Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jonas Hornschuh

Weiterentwicklung eines Fahrradergometers als intuitive Steuerung für virtuelle Welten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jonas Hornschuh

Weiterentwicklung eines Fahrradergometers als intuitive Steuerung für virtuelle Welten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 3. März 2015

Jonas Hornschuh

Thema der Arbeit

Weiterentwicklung eines Fahrradergometers als intuitive Steuerung für virtuelle Welten

Stichworte

Natural User Interface, Ubiquitous Computing, Human Computer Interaction, Ergometer, EmoBike, Fahrradsimulation, Latenz, Präzision, Raspberry Pi

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Erweiterung eines Fahrradergometers um eine natürliche Benutzerschnittstelle und die Integration des Ergometers in eine Fahrradsimulation. Die Simulation wird im Kontext des EmoBike Projekts entwickelt. Zunächst wird analysiert was die natürliche Interaktion mit einem Fahrrad ausmacht und wie diese auf das Ergometer übertragen werden kann. Als Ergebnis der Analyse wird der starre Lenker gegen ein drehbares Modell ausgetauscht, dessen Bewegungen von einem Computersystem erfasst und für die Wiedergabe in einer virtuellen Welt aufbereitet werden. Zusätzlich wird ein Softwareadapter entwickelt, der das Ergometer in die Simulation integriert und so die Fernsteuerung des Tretwiderstands in Abhängigkeit der virtuellen Umgebung ermöglicht. Im Anschluss wird evaluiert inwieweit die Benutzerschnittstelle gegenüber dem Ausgangszustand verbessert werden konnte und worauf beim Entwickeln von natürlichen Interfaces geachtet werden muss.

Jonas Hornschuh

Title of the paper

Further development of a bicycle ergometer as an intuitive controller for virtual worlds

Keywords

Natural User Interface, Ubiquitous Computing, Human Computer Interaction, ergometer, EmoBike, bycicle simulation, latency, precision, Raspberry Pi

Abstract

This work deals with the extension of a bicycle ergometer with a natural user interface and the integration of the ergometer into a bycicle simulation. The simulation is being developed in the context of the EmoBike project. The analysis shows what constitutes the natural interaction with a bicycle and how this can be transferred to the ergometer. As a result of the analysis, the rigid handle bar is replaced by a rotatable model whose movements are received by a computer system and processed for playback in a virtual world. In addition, a software adapter is developed, which integrates the ergometer into the simulation, allowing remote control of the pedal resistance as a function of the virtual environment. Afterwards an evaluation is made to determine how much the user interface has improved compared to the initial state, and what to consider during the development of natural interfaces.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Aufgrund des technischen Fortschritts und den damit verknüpften Möglichkeiten halten Computer zunehmend Einzug in unser direktes Umfeld. Smartphones, Smart-TVs, Wearables, selbststeuernde Fahrzeuge, aber auch Haushaltsgeräte sind Beispiele für die allgegenwärtige Präsenz von Computern. Mark Weiser hat den Begriff Ubiquitous Computing [vgl. Weiser (1991)] geprägt, der diese Rechnerallgegenwart beschreibt. Die Assoziation eines Computerinterfaces bestehend aus Maus, Tastatur und Bildschirm trifft auf viele Geräte nicht mehr zu. Stattdessen interagieren wir mit Computern teilweise so natürlich, dass uns deren Anwesenheit nicht mehr bewusst ist (Disappeared Computing). Sei es die Sprachsteuerung des Smartphones, z.B. mit Google Now: „Zeige Restaurants in der Nähe“, oder die Gestensteuerung eines Konsolenspiels, die es ermöglicht ohne Gamecontroller zum Beispiel gegen einen virtuellen Tischtennisgegner anzutreten. Zurückzuführen ist dies auf die Veränderung der Schnittstellen zu Computersystemen, die sich mit Sprache und Gestik immer stärker der zwischenmenschlichen Kommunikation annähern. Diese als „natürlich“ bezeichneten Benutzerschnittstellen, „Natural User Interfaces“, wobei „natürlich“ hier im Sinne von „intuitiv“ oder „vertraut“ zu verstehen ist [vgl. Bærentsen (2001), zit. in: Bernin (2011)], sind ein Grund für die zunehmende Integration von Computern in unseren Alltag. Gleichwohl gibt es noch viele Bereiche in denen die Interaktion zwischen Mensch und Computer (Human Computer Interaction [vgl. Hewett u. a. (1992)]) keine Natürlichkeit aufweist und dessen Verbesserung die Motivation des Autors ist.

1.2. Zielsetzung

Diese Bachelorarbeit entsteht im Rahmen des EmoBike Projekts, dessen wissenschaftlicher Schwerpunkt auf der Erkennung und Erforschung menschlicher Emotionen [vgl. Picard (1997)] im Bereich der Mensch-Maschine-Interaktion liegt. Die Basis bildet eine zu entwickelnde Fahrradsimulation bestehend aus einem Fahrradergometer als Eingabe-Controller und einer virtuellen Computerwelt zur Visualisierung einer interaktiven Umgebung. Diese Arbeit geht

der Frage nach, wie sich das Benutzerinterface eines Ergometers erweitern lässt, um eine natürliche Steuerung für ein realistisches Fahrradfahrerlebnis zu schaffen. Der Fokus liegt zum einen auf der Implementierung eines natürlichen Lenksystems, dessen Bewegungen erfasst und in die virtuelle Welt übertragen werden müssen. Zum anderen liegt er auf der Realisierung eines Systems, das Veränderungen der virtuellen Umgebung empfängt und daraufhin ein Feedback in Form eines wechselnden Tretwiderstandes gibt. Als Grundlage für den Informationsaustausch zwischen dem realen Objekt und der virtuellen Welt muss eine Infrastruktur entwickelt und aufgebaut werden, die sowohl die Kommunikation der geplanten, aber auch die Erweiterbarkeit um zusätzliche Systeme sicherstellt.

1.3. Gliederung

Diese Arbeit gliedert sich in fünf Kapitel, die hier kurz zusammengefasst werden. Das Kapitel 2 stellt das EmoBike Projekt vor, in dessen Kontext diese Arbeit entstanden ist, und ordnet die Funktion des Fahrradergometers in das Gesamtprojekt ein. Die Analyse in Kapitel 3 geht zunächst auf das Interface des Ergometers ein und legt dar welche Möglichkeiten bestehen um eine intuitive Steuerung aufzubauen, die natürliche Interaktionen erlaubt. Davon leiten sich die Zielsetzung sowie die zu erreichenden Teilziele ab. Die nachfolgende Anforderungsanalyse beschreibt die einzelnen Zielvorgaben und welche Bereiche besonders berücksichtigt werden müssen. Anschließend wird in Kapitel 4 die Architektur vorgestellt und klar abgegrenzt welche Teilbereiche im Rahmen dieser Arbeit entwickelt werden. Zudem wird hier die Grundlage für das Kommunikatonssystem entworfen, das die späteren Komponenten miteinander verbindet. Es folgt das Kapitel 5 mit der Realisierung der in der Analyse herausgearbeiteten Ziele. Es wird jede Komponente für sich beschrieben und aufgezeigt welche Hindernisse bei der Implementierung auftraten und wie diese überwunden werden konnten. Die Evaluation fasst das Ergebnis der Realisierung zusammen und gibt eine Perspektive für zukünftige Erweiterungen. Kapitel 6 zieht das Fazit der gesamten Arbeit und gibt einen Ausblick auf die Möglichkeiten zur Entwicklung von Natural User Interfaces.

