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. Stefan Wesner

Betreuer:
Dipl.-Inf. Lutz Schubert

Fassung 18. November 2015

Satz: PDF-LaTeX 2ε

Abstract

Automation und eingebettete Systeme gehören in der heutigen Zeit bereits zum Standard. In der Industrie sind Roboter schon lange nicht mehr wegzudenken. Sie erledigen sowohl hoch präzise, als auch robuste, kraft-aufwendige Arbeiten wesentlich genauer und schneller als ein Mensch es je können wird. Ein weiteres Einsatzgebiet von Robotern sind für Menschen unzugängliche oder gefährliche Orte, wie zum Beispiel mit Gas gefüllte Höhlen oder sogar Orte außerhalb der Erdatmosphäre (zB. Mars-Rover). Doch ein weiterer, immer stärker anwachsender Anwendungszweig von Robotern ist die Unterhaltungsbranche. Diese Bachelorarbeit beschäftigt sich mit der Ansteuerung und Bedienbarkeit eines eigens entwickelten Fussballroboters.
Die Konstruktion der akkubetriebenen Roboter inklusive Softwareschnittstelle ist Thema einer weiteren Bachelorarbeit, auf die hier nur verwiesen wird.

Ziel dieser Bachelorarbeit ist es, ein Fussballspiel zwischen zwei Benutzern zu ermöglichen. Hierbei bilden Smartphones oder andere Computer mit einem Webbrowser die Fernsteuerung für die Roboter. Über diese Fernsteuerung ist es möglich die Roboter zu steuern und mithilfe eines am Roboter befestigten Schussapparats, ein Tor zu erzielen. Am Spielfeldrand sind wie im Fussball üblich Tore aufgestellt. Fällt ein Tor, findet eine Torerkennung statt und der entsprechende Spielstand wird an dem Bediengerät angezeigt. Außerdem findet eine Kameraübertragung von den Robotern zu den Bediengeräten statt, sodass das Spiel auch gespielt werden kann, ohne sich in unmittelbarer Nähe der Roboter zu befinden. Ebenfalls Teil der Arbeit ist es zu gewährleisten, dass das Spiel auch ohne manuelles Eingreifen spielbar bleibt. Dafür ist es notwendig, dass die Roboter bei niedrigem Akkustand selbstständig die Ladestation aufsuchen und sich automatisch laden. Damit die Roboter frei und ohne Hindernisse fahren können, findet die Kommunikation über Funk statt. Hierbei wird eine Client-Server Architektur gewählt, die es erlaubt, unabhängig von den Robotern unterschiedliche Bediengeräte zu implementieren. Ein weiterer Vorteil dabei ist, dass auch eine Kommunikation zwischen Server und Roboter stattfinden kann, ohne dass eine Fernbedienung verbunden ist, was für das selbstständige Anfahren der Ladestation wichtig ist. Das Finden der Ladestation funktioniert mittels einer Infrarot-LED, die impulslängenmodulierte Signale sendet.

Inhaltsverzeichnis

1 Einleitung

Die Geschichte der Roboter reicht bis in die Antike zurück. Schon dort gab es erste Versuche mit Automaten, die zum Beispiel Musik spielen sollten oder automatisch Theater spielen konnten. Mit dem Verlust der antiken Kulturen gingen jedoch auch die Kenntnisse über die Automation verloren. Erst im 13. Jahrhundert wurde ein Buch eines arabischen Ingenieurs bis in die westliche Welt bekannt, was Gerüchten zufolge auch Leonardo da Vinci für die Automation inspiriert haben soll. Roboter wie wir sie kennen gibt es erst seit Mitte des 20. Jahrhunderts. Der technische Wendepunkt, der mit der Erfindung des Transistors kam, machte es erst möglich, elektrische Schaltungen in einer Größe zu fertigen, die für Roboter notwendig ist. Seit diesem Zeitpunkt liegt ein wertvoller und nicht mehr wegzudenkender Industriezweig auf dem Gebiet der Robotik. Mit dem Einstieg der Robotik in die Industrie wurde die Produktion um ein vielfaches schneller und präziser, als sie von Menschenhand je vorgenommen werden könnte. Auch in Bereichen oder Umgebungen, die für Menschen lebensgefährlich oder ohnehin lebensunmöglich sind, spielen Roboter eine bedeutende Rolle. Ein Beispiel hierfür ist der Mars-Rover. Der Mars-Rover ist ferngesteuertes Fahrzeug, welcher für die Marsforschung verwendet wird und größten Teils von der Erde aus gesteuert wird. Wie die meisten ferngesteuerten Fahrzeuge ist er mit einer Vielzahl von Sensoren und Werkzeugen ausgestattet.

Seit Ende der 90-er Jahre gibt es einen weltweiten Wettbewerb namens RoboCup. Bei RoboCup geht es darum, ein Fussballspiel von Robotern austragen zu lassen. Diese Bachelorarbeit beschäftigt sich ebenfalls mit dem Thema der Steuerung von mobilen Robotern mit dem Ziel, ein Roboter-Fussballspiel zu ermöglichen. Im Gegensatz zum RoboCup, bei dem die Roboter entweder autonom, oder von einem zentralen Computer gesteuert werden, der mittels einer Kamera über dem Spielfeld die Bewegungen und Situationen erfasst, wird in dieser Bachelorarbeit die Steuerung der Roboter vom Menschen übernommen. 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. Mithilfe von mobilen Endgeräten 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 mobilen Endgeräten, welche die Steuercontroller bilden, 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 Akku einen Mindestprozentsatz unterschreitet, wird automatisch die Ladestation angefahren, damit sich der Akku des Roboters selbstständig, also ohne menschliches Eingreifen, lädt. Die Fertigung und Implementierung der Roboter ist Teil einer anderen Bachelorarbeit, auf die hier nur Oberflächlich eingegangen wird.

