=

[3] : *NE555 als astabile Kippstufe / astabiler Multivibrator*. – URL http://www.elektronik-kompendium.de/sites/slt/0310131.htm

[4] : *RoboCup*. – URL https://www.robocup.de

[5] : *Solid Edge*. – URL www.siemens.com/SolidEdge

[6] : *Steuerung eines Fussballroboters - Code*. – URL https://github.com/patrick225/Bachelorarbeit

[7] Schwartz, Mischa: *Mobile Wireless Communications*

[8] Tanenbaum, Andrew S.: *Computernetzwerke 4., überarbeitete Auflage*

[9] Ulbrich, Alexander: *Entwicklung eines Fussballroboters*

[10] Weber, Michael: *Mobile and Ubiquitous Computing - Vorlesungsfolien*

Erklärung

Ich, Patrick Lutz, Matrikelnummer 752774, erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Patrick Lutz