2. Das EmoBike Projekt

2.1. Ziele des Projekts

Das EmoBike Projekt ist im Forschungsbereich der Human Computer Interaction (HCI) angesiedelt und beschäftigt sich mit der Fragestellung, wie sowohl die Interaktion zwischen Mensch und Maschine (Mensch-Maschine-Interaktion) als auch die Reaktion der Maschine auf den Zustand des Menschen verbessert werden kann. Neben dem physischen soll vor allem der emotionale Zustand erkannt und analysiert werden. Die Entwicklung eines Referenzsystems wird im Kontext eines Fahrradergometers durchgeführt, wodurch sich gute Möglichkeiten für Messungen, Interaktionen und Feedback in Form von Aktoren[1] bieten. Die Messtechniken umfassen zum Zeitpunkt der Erstellung dieser Arbeit Sensoren für Puls, Änderung der Körpertemperatur, EDA (elektrodermale Aktivität) und EEG (Elektroenzephalografie) zum Ermitteln der körperlichen Verfassung sowie kamerabasierte Systeme zum Aufzeichnen von Mimik und Emotionen. Die Erweiterung des Ergometers hin zu einem Natural User Interface (NUI) soll die Untersuchungen verbessern indem ein Prototyp geschaffen wird, der die Interaktion mit einem realen Fahrrad abbildet. Es geht dabei um zusätzliche mechanische Elemente, wie Bremse, Gangschaltung und Lenkung, die sensorisch erfasst werden und dem Nutzer ein geeignetes Feedback geben. Die Entwicklung eines solchen Natural User Interfaces ist das zentrale Thema dieser Arbeit.

2.2. Das EmoBike Szenario

Der Forschungsschwerpunkt des EmoBike Projekts liegt auf der Untersuchung von Emotionen und deren Beeinflussung. Um den Proband von diesem Umstand abzulenken wurde ein Szenario entworfen, dass zunächst nichts mit Emotionen zu tun hat. Dadurch wird verhindert, dass der Proband sein Verhalten bewusst oder unterbewusst anpasst und dadurch gegebenenfalls die Ergebnisse verfälscht. Mit dem EmoBike Szenario wird dies erreicht indem dem Probanden eine vertraute Situation simuliert wird: Das Fahrradfahren.

[1] Antriebselemente, Gegenstück zu Sensoren

Die Entwicklung des Referenzsystems konzentriert sich auf die Realisierung des EmoBike Szenarios als Fahrradsimulation (Abbildung 2.1). In dem Szenario sitzt der Proband auf dem Ergometer und interagiert damit wie mit einem gewöhnlichen Fahrrad: er tritt in die Pedalen, bewegt den Lenker und beeinflusst die Geschwindigkeit mit Bremse und Gangschaltung. Vor dem Probanden steht ein Bildschirm mit einer virtuellen Welt, dem sogenannten EmoBike Game, die sich dem Probanden in der Egoperspektive[2] präsentiert. Die Interaktionen mit dem Ergometer sind mit der virtuellen Welt verknüpft und steuern die digitale Version des Fahrrads im Spiel. Sie werden in passende Bewegungen umgesetzt, so dass der Proband das Gefühl bekommt mit dem Ergometer durch die virtuelle Welt zu fahren. Durch ausgewählte Umgebungen wird Einfluss auf das Verhalten von Ergometer und Proband genommen. Während eine Steigung einen größeren Tretwiderstand bewirkt, kann eine scharfe Kurve den Probanden dazu veranlassen die Bremse zu ziehen.

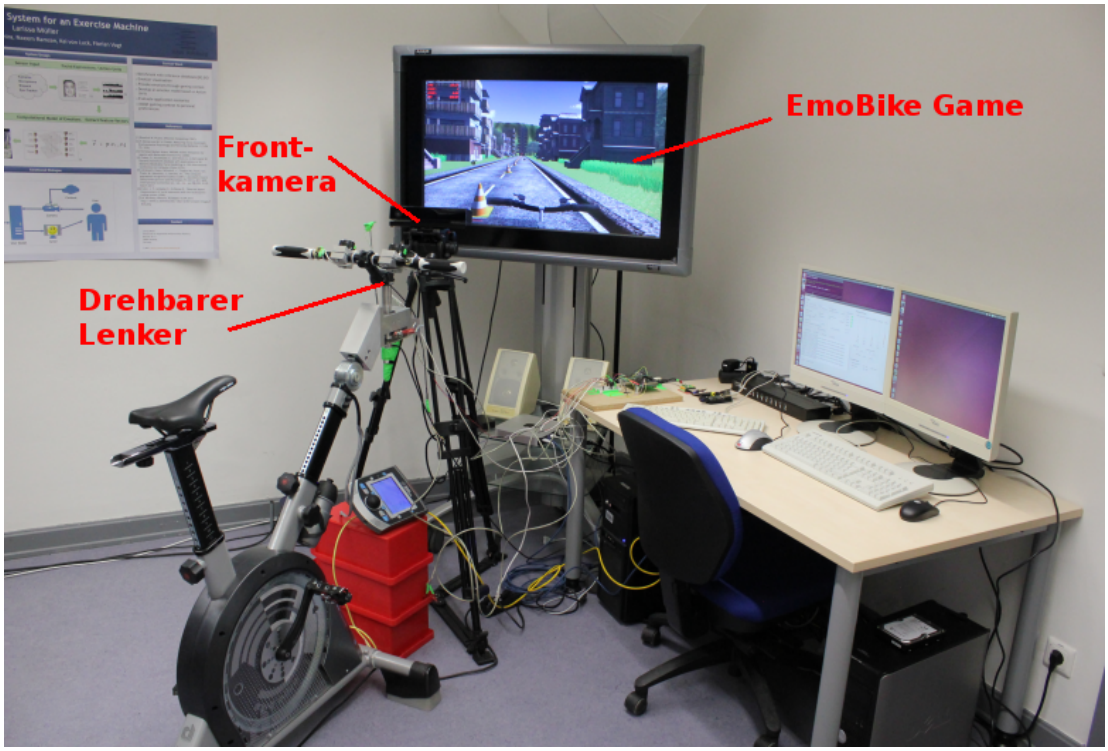


Abbildung 2.1.: Das Emobike Szenario. Die seitlich montierte Kamera ist nicht sichtbar.

[2]Die Darstellung der Spielwelt erfolgt durch die Augen des Akteurs

Der Proband wird mit Biosensoren ausgestattet die seinen körperlichen Zustand messen. Zwei Kameras sammeln visuelle Daten. Eine Kamera direkt vor dem Probanden zeichnet sein Gesicht auf während die zweite - seitlich montierte - Kamera die ganze Person erfasst. Anhand der Kameradaten werden die Gesten und Gesichtszüge des Probanden ausgewertet und daraus Emotionen abgeleitet. Zusammen mit den Biodaten kann die allgemeine Verfassung des Probanden ermittelt und überwacht werden.

3. Analyse

3.1. Aufgabe des Ergometers

Simulatoren sind die Nachbildung von komplexen realen Systemen und dienen der gefahrlosen Ausbildung an diesen Systemen. Beispiele sind der Fahrsimulator für Fahrschulen zur Vorbereitung auf die ersten Fahrstunden von Simutech [vgl. Simutech (2014)], der Radarsimulator der Hamburger Wasserschutzpolizei [vgl. Barco (2015)] und der Flugsimulator der TU Graz [vgl. TU Graz (2015)].

Die Fahrradsimulation im EmoBike Szenario verfolgt einen anderen Ansatz. Das Ziel ist nicht das Radfahren zu erlernen, sondern den Probanden während des Radfahrens zu beobachten. Es wurde bewusst eine allgemein vertraute Tätigkeit gewählt, die der Proband intuitiv ausführen kann ohne zuvor eine längere Einweisung zu erhalten oder ein Handbuch zu lesen. Die Voraussetzung dafür ist, dass das Ergometer als Benutzerschnittstelle der Fahrradsimulation, auch Mensch-Maschine-Schnittstelle oder Human Machine Interface (HMI) genannt, eine natürliche Steuerung bietet. So eine als Natural User Interface bezeichnete Benutzerschnittstelle ist dadurch gekennzeichnet, dass sie die Interaktion mit einem Computersystem so abbildet, wie es vom Vorbild aus der realen Welt bekannt ist [vgl. Bernin (2011)]. In diesem Zusammenhang bedeutet „natürlich“, dass sich das Ergometer wie ein echtes Fahrrad steuern lässt. Die Benutzbarkeit (Usability) eines Systems zeichnet sich unter anderem durch seine Erlernbarkeit aus. Durch die Wahl einer bekannten Steuerung ist das System sofort benutzbar. Die Aufmerksamkeit des Probanden wird nicht auf die Auseinandersetzung mit der Technik gelenkt sondern kann auf das EmoBike Game gerichtet werden, das die für die Untersuchungen notwendigen Szenarien und Ereignisse darstellt bzw. auslöst. Das Zusammenspiel aus einer natürlichen Steuerung, einer interaktiven, authentischen, digitalen Welt und einem realistischen Reaktionsverhalten der getätigten Interaktionen in der digitalen Welt ist die Voraussetzung für einen hohen Grad an Immersion. Also das Eintauchen in die virtuelle Umgebung, die die Präsenz eines Computersystems verblassen lässt (Disappeared Computing).

