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 30. Oktober 2015
© 2014 Patrick Lutz
This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.0/de/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.
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.

Diese Bachelorarbeit beschäftigt sich mit dem Thema der Steuerung von mobilen Robotern mit dem Ziel, ein Roboter-Fussballspiel zu ermöglichen. 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.

2 Grundlagen

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.
Aufgrund dieser Eigenschaften wird UDP in Bereichen eingesetzt, in denen es auf hohe Über-

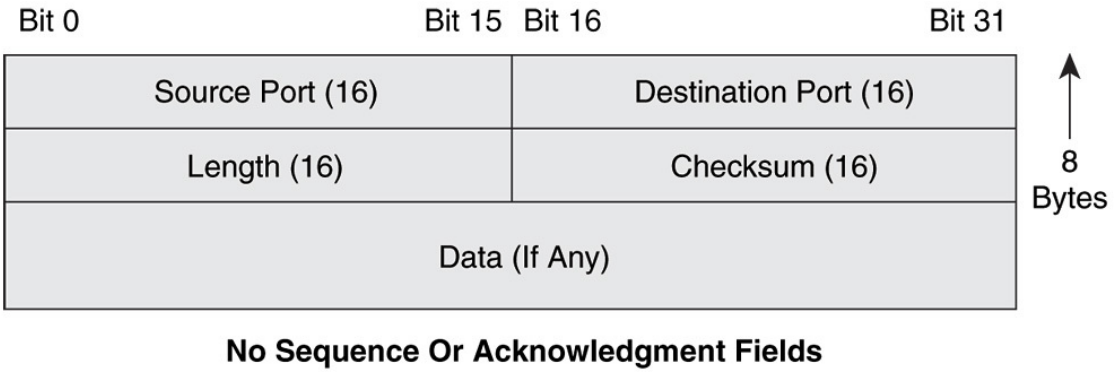

Abbildung 2.1: UDP Header

tragungsgeschwindigkeit ankommt und eventuelle Paketverluste zu verkraften sind, bzw. von höheren Schichten aufgelöst werden.