1.1 Anforderungen

Das Ziel der Bachelorarbeit ist es, ein Fussballspiel für zwei Spieler zu realisieren. Hierfür werden zwei Roboter auf einem begrenzten Spielfeld positioniert. Jeweils am Rand des Spielfelds wird ein Tor montiert, in das der Ball befördert werden soll. Mittels mobilen Endgeräten und herkömmlichen PCs soll es möglich sein, die Steuerung über einen der Roboter zu übernehmen. Sobald beide Roboter mit einem Steuergerät verbunden sind, soll ein Spiel gestartet werden und die Torerkennung funktionieren. Das bedeutet, sobald ein Spieler ein Tor erzielt hat, erkennt das System dieses und erhöht den Spielstand dementsprechend. Mittels einer Kamera, die auf den Robotern montiert ist, soll eine Videoübertragung aus Sicht der Roboter stattfinden. Dadurch soll es auch möglich sein, die Roboter zu steuern, ohne sich in Sichtkontakt mit den Robotern zu befinden. Da die Roboter über einen Akku verfügen, soll die Kommunikation kabellos erfolgen, damit ein freies Fahren der Roboter sichergestellt ist. Weil der Akku eine begrenzte Kapazität hat, soll der Roboter bei niedrigem Akkustand selbstständig die Ladevorrichtung anfahren und während der Ladezeit keine Steuerbefehle annehmen. Aufgrund der Interaktion mit menschlichen Benutzern, ist es wichtig, dass der Roboter eine möglichst rechtzeitige Steuerung darstellt, um Verzögerungen und die dadurch entstehende Unkontrollierbarkeit zu vermeiden.

1.2 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 mussten Protokolle für beide Übertragungsrichtungen definiert werden. Im Folgenden sind die Befehle 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.

Abbildung 1.1: Befehle, die der Roboter annimmt

Hier wird ersichtlich, dass selbst für geradeaus fahren keine Funktion oder derartiges bereit steht, lediglich die beiden Motoren können unabhängig voneinander angesteuert werden. Dadurch muss jegliche Logik, die das Steuern des Roboters ermöglicht, in dieser Arbeit behandelt werden.

In der anderen Richtung, also Befehle 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

Abbildung 1.2: Befehle, die der Roboter versendet

Außerdem hat man sich auf eine Befehlsfrequenz von 10Hz geeinigt. Bei dieser Frequenz wird der Roboter nicht überfordert und der Benutzer hat dennoch mit keinen Verzögerungserscheinungen zu kämpfen. auch die Statusnachrichten kommen in diesem Intervall, es sei denn, es werden noch Bilddaten angehängt, dann sendet der Roboter ohne Unterbrechungen.

2 Grundlagen

2.1 UDP

Beim User Datagram Protocol 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. Hierzu zählen:

Sicherheit Die Sicherheit, dass Pakete überhaupt ankommen

Reihenfolge Die Reihenfolge, in der Pakete ankommen

FlowControl Ein Überlaufschutz des Empfängerspeichers (FlowControl)

CongestionControl Stauvermeidung in der Übertragungskette (CongestionControl)

Angriffe Schutz gegen Paketmanipulation von dritten

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.