Abbildung 3.1.: Ausgangspunkt: Das Ergometer im Originalzustand (Foto: Daum Electronic)

Aus der Sicht des EmoBike Games ist das Ergometer (Abbildung 3.1) ein Eingabegerät bzw. ein Gamecontroller. Eine natürliche Steuerung von Computerspielen kann mit vielen Gamecontrollern nicht erreicht werden, weil sich die reale und die virtuelle Steuerung unterscheiden. Ein Beispiel ist ein Autorennspiel, das mit einem Gamepad gesteuert wird. Der analoge Miniaturjoystick bestimmt die Fahrtrichtung des Autos. Der Benutzer muss die Joystickstellung auf dem Gamepad auf den Lenkeinschlag im Spiel umsetzen. Der Erfolg hängt von der Erfahrung des Benutzers ab. Tauscht man das Gamepad gegen ein Lenkrad, führt das zu einer direkten Verknüpfung von der aus der Wirklichkeit bekannten Steuerung eines Autos und der Steuerung im Computerspiel. Dadurch ist es computerspielunerfahrenen Autofahrern möglich, ihre Fähigkeit in Bezug auf die Lenkung eines Fahrzeugs in die virtuelle Welt zu übertragen. Zur Steigerung der natürlichen Steuerung und Immersion könnten ein Autositz, Schalthebel sowie Pedalen für Bremse, Gas und Kupplung hinzugefügt werden. Sukzessiv würde eine

Autosimulation entstehen. Als Controller für eine Fahrradsimulation ist die Erweiterung des Ergometerinterfaces der wesentliche Aspekt, um das erwartete Verhalten eines Fahrrads nachzubilden. Das erwartete Verhalten wird auch als mentales Modell bezeichnet. Dieses Konzept besagt, dass die Bedienung einer Maschine nach einem Modell erfolgt, das ein Benutzer sich im Kopf bildet und von dem er annimmt, dass es die Funktionsweise des Systems widerspiegelt [vgl. Carroll (1990), zit. in: Bernin (2011)].

3.2. Das Benutzerinterface

Für die Entwicklung eines natürlichen Fahrradinterfaces müssen zunächst die Eigenschaften eines Fahrrads und deren Realisierung mit einem Ergometer analysiert werden. Die wichtigsten Interaktionen sind Lenken, Bremsen, Schalten und die Pedalen treten. Hinzu kommen weitere Bewegungen, die eher unterbewusst ausgeführt werden, wie das Gleichgewicht halten und sich beim Lenken in die Kurve lehnen. Ein Ergometer steht überlicherweise fest auf dem Boden. Um Neigungen in seitlicher Richtung zu ermöglichen muss entweder das Gestell des Ergometers oder der Untergrund angepasst werden. Letzteres kann unabhängig vom Gerät, z.B. mit einer gefederten Plattform, umgesetzt werden. Der Aufwand hängt davon ab, ob entsprechende Plattformen bereits verfügbar sind oder ob dazu eine Eigenentwicklung notwendig ist. Der alternative Umbau des Gestells ist voraussichtlich aufwendiger, denn neben einer Vorrichtung zum Neigen sind auch Änderungen am Ergometer erforderlich damit es mit der Vorrichtung verschraubt werden kann.

Zur Verbesserung der Steuerung müssen zusätzliche Interaktionsmöglichkeiten geschaffen werden. Davon ausgehend, dass jedes Fahrradergometer Pedalen besitzt, konzentrieren sich die Erweiterungen auf den Lenker. Um die Fahrtrichtung in einer natürlichen Weise bestimmen zu können, muss der Lenker drehbar sein. Dies kann erreicht werden indem ein Lenker mitsamt Halterung von einem vorhandenen Fahrrad genommen und an das Ergometer montiert wird. Oder es wird eine individuell für das Ergometer angepasste Lösung bestehend aus Halterung, Lenkstange und Lenker konstruiert und gefertigt. Beides ist mit hohem Aufwand verbunden und muss speziell für das verwendete Ergometer umgesetzt werden. Für Bremse und Gangschaltung können Standardbauteile verwendet werden, die an den Lenker montiert werden.

Neben den natürlichen Bewegungen spielt auch ein natürliches Verhalten des Benutzerinterfaces eine wichtige Rolle. Der Tretwiderstand muss sich abhängig von Gang und Bremse verändern. Das Ziehen der Bremse benötigt einen Grundwiderstand, der mit stärkerem Zug zunimmt. Im Zusammenspiel mit dem EmoBike Game kommen noch weitere Elemente hinzu.

Die virtuelle Umgebung nimmt ebenfalls Einfluss auf den Tretwiderstand, z.B. durch den Untergrund oder eine Steigung. Mit dem Schalten der Gänge muss die Geschwindigkeit im Game angepasst werden. Einige Änderungen lassen sich mechanisch lösen, andere benötigen eine Software zur Steuerung. Jede Interaktion des Menschen muss gemessen werden, damit eine passende Reaktion des Ergometers erfolgen kann. Welche Mittel zur Messung eingesetzt werden, und wie die Verbindung zur Software erfolgt, hängt von der konkreten Realisierung ab. Der Widerstand der Bremse könnte beispielsweise mit einer Feder realisiert werden und für den virtuellen Untergrund könnte ein Reibwiderstand definiert werden, der in die Berechnung des Tretwiderstands mit einfließt.

3.3. Ist-Zustand

Das Ergometer war zum Zeitpunkt des Projektbeitritts bereits vorhanden. Es handelt sich um ein Gerät des Typs „Ergo_Bike Premium 8i“ der Firma Daum Electronic (technische Daten siehe 5.2.1). Dieses Modell besitzt eine für das Projekt wesentliche Voraussetzung: Die optionale Steuerung durch ein externes Gerät. Das bedeutet, dass sich die Parameter des Ergometers nicht nur mit dem Gerätecomputer, dem sogenannten Cockpit, sondern auch von einem externen System auslesen und verändern lassen. Dies stellt eine wichtige Eigenschaft für den Informationsaustausch mit dem EmoBike Game dar.

Das Cockpit ist am Lenker des Ergometers angebracht und erlaubt die üblichen Einstellmöglichkeiten zu Strecke, Widerstand, Pulsbereich usw. Die Interaktionsmöglichkeiten sind auf das Treten und das Verändern von Widerstand und Gangschaltung über Knöpfe am Cockpit beschränkt.

Für das Ergometer existiert eine Software mit dem Namen „Ergoplanet“[1]. Damit lassen sich, im Vorfeld mit einer Kamera aufgenomme Strecken, mit dem Ergometer abfahren (Abbildung 3.2). Eine Richtungssteuerung ist jedoch nicht möglich, was zum einen an den linearen Videos liegt und zum anderen an fehlenden Bedienelementen am Ergometer. In der Nacht des Wissens 2013[2] wurden Versuche mit dem Ergometer in Kombination mit Ergoplanet durchgeführt. Die Ergebnisse zeigen deutlich, dass sich die Probanden mehr Interaktion mit dem Ergometer wünschen. Die fehlende Lenkung sorgte insbesondere in Kurven für ein Gefühl mangelnder Kontrolle, da die Fahrtrichtung vom Video vorgegeben wird und der Proband darauf keinen Einfluss nehmen kann. Weitere Kritik erhielt das Ruckeln des Videos wenn man zu langsam fährt. Die Bildwiederholrate des Videos hängt direkt mit der Geschwindigkeit des Ergometers

[1] www.ergoplanet.de (letzter Zugriff: 01.03.2015)
[2] www.nachtdeswissens.hamburg.de (letzter Zugriff: 01.03.2015)

zusammen. Ist diese nicht hoch genug erreicht das Video nicht die für eine flüssige Darstellung benötige Anzahl an Bildern pro Sekunde, üblicherweise 25 Frames per Second (FPS). Als Verbesserungsvorschläge wurden Fahrtwind und die Einbeziehung physikalischer Aspekte, wie z.B. das Ausrollen nachdem nicht mehr getreten wird, genannt.