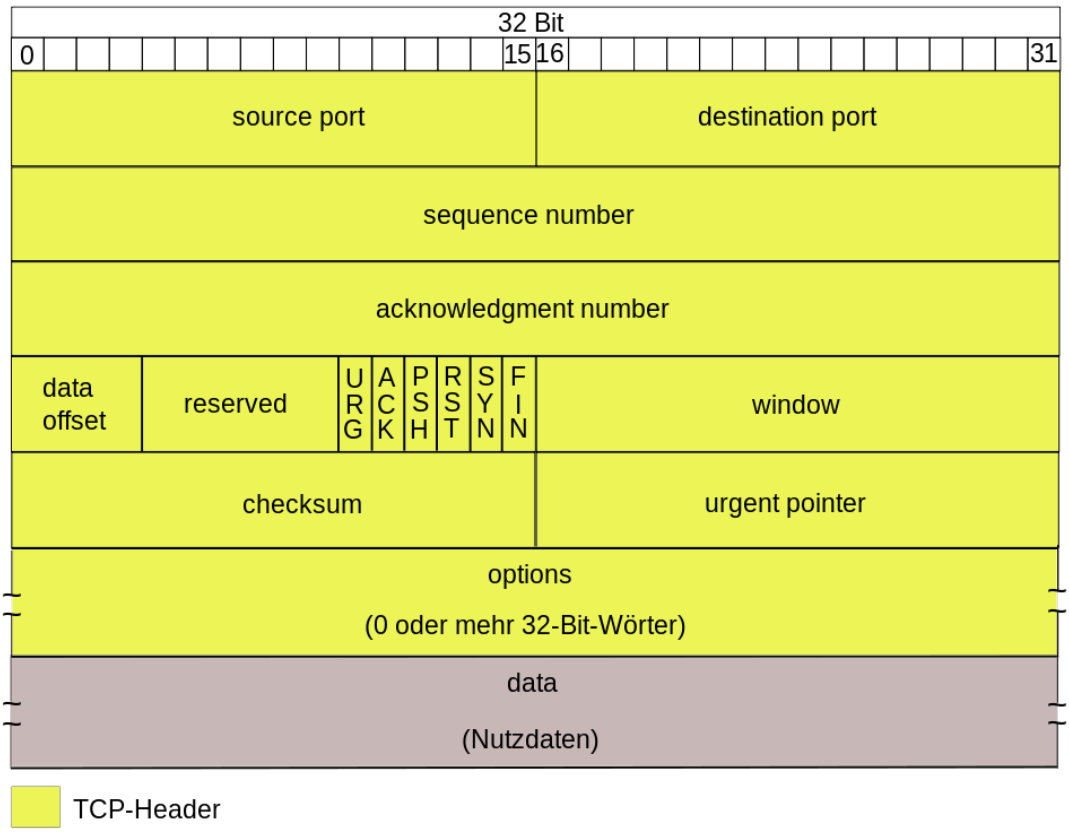

Abbildung 2.2: TCP Header

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 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