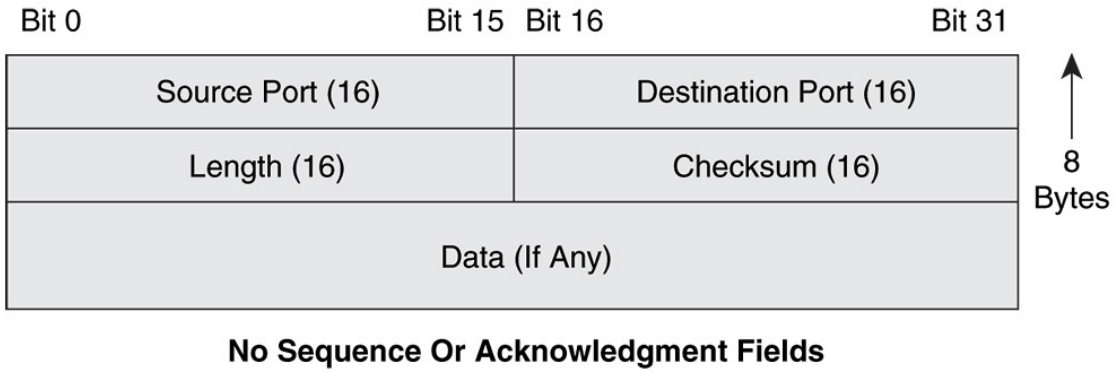


Abbildung 2.1: UDP Header

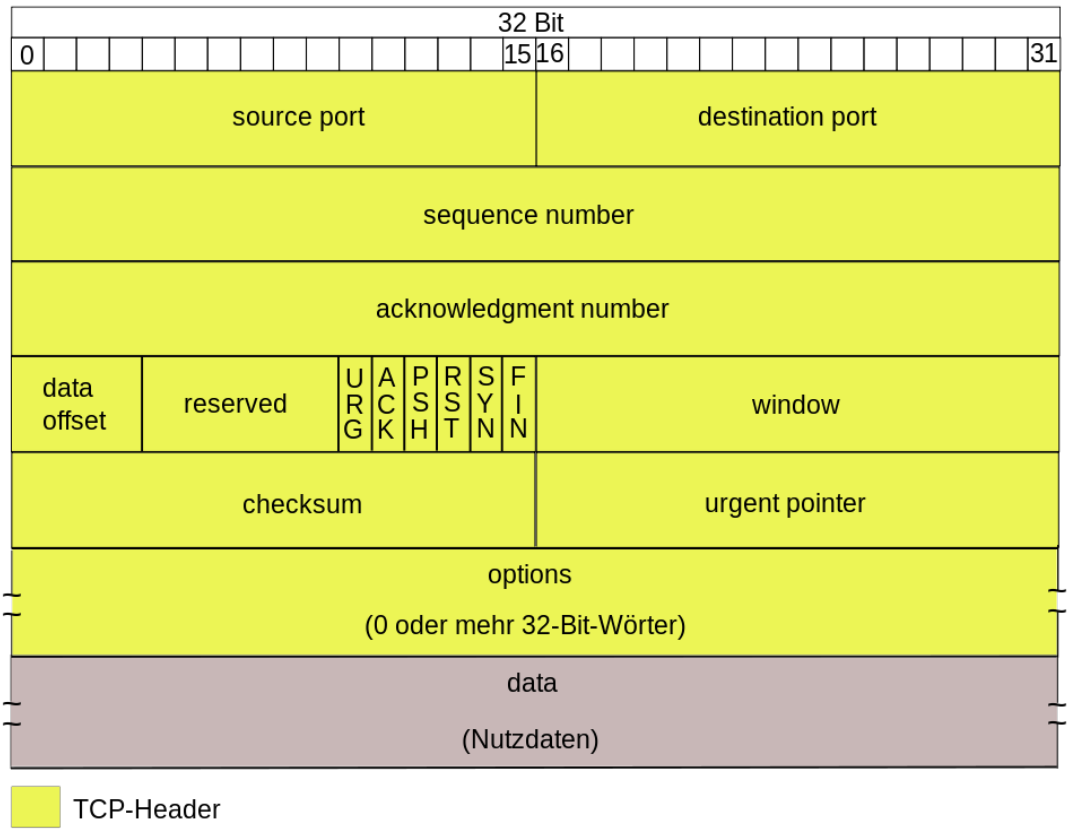


Abbildung 2.2: TCP Header

Aufgrund dieser Eigenschaften wird UDP in Bereichen eingesetzt, in denen es auf hohe Übertragungsgeschwindigkeit ankommt und eventuelle Paketverluste zu verkraften sind, bzw. von höheren Schichten aufgelöst werden.

2.2 TCP

Beim Transmission Control Protocol 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 dem 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

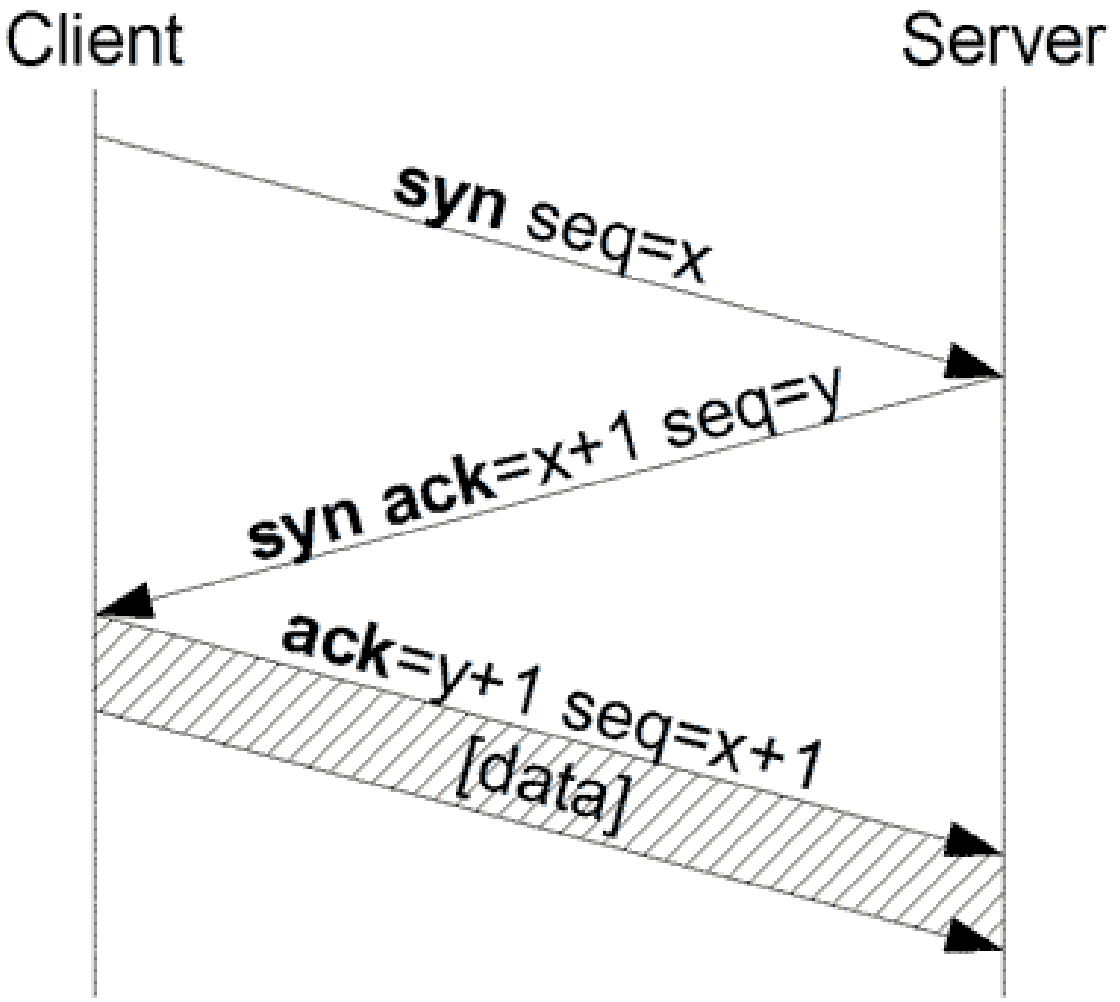


Abbildung 2.3: TCP 3-Wege-Handshake

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.

2.3 HTTP

Das Hypertext Transfer Protocol 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.
Die möglichen Befehle sind:

GET fordert eine Ressource auf dem Server an

POST sendet Daten zur weiteren Verarbeitung zum Server

HEAD fordert lediglich den HEADER eines Responses an, der auf einen GET-Request folgend würde

PUT lädt eine Ressource auf den Server

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

TRACE sendet die Anfrage, so wie sie empfangen wurde zurück

OPTIONS liefert eine Liste mit den vom Server unterstützten Optionen

CONNECT wird für SSL-Tunneling 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. In Abbildung 2.4 ist eine solche Kommunikation dargestellt.
–

2.4 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

Abbildung 2.4: HTTP Kommunikation

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.