Abbildung 3.2.: Versuchsaufbau mit dem Ergometer und Ergoplanet (Foto: Kai Rosseburg)

3.4. Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung eines intuitiven Gamecontrollers mit einem natürlichen Fahrrad-Benutzerinterface zur Steuerung des EmoBike Games basierend auf einem Fahrradergometer.

Diese Aufgabe gliedert sich in mehrere Teilbereiche. Die vorhandene Steuerung des Ergometers muss um zusätzliche Interaktionsmöglichkeiten erweitert werden, die sich an einem realen Fahrrad orientieren. Der Schwerpunkt liegt auf dem Ausbau des Lenkers, der zusammen mit den Pedalen notwendig ist um die Mindestanforderungen an die Interaktion mit dem EmoBike Game zu erfüllen: Fortbewegung und Richtungsbestimmung. Unabhängig davon wie die Len-

kung realisiert wird, muss der Lenkausschlag gemessen und an das EmoBike Game übertragen werden.

Die Fortbewegung bzw. Geschwindigkeit wird vom Cockpit (siehe Abschnitt 3.3) gemessen. Diese und andere Informationen, beispielsweise der Tretwiderstand, müssen von dort abgerufen und an das EmoBike Game übertragen werden. Zusätzlich ist eine Schnittstelle notwendig, die ein einfaches Verändern von Parametern im Cockpit ermöglicht und es dem EmoBike Game und anderen Komponenten erlaubt Parameter nicht nur zu lesen, sondern auch aktiv zu setzen. Ein Beispiel ist die Änderung des Tretwiderstands in Abhängigkeit von Steigung oder Untergrund in der virtuellen Welt.

Aufgrund unterschiedlicher Anforderungen (siehe 5.1.2) verteilen sich die EmoBike Komponenten auf mehrere Systeme. Um die Bedienung des Gesamtsystems zu vereinfachen wird eine GUI-basierte[3] Anwendung, das Operator Frontend, benötigt, die die Konfiguration und Überwachung der einzelnen Komponenten von einer zentralen Stelle aus ermöglicht. Die Komponenten sind lose miteinander gekoppelt und kommunizieren über Nachrichten. Es muss ein Nachrichtenprotokoll entwickelt und eine Infrastruktur zur Übertragung der Nachrichten integriert werden.

Zusammenfassung der Aufgaben:

- Erweiterung des Ergometers um ein natürliches Lenksystem
- Bereistellung einer einheitlichen Schnittstelle zur Steuerung des Ergometers
- Entwicklung eines Operator Frontends zur Konfiguration und Überwachung
- Aufbau eines Kommunikationssystems mit eigenem Nachrichtenprotokoll

3.5. Anforderungsanalyse

Lenkung

Das Ergometer besitzt keine Lenkmöglichkeiten. Der vorhandene Lenker ist starr und dient nur zum Festhalten. In einer ersten Testversion wird die Lenkung zunächst über zwei Taster realisiert, die das links- und rechts-Lenken erlauben. Die Taster werden mit einem Rechner verbunden, der den Anschluss von Sensoren über Ein-/Ausgabepins erlaubt. Der Rechner muss die Signale der Taster entgegen nehmen, daraus einen Richtungswert erstellen und diesen an

[3] GUI: Graphical User Interface

das EmoBike Game übertragen. Mit dieser Lösung lässt sich der gesamte Kommunikationsweg von der Lenkeingabe bis zur sichtbaren Änderung auf dem Display des Games überprüfen.

Das Lenken über Taster bildet nicht die uns bekannte natürliche Steuerung eines Fahrrads ab und stellt ein Hindernis in Bezug auf die Entwicklung eines Natural User Interfaces dar. Um dies zu verbessern wird im zweiten Schritt der starre Lenker durch einen drehbaren Lenker ausgetauscht. In Zusammenarbeit mit dem Studenten Thomas Berntien aus dem Fachbereich Fahrzeugbau wurde dazu eine entsprechende Lenkvorrichtung entworfen. Diese als Steckmodul konzipierte Vorrichtung ersetzt den bisherigen Lenker und ermöglicht die Drehbewegungen. Die Konstruktionszeichnungen wurden von Thomas Berntien erstellt (siehe Anhang A.2). Über einen an der Lenkstange befestigten Sensor muss die Drehrichtung erfasst und in einen Richtungswert umgewandelt werden. Dies geschieht ebenfalls über den oben genannten Rechner, der den ermittelten Richtungswert an das EmoBike Game überträgt.

Damit die Steuerung als vertraut bzw. natürlich wahrgenommen wird muss auf zwei Aspekte besondere Rücksicht genommen werden: Latenz und Präzision. Mit Latenz ist die Dauer vom Ausführen einer Bewegung bis zur Reaktion auf dem Bildschirm gemeint. Für den Menschen stellt sich eine Verzögerung bis 100 ms noch als Echtzeit dar [vgl. Nielsen (1993), zit. in: Bernin (2011)]. Liegt der Wert über 100 ms kann die Verzögerung von den Probanden wahrgenommen werden. Die Anbindung des Lenksystems über eine Netzwerkverbindung wird die Latenz erhöhen und es muss untersucht werden, ob der Schwellwert von 100 ms dadurch überschritten wird.

Der zweite Aspekt ist die Präzision. Die Lenkbewegung eines Probanden muss möglichst präzise mit den Auswirkungen auf dem Bildschirm übereinstimmen. Die Taster können nur Vollausschläge abbilden, d.h. maximaler Lenkausschlag nach links oder nach rechts. Wird keiner der beiden Taster betätigt entspricht das der Nullstellung und das Ergometer fährt gerade aus.

Beim drehbaren Lenker gibt es mehrere Dinge zu beachten. Im Gegensatz zu den Tastern werden hier viele Zwischenschritte gemessen. Diese müssen in einen passenden Richtungswert umgerechnet werden, so dass die virtuelle der realen Lenkerstellung entspricht. Bei einem Verlust von einzelnen Sensorwerten kann sich die Stellung des Lenkers verschieben, was man auch als Driften bezeichnet. Über einen längeren Zeitraum oder bei häufigen Verlusten kann diese Verschiebung groß genug werden um vom Fahrer bemerkt zu werden. Ist dies der Fall muss analysiert werden wie man dem entgegenwirken kann. Zuletzt kann es, abhängig von der Auflösung des Sensors, sein, dass die Zwischenschritte zu grob sind, was sich auf dem Bildschirm

durch ruckeln äußert. Sollte dies passieren müssen die Richtungswerte gegebenenfalls in der Software interpoliert werden.

Kommunikationssystem

Für den Informationsaustausch innherhalb des EmoBike Systems wird eine zentrale Schnittstelle für alle Komponenten benötigt. Der Aufbau eines Kommunikationssystems zur Vermittlung von Nachrichten (Abschnitt 4.1) und der Entwurf eines Nachrichtenprotokolls (Abschnitt 4.4) sollen sicherstellen, dass sowohl der Transport als auch die Verarbeitung der Nachrichten einem einheitlichen Muster folgen und somit eine wohldefinierte und unabhängige Schnittstelle für aktuelle und zukünftige Systeme zur Verfügung steht. Beim Design des Nachrichtenprotokolls sollte auf eine flexible und erweiterbare Struktur geachtet werden.

Ergometer Schnittstelle

Das Ergometer lässt sich von einem externen System mithilfe eines herstellerspezifischen Protokolls [Daum Electronic GmbH (2005)] steuern. Zur nahtlosen Integration in das EmoBike System müssen die Protokollbefehle gekapselt werden, so dass das Ergometer vom EmoBike Game und anderen Komponenten gesteuert werden kann. Voraussetzung dafür ist ein Adapter (Adapterpattern [vgl. Gamma u. a. (1995)]), der zur einen Seite hin eine Verbindung zum Ergometer, genauer dem Cockpit, herstellt und zur anderen Seite einen Anschluss an das Kommunikationssystem aus 3.5 realisiert. So können einfache Getter- und Setter-Nachrichten[4] von den EmoBike Komponenten empfangen und in passende Ergometer-Befehle übersetzt werden. Die Entwicklung des Adapters ist die Vorausetzung für die Übertragung der Geschwindigkeit an das EmoBike Game und die Steuerung des Tretwiderstands in Abhängigkeit von der virtuellen Umgebung.