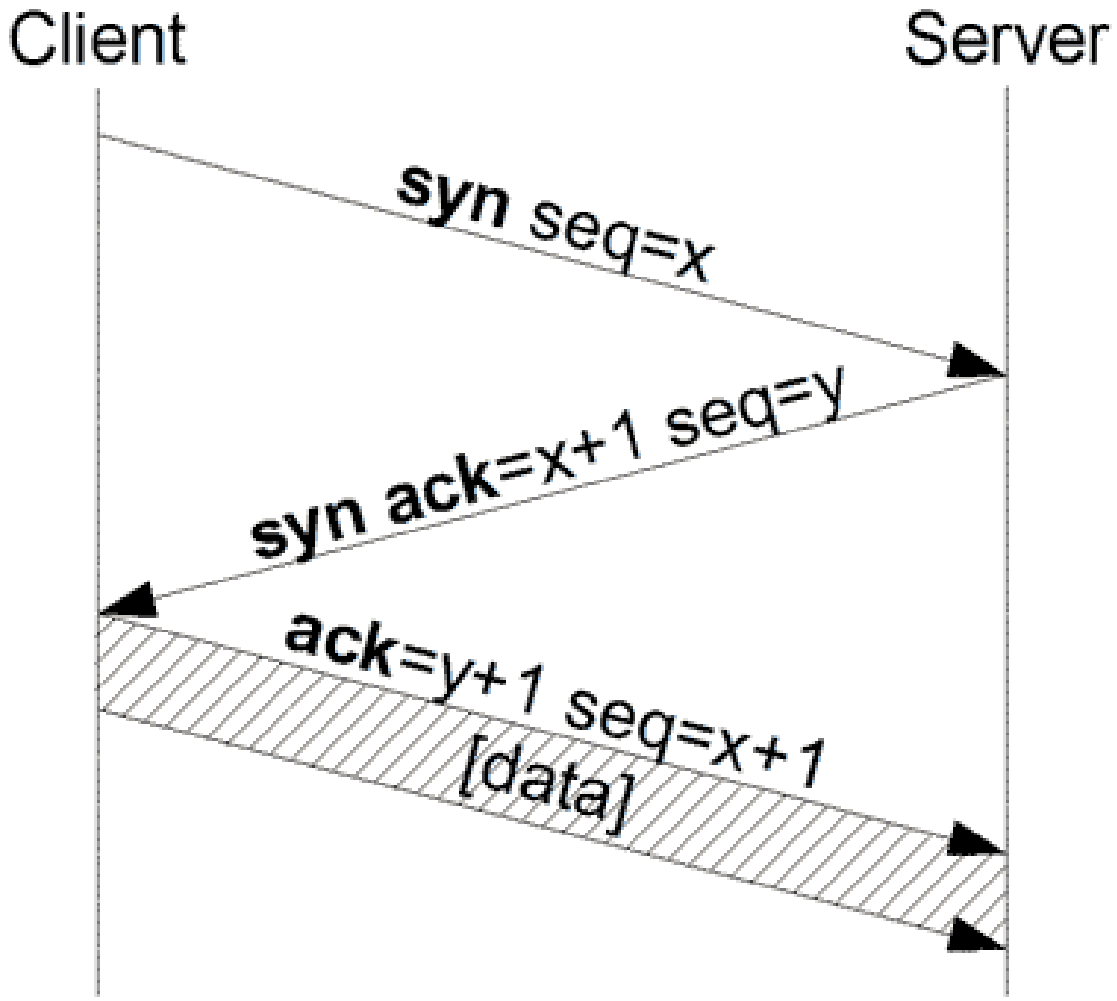

Abbildung 2.3: TCP 3-Wege-Handshake

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.

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.
–

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

Abbildung 2.4: HTTP Kommunikation

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.

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.

3 Hauptteil

3.1 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.

3.2 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 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 zwangs-

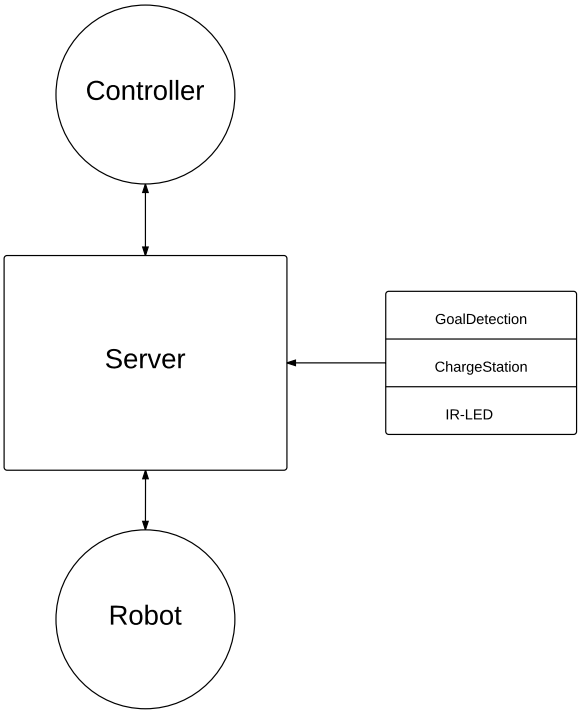

Abbildung 3.1: Client-Server Architektur

weise 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.

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 Informationskapselung, die bei der paarweisen Architektur entstehen würde, ist es sinnvoller die

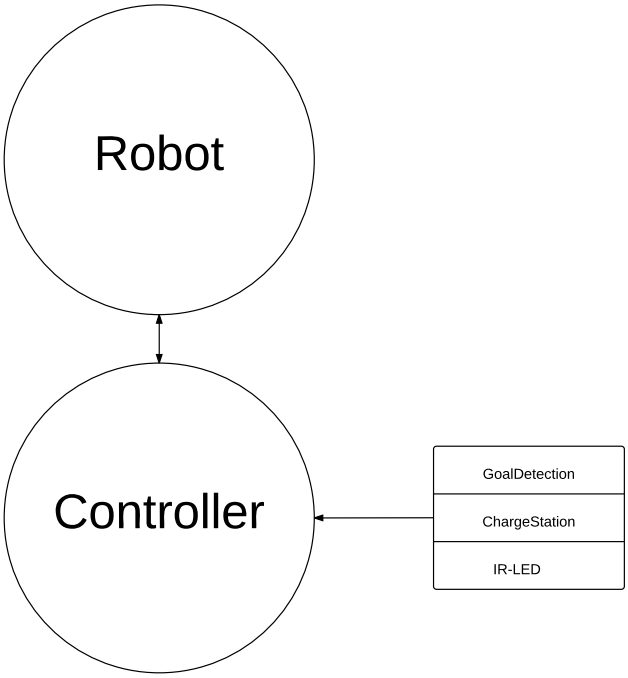

Abbildung 3.2: Paarweise Architektur

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.

3.3 Kommunikation

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).

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.

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.

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