2.5 Jetty Webserver

Jetty 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.

2.6 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 %

3 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 dann 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. Der Status des Roboters soll 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.

Ausgehend von diesen Bedingungen, die die Architektur erfüllen muss, kamen folgende Architekturen in Frage:

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. 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.

Nach Abwägung der Vor- und Nachteile beider Architekturen schien es vernünftiger sich für die Client-Server-Architektur zu entscheiden. Einer der Hauptgründe hierfür ist die hervorragende Erweiterbarkeit in Bezug auf die Controller. Da keine Logik in den Controllern ist, muss sie auch nicht für jeden Controller separat entwickelt werden, beziehungsweise nicht einmal vorhanden sein. Gerade 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

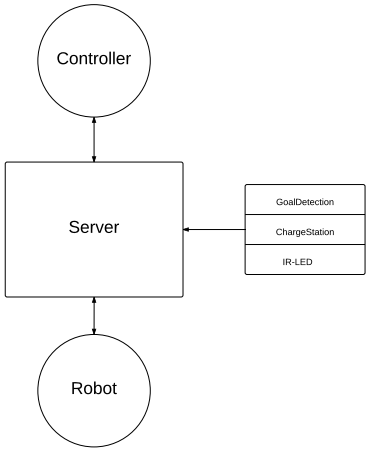


Abbildung 3.1: Client-Server Architektur

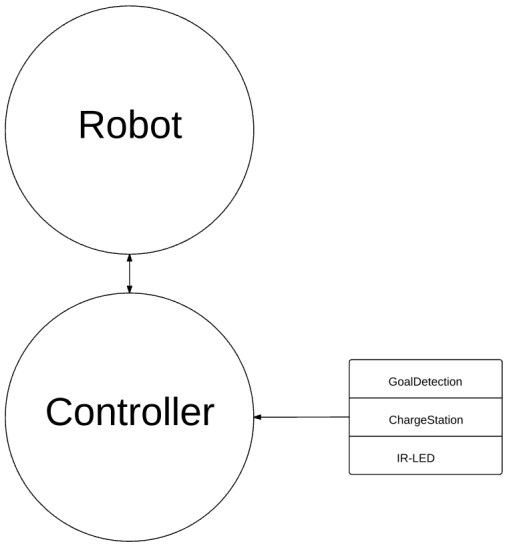


Abbildung 3.2: Paarweise Architektur

Informationskapselung, die bei der paarweisen Architektur entstehen würde, ist es sinnvoller die Client-Server-Architektur zu bevorzugen. Nur wenn alle Informationen über das Spiel, also über jeden Roboter und jeden Controller zentral verwaltet werden, ist es möglich die Informationen korrekt zu verarbeiten und entsprechend darauf zu reagieren. So wird zum Beispiel ein neues Spiel gestartet, sobald beide Roboter mit einem Controller verbunden sind. Bei der paarweisen Architektur wäre dies nicht möglich, da die beiden Controller-Roboter-Einheiten vom jeweils anderen keine Informationen über deren Status erhalten. In Anbetracht der Rechtzeitigen Steuerung bietet die gewählte Architektur einen Nachteil gegenüber der paarweisen. 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 verzögerungslos erscheinende Steuerung zu gewährleisten.

4 Kommunikation

4.1 Übertragungstechnologie

Da die Roboter, sowie auch die Controller über Funk mit dem Server kommunizieren sollten, gab es an dieser Stelle einige Entscheidungen.
Generell musste hierbei zwischen der Roboter-Server und der Controller-Server Kommunikation unterschieden werden. Für beide Wege 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 5.1.1).

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. Auch in Punkto Energieeffizienz wäre mit Bluetooth Low Energy eine gute Lösung gefunden, da es im Vergleich zu WLAN beim Senden nur in etwa ein sechstel des Stroms verbraucht .
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 ein vielfaches schneller (stark von Version abhängig).

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) 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 aber 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. Im Gegensatz zu TCP profitiert UDP allerdings von seiner Eigenschaft wenig Traffic zu verursachen. Da bei TCP jedes Paket bestätigt wird muss der Roboter jedes Paket mehrmals verarbeiten. Die Hardware des Roboters ist jedoch eher schwach bestückt, weshalb der hohe Traffic letzendlich zu längeren Verzögerungen führt als bei UDP. 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 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 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.

5.1 Das Tor

Das Tor vereint in sich drei Funktionalitäten. Die Infrarot-LED, die Torerkennung und die Ladestation. Es wurde unter Verwendung von 3D-Druck erstellt und konnte dadurch genau an die Anforderungen angepasst entworfen werden.

5.1.1 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 verschieden lang an und ausschalten.

Der Aufbau ist wie in Abbildung 5.2 zu sehen folgendermaßen: Am oberen Endes Tores sind drei Infrarot-LEDs befestigt. Die äußeren beiden haben einen Abstrahlwinkel von 40°und die 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