Operator Frontend

Das sogenannte ControlCenter ist das Operator Frontend und die grafische Benutzerschnittstelle zur Bedienung des EmoBike Systems. Die Anforderungen an das ControlCenter sind unterschiedlich. Für ein Experiementiersystem wie das EmoBike ist es nützlich, wenn sich Parameter zur Laufzeit ändern lassen. Mit einer Anbindung an das Kommunikationssystem aus 3.5 lassen sich die Eigenschaften der einzelnen Systeme anpassen, sofern diese eine entsprechende Schnittstelle bereitstellen. Ein Beispiel ist die zuvor genannte Ergometer Schnittstelle, deren Aufgabe genau darin besteht: Die Bedingungen der Simulation wie Tretwiderstand oder

[4] Befehle zum Auslesen und Setzen von Werten

Steigung sollen sich jederzeit vom ControlCenter aus modifizieren lassen. Hintergründe sind die Einflussnahme auf das Verhalten des Probanden, die Korrektur von etwaigem Fehlverhalten der Systeme und die Untersuchung des Verhaltens an den Parametergrenzen, die in der Simulation selten oder nie erreicht werden. Darüber hinus soll das ControlCenter die aktuellen Systemzustände und die laufende Nachrichtenkommunikation anzeigen und teilweise die Steuerung des EmoBike Games ermöglichen. Beispielsweise kann das Laden der Levels mit dem ControlCenter die Durchführung von Versuchen erleichtern, da die Levels nicht direkt am EmoBike Game PC ausgewählt werden müssen, wofür der laufende Versuch unterbrochen werden müsste.

4. Design

4.1. Architektur

Der Aufbau des EmoBikes verfolgt das Konzept eines verteilten Systems [vgl. Tanenbaum und Bos (2014)]. Dies ist ein Zusammenschluss von unabhängigen Prozessen (Rechnern oder Anwendungen), die ihre Informationen nicht über einen gemeinsamen Speicher, sondern über Nachrichten miteinander teilen. Nach außen hin stellen sich die verteilten Prozesse als ein einziges System dar. Hinter den Prozessen verbergen sich hier die verschiedenen Komponenten für Sensorik und Aktorik. Mit der losen Kopplung über Nachrichten lässt sich das System als Plug-In Architektur realisieren, die den Austausch einzelner Komponenten und die Erweiterung um zusätzliche Komponenten ermöglicht, ohne sich dabei auf die Funktion der vorhandenen Komponenten auszuwirken. Der Aufbau orientiert sich am Model-View-Controller (MVC) Entwurfsmuster [vgl. Buschmann u. a. (1996)]. Abbildung 4.1 zeigt die logische Anordnung, die zusätzlich zu den MVC-Einheiten eine Datenerfassung zur persistenten Speicherung besitzt [vgl. Zagaria (2015)]. Über ein Netzwerk werden die Nachrichten übertragen, die Vermittlung erfolgt mit einem sogenannten Messagebroker (siehe Abschnit 4.3).

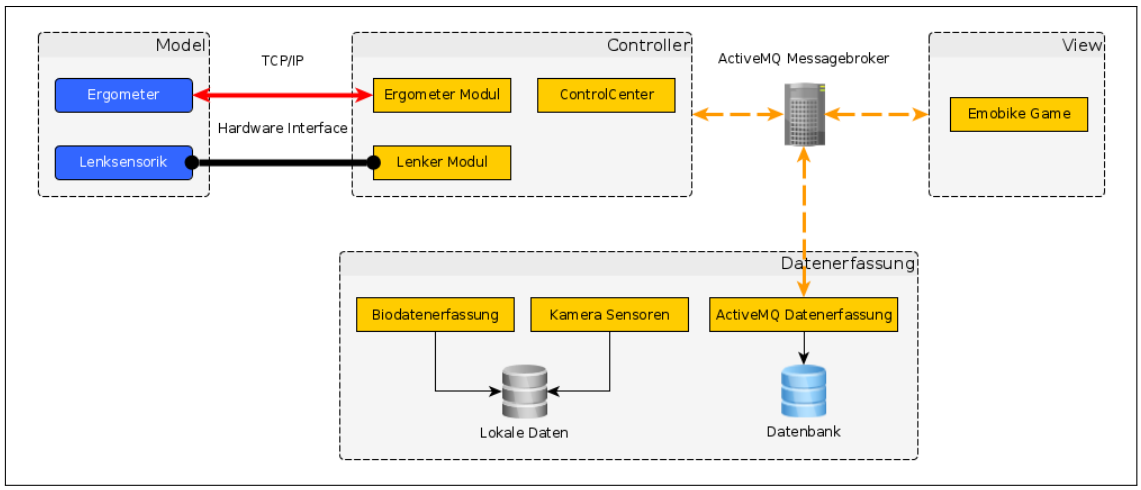


Abbildung 4.1.: EmoBike Model-View-Controller Architektur (Grafik: Zagaria (2015))

4.2. Model-View-Controller

Mit dem Model-View-Controller Entwurfsmuster wird in diesem Projekt das Ziel verfolgt die Einheiten voneinander unabhängig zu halten. Dadurch ist es möglich einzelne Teile auf andere Plattformen zu portieren, Änderungen vorzunehmen oder auch Komponenten wieder zu verwenden, ohne das Gesamtsystem zu beeinflussen. Die Implementierung der Controller-Module und den daran angebundenen Modelkomponenten ist zentraler Bestandteil dieser Arbeit. Die Bereiche View und Datenerfassung werden von anderen Mitgliedern des Gesamtprojekts EmoBike entwickelt.

Model :

Die Sensoren und Aktoren von Ergometer und Lenker halten die darzustellenden Rohdaten. Diese Rohdaten bilden das Model für das EmoBike Game. Das Ergometer wird über ein proprietäres Netzwerkprotokoll (siehe 5.5) gesteuert mit dem sich Geschwindigkeit, Widerstand und Umdrehung der Pedalen abfragen lassen. Zusätzlich kann hierüber der Widerstand gesetzt werden. Der Sensor am Lenker erfasst die aktuelle Lenkrichtung, die die Fahrtrichtung im EmoBike Game beeinflusst.

Controller :

Der Controller besteht aus mehreren Softwarekomponenten, die die Daten zwischen dem Model und der View vermitteln. Hier werden die vom Model gelieferten Rohdaten aufbereitet und über Netzwerk an den Messagebroker gesendet. Von dort gelangen die Daten zum EmoBike Game. In umgekehrter Richtung kann das EmoBike Game Änderungen anfordern, die vom Controller an das Model übergeben werden. Alle EmoBike Komponenten, die sich für die Daten des Models interessieren, müssen sich beim Messagebroker registrieren, der dafür sorgt, dass die Daten an alle Teilnehmer übermittelt werden.

Das Ergometermodul sorgt für die Anbindung des Ergometers an den Messagebroker und das Lenkermodul empfängt die Signale der Lenksensoren, die über ein Hardware Interface direkt an den Rechner angeschlossen werden, der das Modul ausführt. Mit dem ControlCenter lassen sich die Aktivitäten des EmoBike Systems überwachen und teilweise steuern. So lassen sich die Module hierüber neu starten und einige Parameter wie Widerstand und Steigung zu Testzwecken setzen. Alle über den Messagebroker versandten Nachrichten werden in einem Logbereich angezeigt.

View :

Das EmoBike Game ist die Präsentationskomponente des EmoBike Systems. Es visuali-

siert die Daten des Models. Zusätzlich ist es die Aufgabe des EmoBike Games Ereignisse auszulösen um beim Probanden bestimmte emotionale Reaktionen hervorzurufen. Die Reaktionen werden erfasst und abgespeichert. Die Game Ergeignisse werden über den Messagebroker zur Speicherung an die Datenerfassung übertragen.

Datenerfassung :
Aufgabe der Datenerfassung ist die persistente Speicherung der Daten für spätere Analysen. Alle über den Messagebroker ausgetauschten Nachrichten werden in eine Datenbank geschrieben. Kamera- und Biodaten werden aufgrund ihrer Größe in Dateien gespeichert.

4.3. Der Messagebroker ActiveMQ

Allgemein dient ein Messagebroker zur Vermittlung von Nachrichten zwischen einem Sender und einem oder mehreren Empfängern. Die gesamte Kommunikation wird in diesem Projekt über den Messagebroker Apache ActiveMQ abgewickelt. Diese Software wird im „Living Place Hamburg“ eingesetzt und hat sich dort bewährt [vgl. Otto und Voskuhl (2011)]. Um auf die dort gesammelten Erfahrungen zurückgreifen zu können wird auch hier dieser Messagebroker verwendet. Apache ActiveMQ ist ein freier, in Java programmierter und unter Apache 2.0 Lizenz [apache.org (2015)] stehender Messagebroker. Er bietet eine breite Unterstützung von Programmiersprachen wie Java, C, C++, C#, Ruby, Perl, Python und PHP.

Es gibt zwei unterschiedliche Konzepte der Nachrichtenvermittlung:

Queue :
In einer Queue gibt es zwei Arten von Clients: Producer und Consumer. Der Producer erzeugt Nachrichten und sendet diese an die Queue. Der Messagebroker erkennt die neue Nachricht in der Queue und informiert die Consumer, dass eine Nachricht zur Abholung (Verbrauch) bereit steht. Der Consumer mit der kürzesten Reaktionszeit entnimmt der Queue die Nachricht. Die Nachricht steht den anderen Consumern dann nicht mehr zur Verfügung. Dies ist ein klassisches Erzeuger-Verbraucher-System.

Topic :
Das Topic kennt ebenfalls zwei Arten von Clients: Publisher und Subscriber. Dies sind lediglich andere Namen für die gleichen Funktionen von Producer und Consumer in der Queue. Der Unterschied liegt in der Bereitstellung der Nachrichten. Diese werden nicht von einem Consumer verbraucht sondern bleiben erhalten und können von jedem Subscriber gelesen werden.

In der EmoBike-Architektur sind Nachrichten häufig für mehrere Empfänger bestimmt weshalb alle Nachrichtenkanäle als Topics erstellt werden. Abbildung 4.2 zeigt die verwendeten Topics und die Nachrichten die darüber versendet werden.

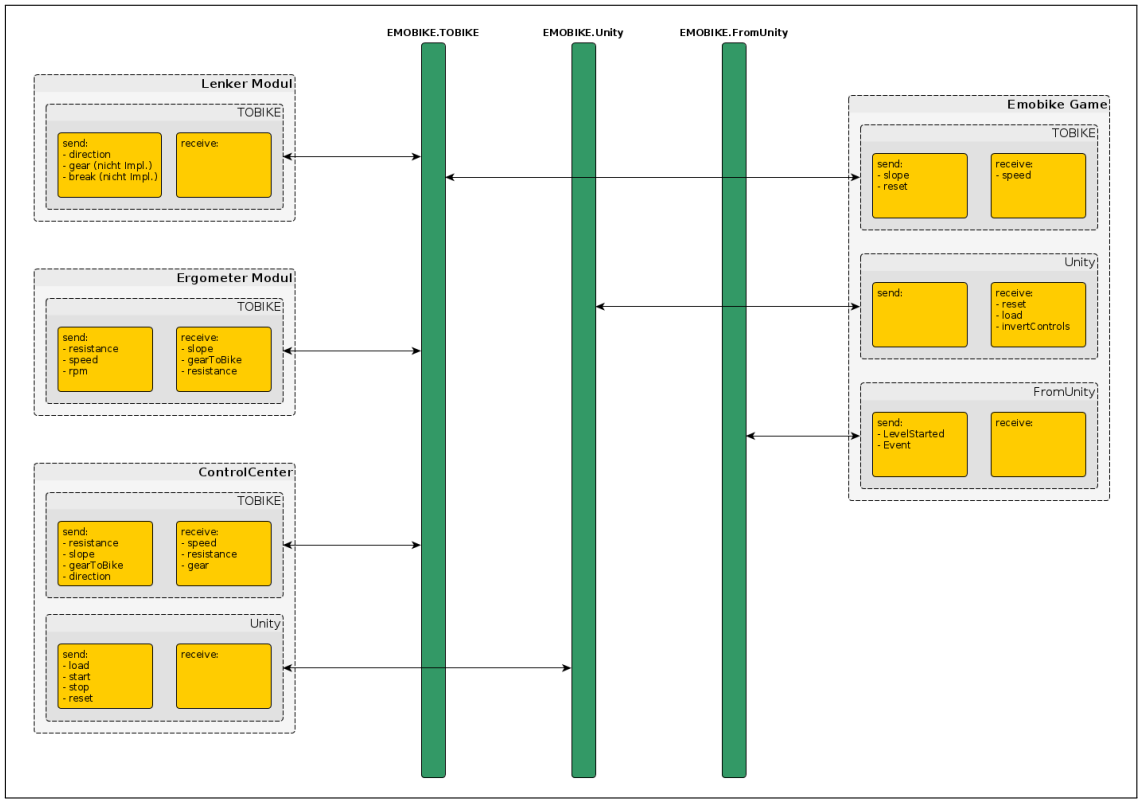


Abbildung 4.2.: EmoBike ActiveMQ Topics

4.4. EmoBike Nachrichtenprotokoll

Die Kommunikation in einem verteilten System erfolgt über Nachrichten. Diese lose Kopplung ermöglicht die in Abschnitt 4.1 genannte Plug-In Architektur. Damit sich alle Komponenten innerhalb des Systems verständigen können muss ein Protokoll definiert werden, das die Synthax und die Semantik der Nachrichten festlegt. Die Struktur der zu übertragenden Daten ist unterschiedlich. Sie reicht von einfachen Anweisungen über Messwerte bis hin zu ganzen Objekten. Die Notation erfolgt in JSON. Ein Vorteil solcher Datenaustauschformate in Textform ist ihre Lesbarkeit auch im nicht interpretierten Zustand. Parser[1] gibt es für alle verbreiteten

[1]Parsen: Zerlegen einer Eingabe in ein für die Weiterverarbeitung notwendiges Format

Programmiersprachen. Dies ist ein wichtiger Punkt, denn trotz der projektinternen Festlegung auf Qt/C++ als Standardprogrammiersprache gibt es Abweichungen wie die vom EmoBike Game verwendete Gameengine Unity, die C# verwendet. Zukünftige Erweiterungen können ebenfalls auf Schnittstellen oder Software Developer Kits (SDK) angewiesen sein, die nicht Qt verwenden.

Protokollformat

Die zu übertragenden Daten bestehen hauptsächlich aus Sensorwerten und Events. Die erste Version sieht die Verwendung von Key-Value-Paaren im Stringformat vor. Das betrifft sowohl die Metadaten als auch die Nutzdaten. Das erste Key-Value-Paar identifiziert den Absender der Nachricht. Das ermöglicht die eindeutige Zuordnung jeder Nachricht und erlaubt den Anwendungen selbst verschickte Nachrichten direkt herauszufiltern, falls sie diese empfangen. Das zweite Key-Value-Paar bestimmt den Typ der Nachricht. „Direction“ in Abbildung 4.3a gibt an, dass diese Nachricht einen Richtungswert enthält.

```
/* Sensorwert */
{
    "sender": "raspberry",
    "type":   "direction",
    "steer":  "-0.10",
}
```

(a) JSON Nachricht mit Richtungswert

```
/* Ergebnis */
{
    "sender":    "unity",
    "type":      "result",
    "coins":     "20",
    "time":      "30000",
    "trycount": "2",
}
```

(b) JSON Nachricht mit Ergebnissen eines Levels

Als Information würde es ausreichen als nächstes direkt den Richtungswert anzugeben. Ein Wert darf aber nicht einzeln auftreten, weshalb als Schlüssel die Einheit des Datums eingetragen wird. In diesem Beispiel ist das der Lenkausschlag. Eine Übersicht über die einzelnen Nachrichtentypen und deren Wertebereiche kann der Abbildung 4.5 entnommen werden. Diese drei Key-Value-Paare bilden die kleinste Nachricht, die verschickt werden kann. Für zusätzliche Informationen können weitere Key-Value-Paare angehängt werden. Abbildung 4.3b zeigt das Ergebnis eines Game Levels. Das Level ist so konzipiert, dass man in möglichst kurzer Zeit alle Münzen auf einem Slalomparcour einzusammeln muss (Abbildung 4.4). Wurden beim Überfahren der Ziellinie noch nicht alle Münzen eingesammlt, wird man auf den Startpunkt zurück gesetzt und muss die verbleibenden Münzen einsammeln. Die Zeit läuft währenddessen weiter. Der Nachrichtentyp ist hier „result“. „Coins“ gibt die Anzahl der eingesammelten Münzen an, „time“ die insgesamt benötigte Zeit zum Einsammeln aller Münzen und „trycount“ wie oft der Parcour abgefahren wurde.