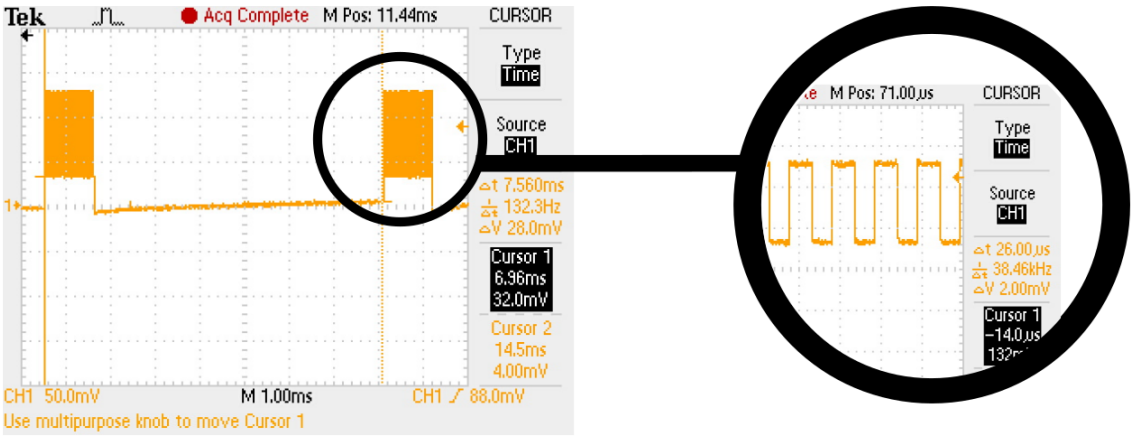


Abbildung 5.1: Beispiel Burst mit einer postivien Pulslänge von ca. 1ms und ca. 6,4ms Pause

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 5.2.4)

5.1.2 Ladestation

Die Ladestation befindet sich am oberen Rand des Tors. Der Roboter verfügt auf der Rückseite (also auf der Seite, auf der auch der Infrarot-Empfänger platziert ist) über drei Ladekontakte. Auf gleicher Höhe sind am Tor Kontakte angebracht, die das Ladegerät darstellen. Auf diese Weise ist es möglich den Roboter zu laden, indem man ihn einfach gegen die Ladestation fährt.

5.1.3 Torerkennung

Grundlage der Torerkennung ist die Überlegung, mit welcher Technologie ein Tor erkannt werden soll. Zur Auswahl standen hier mehrere Möglichkeiten.
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 dadurch eine Komponente darstellen würde und deshalb nicht leicht austauschbar wäre. Man müsste ihn so konzipieren, dass er durch die auf ihn einwirkenden Kräfte nicht zerstört würde, da sobald er

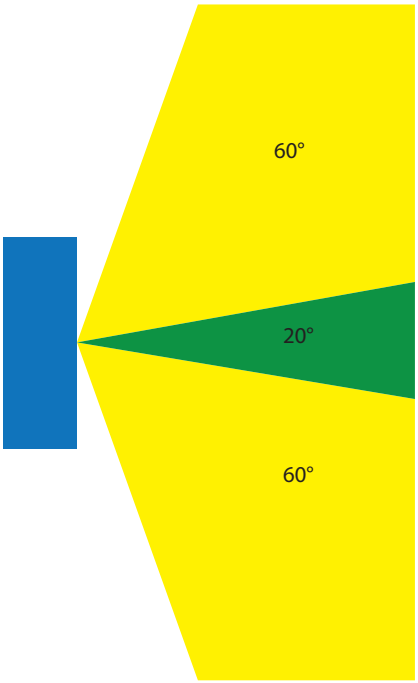


Abbildung 5.2: Schematischer Aufbau der Infrarot LED mit Abstrahlwinkel

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 hängt eine Klappe, die vom Ball kurz nach hinten gedrückt wird. Dort schließt sie zwei Kontakte, die einen Interrupt beim RaspberryPi auslösen.

5.2 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.

Als Hostgerät für den Server wurde ein RaspberryPi ausgewählt. Dieser bietet volle Unterstützung für Java-Programme und bietet mit seinen GPIO-Pins eine optimale Basis für die Ansteuerung und Überwachung der Peripheriegeräte. Als Programmiersprache wurde Java gewählt.

5.2.1 Verbindung

Die Verbindungslogik besteht im Wesentlichen aus fünf Klassen, die in Abbildung 5.3 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. 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. Auch die Authentifizierung wird in dieser Klasse abgehandelt. 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.

5.2.2 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 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,

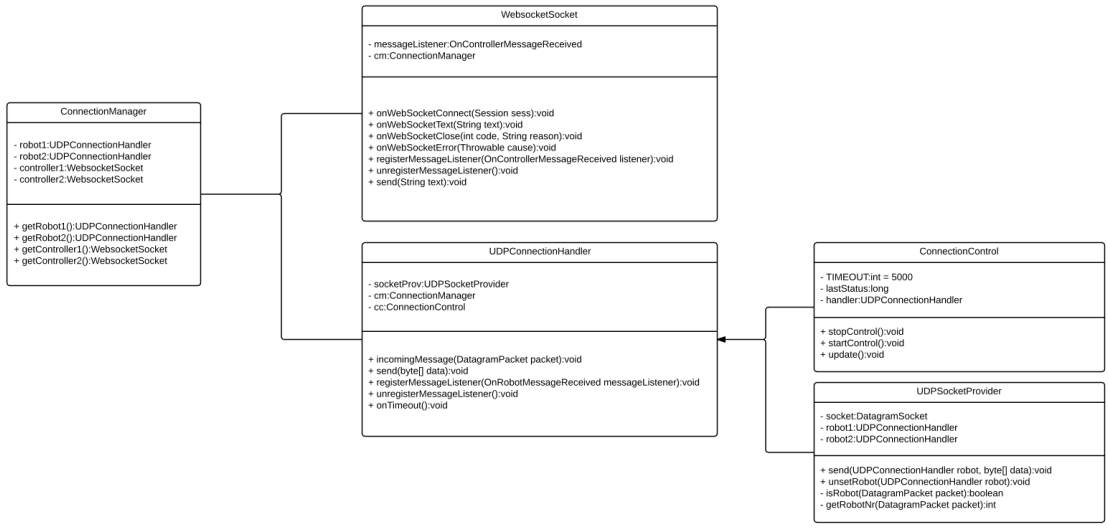


Abbildung 5.3: Klassendiagramm der Verbindungslogik

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 folgt nun ein Beispiel:

Letzte Motorbefehle:
links: 50
rechts: 50

Neue Motorbefehle:
links: 80
rechts: 70

Neue Motorbefehle nach Korrektur (ohne Verhältnis nachzukorrigieren)
links: 60
rechts: 60

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.

Nach Anpassung des Verhältnisses – $\frac{links}{rechts} = \frac{8}{7}$
links: 60
rechts: $\frac{links}{\frac{8}{7}} = 52$

Differential Steuerung

Da die eingehenden Steuerparameter direkt die Motorenbefehle für Links und Rechts beinhalten (vgl. Abschnitt 5.4.1), ist hier keine weitere Interpretation notwendig.

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.

Webanwendung

Diese Steuervariante hat einen bedeutenden Nachteil. Da die Eingangsparameter die momentan gedrückten Tasten sind (siehe Abschnitt 5.3), 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 Älles oder NichtsSSteuerung führt im Spiel zu einer nicht flüssigen, jedoch durch die Möglichkeit der Drehung auf der Stelle zu einer annehmbaren Steuerung.

5.2.3 Bildübertragung

Wie in der Schnittstellenbeschreibung in Kapitel 1.2 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 dann 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 an 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.

5.2.4 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. 5.4). 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 die Minimalschwelle unterschreitet, werden die Infrarot-LEDs über die Klasse BlinkTask angeschaltet. Über das ßeeGoal"Flag (vgl. Schnittstellenbeschreibung, Kapitel 1.2) 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. 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 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.

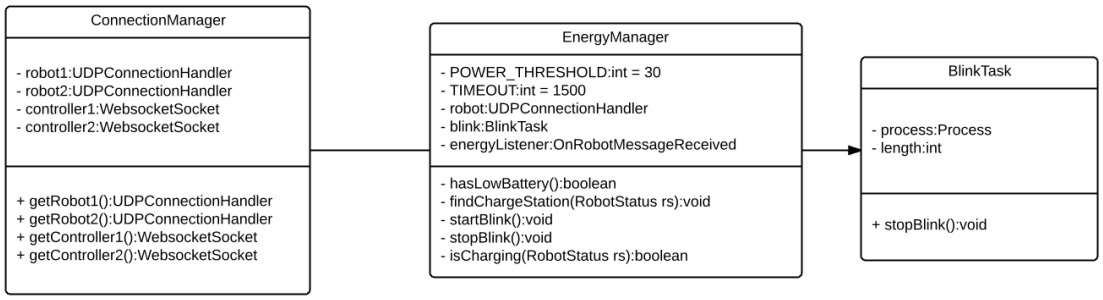


Abbildung 5.4: Klassendiagramm der Torfindung

5.3 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. 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 5.2.2).

5.4 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 7.2 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 Websocket an den Server sendet. Die Klassen Websocket, CommandManager und ControlActivity

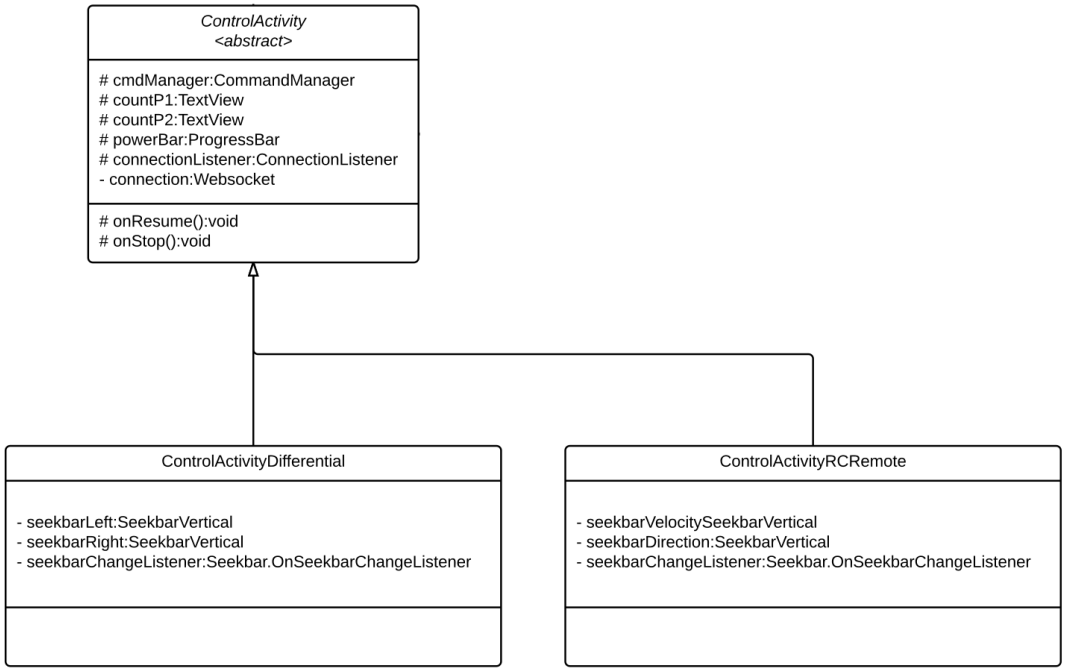


Abbildung 5.5: Klassendiagramm ControlActivities

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.

5.4.1 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. 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.

5.4.2 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

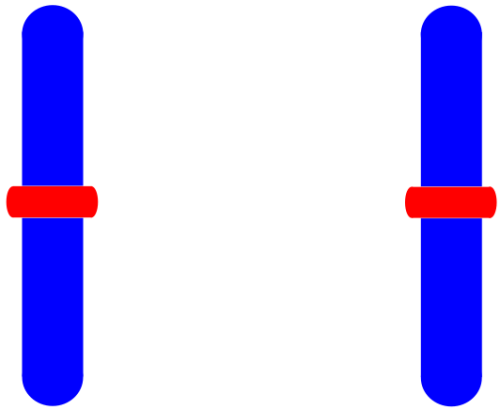


Abbildung 5.6: Schematische Darstellung der Differential Steuerung

Spielzeugautos kennt. Auch hier wird der Schuss wieder über die Sensoren ausgelöst. 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 einfach 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.