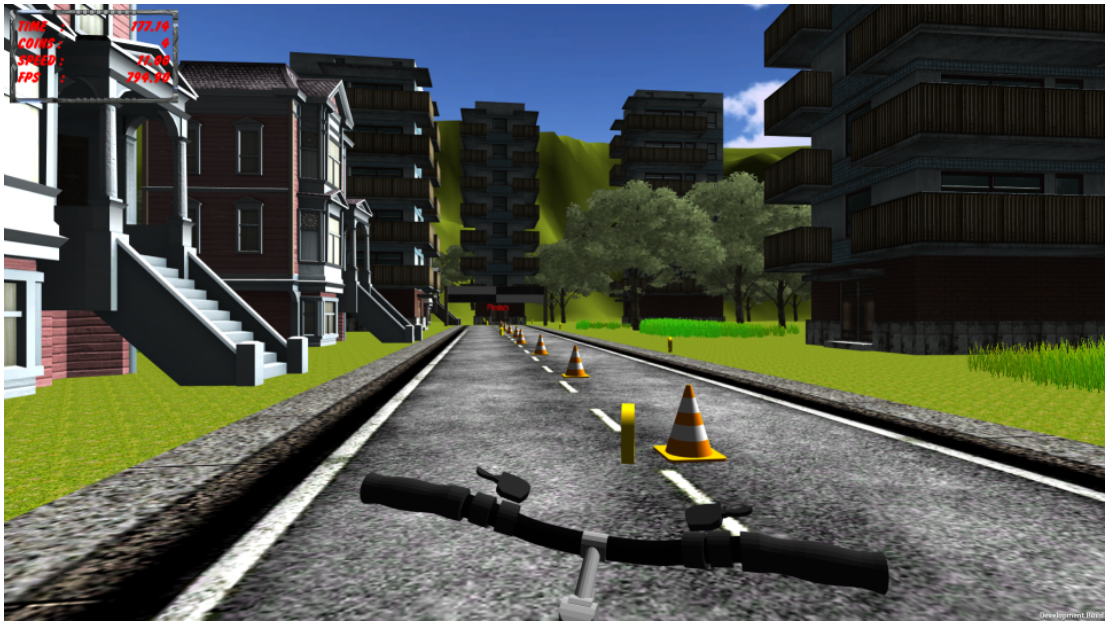


Abbildung 4.4.: Münzen einsammeln im Slalomlevel

Nachricht	Funktion	sender	type	data	Format (C-Syntax)
Richtung	Richtungsausschlag	raspberry1	direction	-1.00 bis 1.00	%1.2f
Referenz	Referenzsignal	raspberry1	reference	-	-
Geschwindigkeit	Geschwindigkeit in Meter pro Sekunde	ergometermodule	speed	-28.0 bis 28.0	%2.1f
Drehzahl	Umdrehungen pro Minute	ergometermodule	rpm	-200.00 bis 200.00	%3.2f
Belastung	Widerstand in Watt	any	resistance	20 bis 1000	%d
Steigung	Steigung in Grad	unity	slope	0.00 bis 360.00	%3.2f
Gangschaltung	Gänge schalten	raspberry2	gear	1 bis 28	%d
Bremse	Geschwindigkeit/Widerstand anpassen	raspberry2	break	0 bis x	%d
Reset	Level neu starten	controlcenter	reset	-	-
Load	Level laden	controlcenter	level	Level1-5	String
Messdaten	Ergebnis Slalomlevel	unity	result	coins (0 bis 100000) time (in msec) retries	%d %d %d
Level geladen	Level wurde geladen	unity	levelLoaded	-	-
Münze eingesammelt	Coin wurde erreicht	unity	coinCollected	-	-

Abbildung 4.5.: Liste der Nachrichten

5. Realisierung

5.1. Gesamtarchitektur

Das EmoBike ist ein verteiltes System bestehend aus mehreren Softwaremodulen, die sich auf unterschiedliche Hardwarekomponenten verteilen.

5.1.1. Hardwarekomponenten

Die Hardware besteht aus dem Ergometer, einem sogenannten „System-on-a-Chip“ (SoC) zur Sensorsignalerfassung und zwei PCs: Game und Operator PC (siehe Abbildung 5.1). Der Game PC führt das EmoBike Game aus und gibt das Bild auf dem vor dem Ergometer platzierten Display wieder. Der Operator PC dient zur Konfiguration und Überwachung der Systeme, hier läuft das ControlCenter. Alle Geräte sind über ein TCP/IP-Netzwerk miteinander verbunden. Die Kommunikation erfolgt über den Messagebroker und das EmoBike Nachrichtenprotokoll.

5.1.2. Softwaremodule

Der Ort der Ausführung der meisten Module wird durch ihre Funktion bestimmt. So muss das Modul zur Erfassung der Lenkrichtung auf einem Rechner laufen, der die entsprechenden Anschlüsse für die Sensoren bereitstellt. Für andere Module spielt das Betriebssystem eine Rolle. Für das EmoBike Game ist ein Windows-Betriebssystem erforderlich. Andere Module, wie das Ergometer Modul, sind unabhängig und können auf einem wenig ausgelasteten System wie dem Operator PC laufen.

Abbildung 5.2 zeigt die einzelnen Module und ihre logischen Verbindungen. Das EmoBike Game ist hier eine Blackbox in die bestimmte Information hinein gehen und andere Informationen heraus kommen. Zum einen sorgt sie für das visuelle Feedback der Interaktion mit dem Ergometer zum anderen hat sie Einfluss auf das Verhalten des Ergometers, zum Beispiel ein veränderter Tretwiderstand beim Wechsel des Untergrunds oder bei einer Steigung.

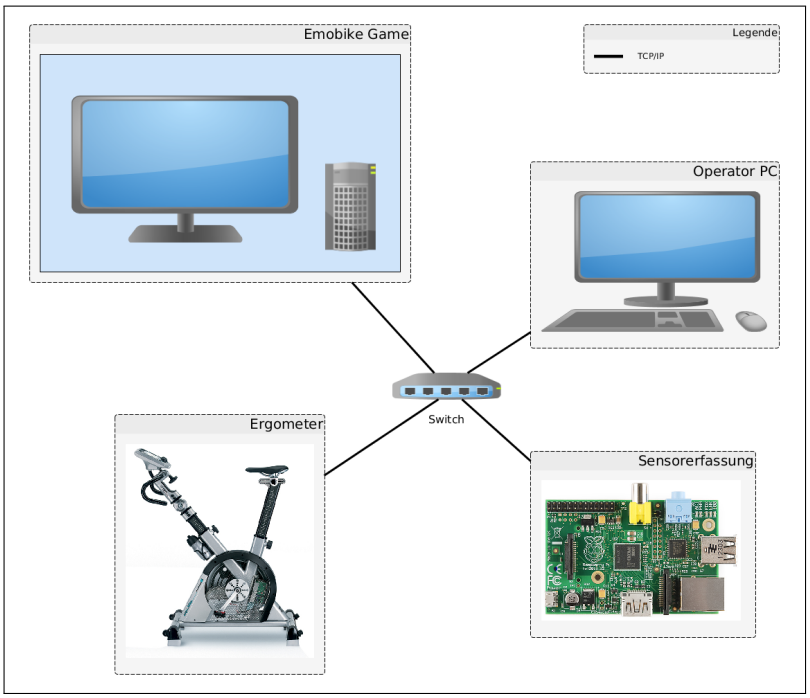


Abbildung 5.1.: Hardwarekomponenten Fotos: Ergometer: Daum Electronic, Raspberry Pi: cdn.sparkfun.com (2015))