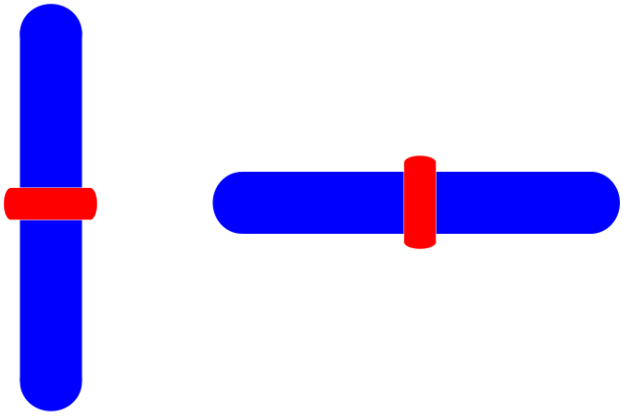
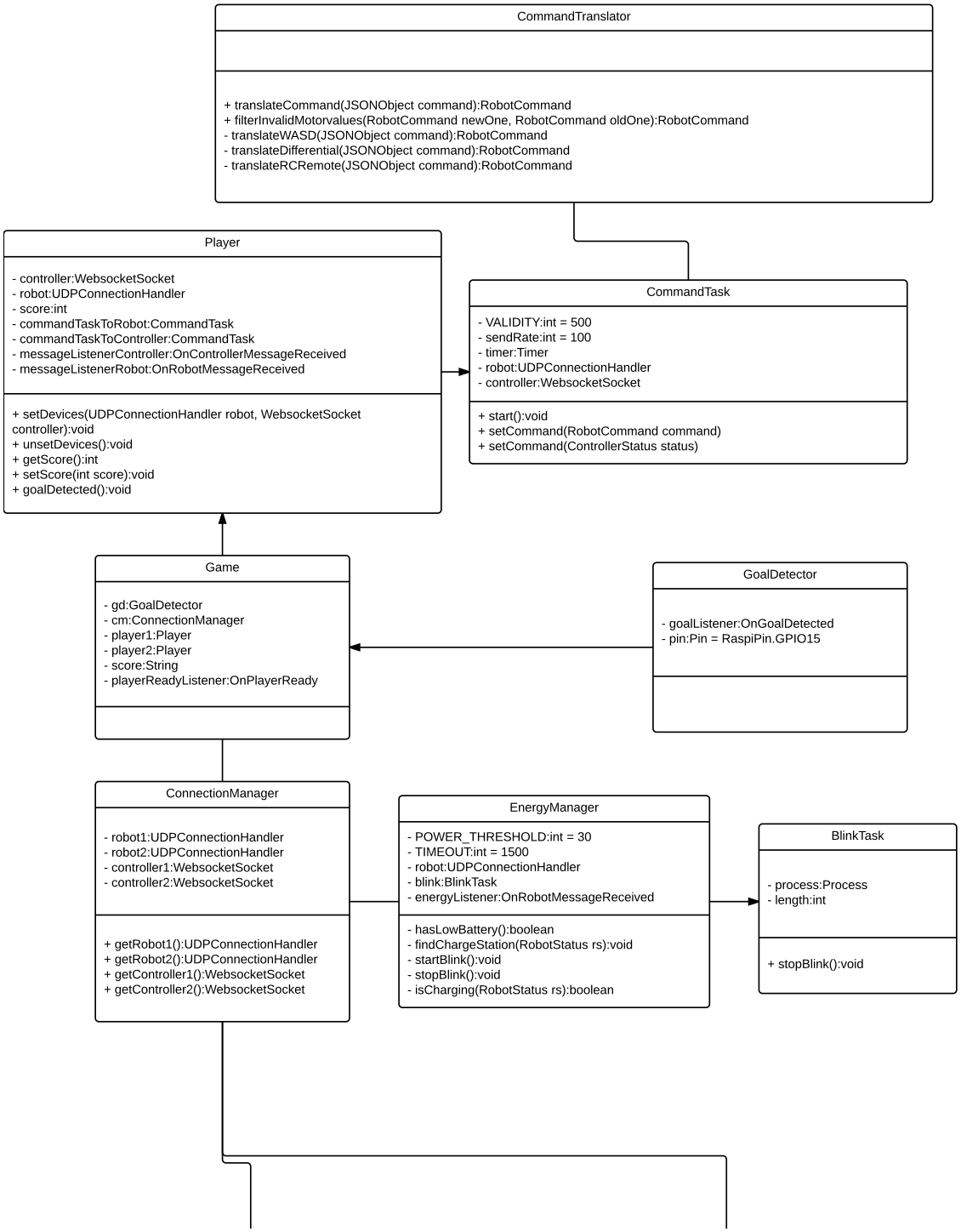


Abbildung 5.7: Schematische Darstellung der RCRemote Steuerung

6 Fazit und Ausblick

7 Anhang


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

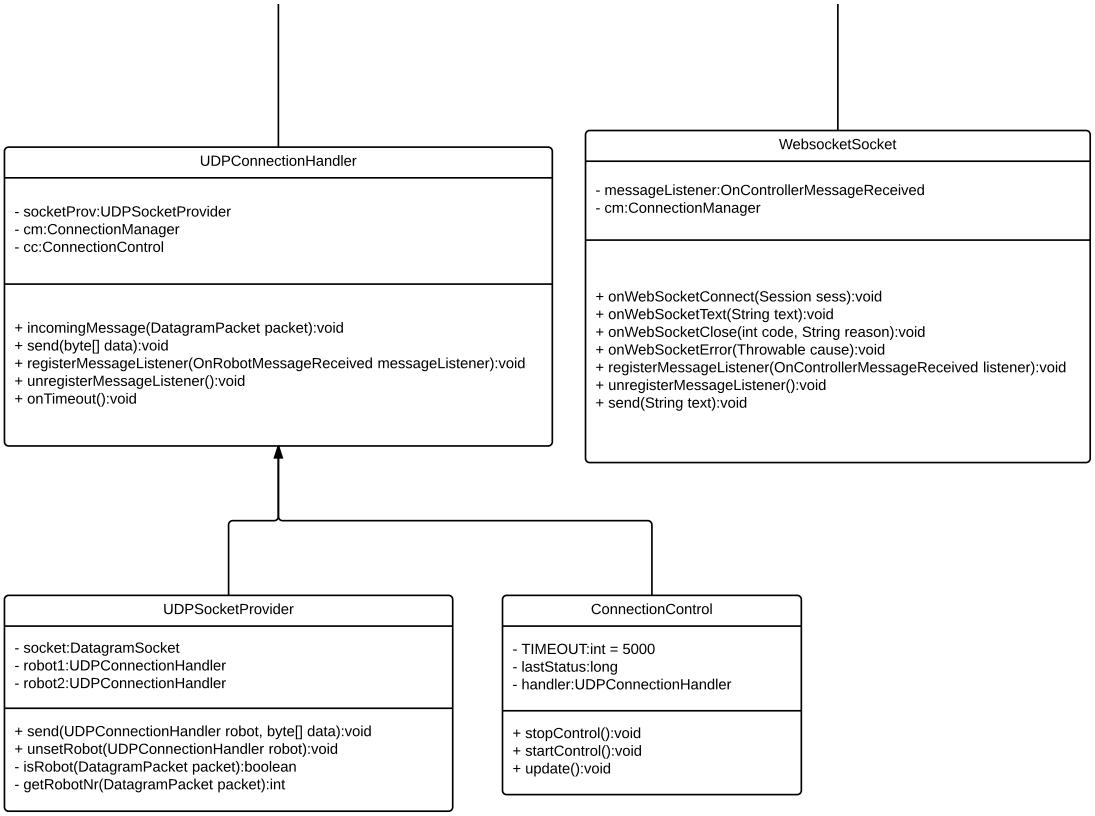


Abbildung 7.1: Klassendiagramm des Servers

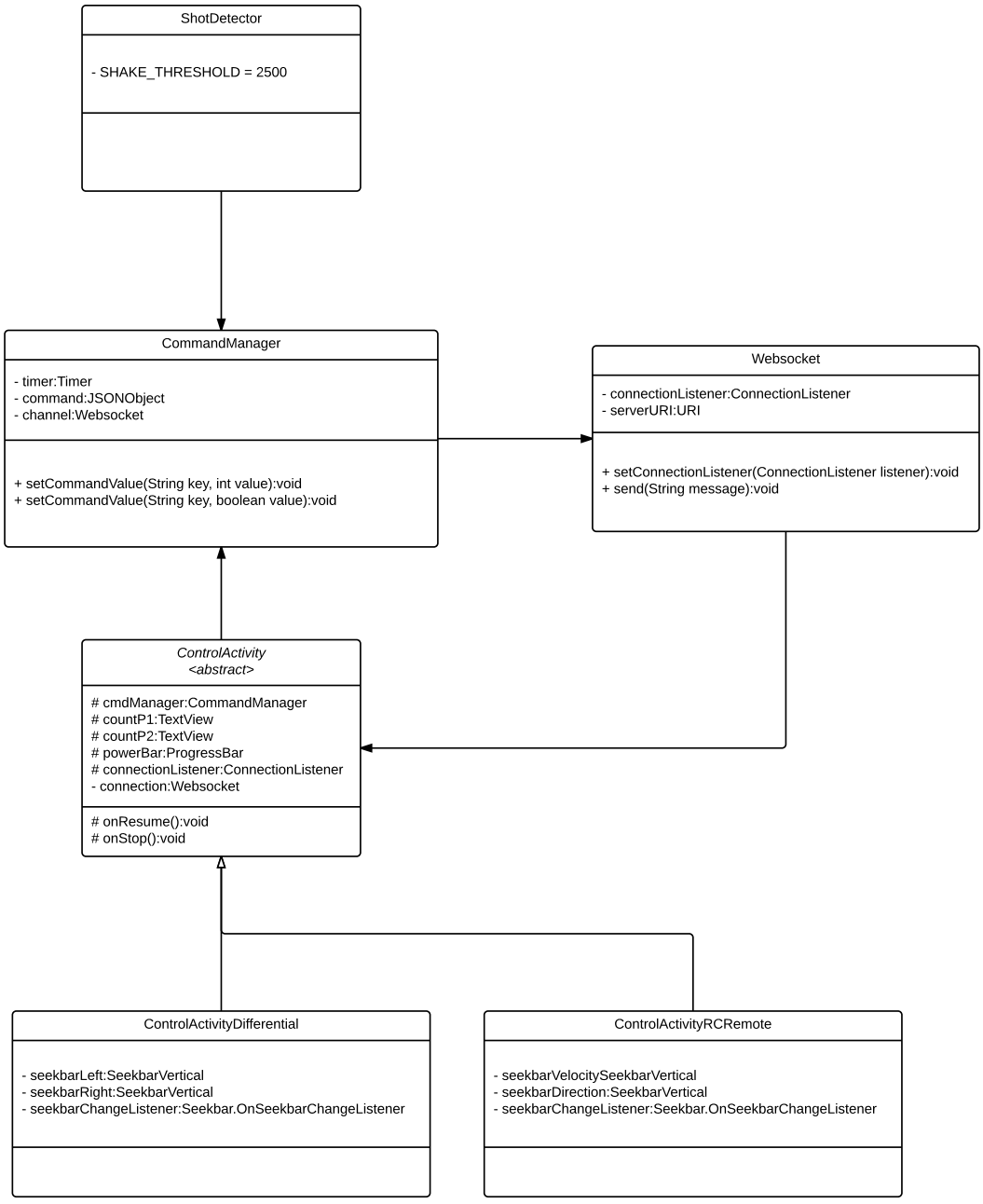


Abbildung 7.2: Klassendiagramm der Android Anwendung

8 Literaturverzeichnis

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