Im Lenker Modul werden die Daten vom Sensor des Lenkers erfasst, daraus ein Richtungswert errechnet und dieser an den Messagebroker übergeben. Das Ergometer Modul fragt die aktuelle Geschwindigkeit vom Ergometer Cockpit ab und sendet diese ebenfalls an den Messagebroker. Das Ergometer Modul steuert auch den Widerstand basierend auf den vom Messagebroker empfangenen Parametern. Über das ControlCenter lassen sich die Systeme konfigurieren, steuern und überwachen. Der Messagebroker ist das zentrale System für die Nachrichtenkommunikation. Abbildung 5.3 zeigt welches Softwaremodul auf welcher Hardwarekomponente ausgeführt wird.

5.2. Eingesetzte Hardware

5.2.1. Ergometer

Die Basis für das EmoBike bildet das Ergometer „Ergo_Bike Premium 8i“ von Daum Electronic. Das Modell ist für den semiprofessionellen Bereich gedacht und bietet einen Tretwiderstand zwischen 20 und 1000 Watt. Alle Einstellungen inklusive der Trainingsprogramme lassen sich über das Cockpit vornehmen. Das Cockpit verfügt über einen Netzwerkanschluss über

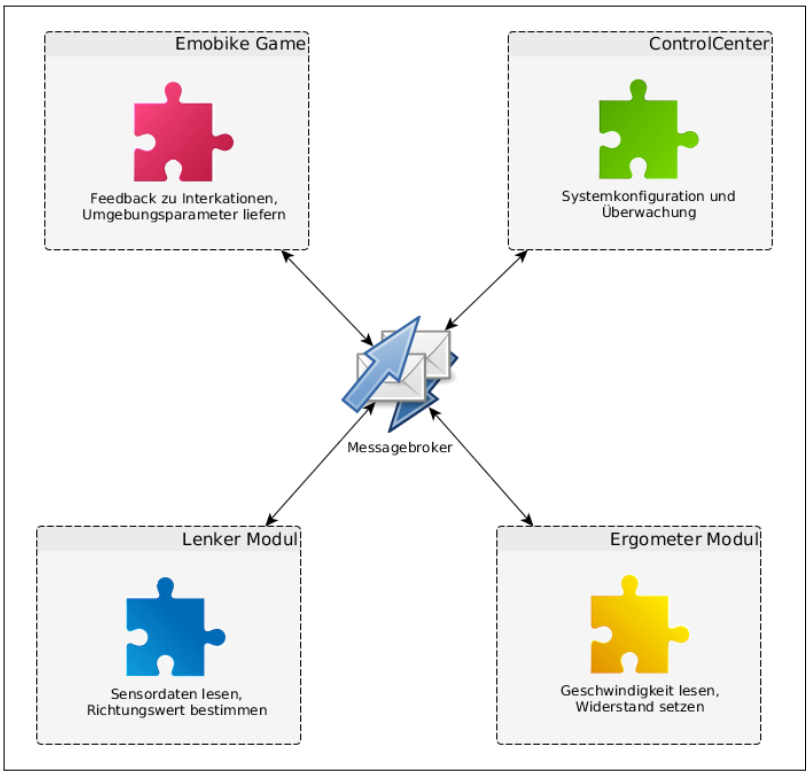


Abbildung 5.2.: Softwaremodule

den sich das Gerät mit dem in 5.5 beschriebenen Protokoll fernsteuern lässt. Die weiteren Spezifikationen lauten [daum electronic.de (2015)]:

- Programmierbare Grenzwerte für Watt, Puls, Energieverbrauch, Trainingszeit, Distanz
- Grafische Anzeige aller Daten (Watt, Puls ...) für jedes einzelne Training
- Grafischer Vergleich eines aktuellen Trainings mit einem früheren Referenztraining Fitness-Ermittlung und -Auswertung
- Über 100 Trainingsprogramme
- Höhenprofilprogramme, Triathlon-Wettkampfstrecken, Elektronische Gangschaltung (28 Gänge), Wattgesteuerte Festprogramme, Kraftprogramme, RPM-gesteuerte Programme,
- Pulsgesteuerte Festprogramme, Manuelles Cardioprogramm
- ErgoPlanet kompatibel
- Großes farbiges Grafikdisplay
- Gewicht: 45 kg

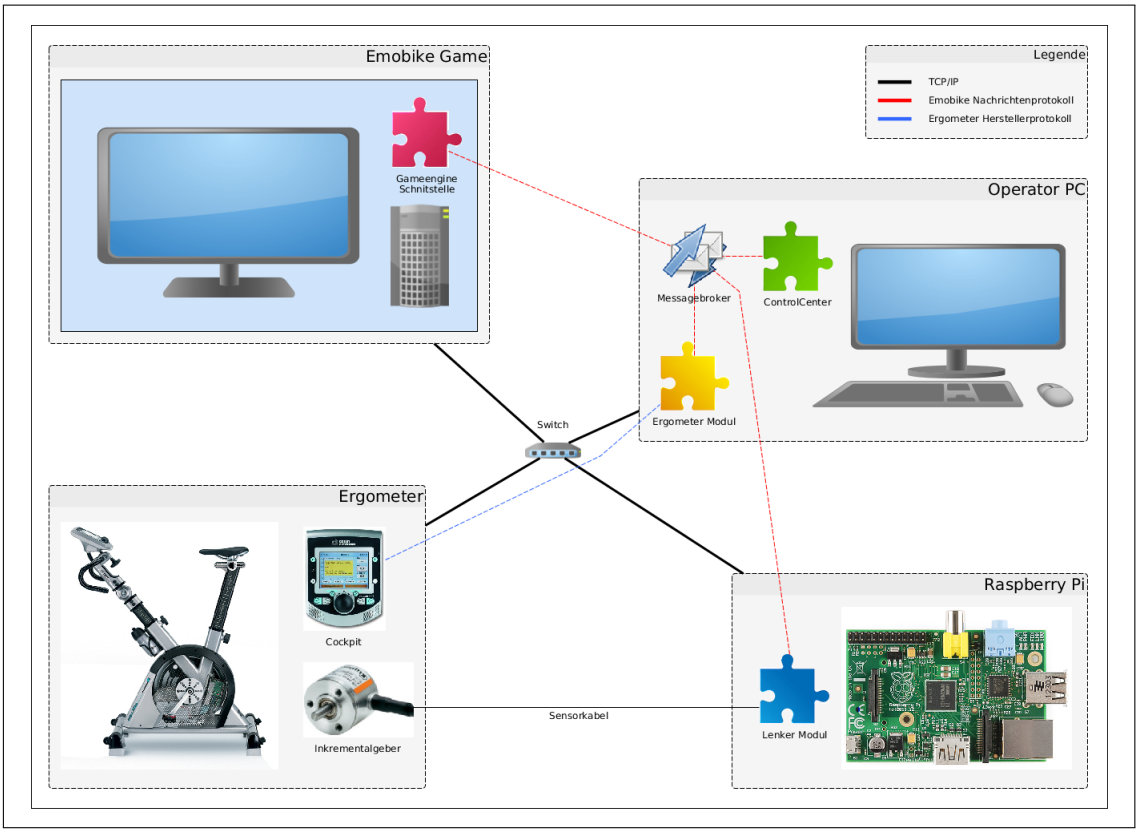


Abbildung 5.3.: Gesamtarchitektur (Fotos: Ergometer und Cockpit: Daum Electronic, Inkrementalgeber: conrad.de (2015), Raspberry Pi: cdn.sparkfun.com (2015))

5.2.2. Raspberry Pi

Die Messung des Lenkausschlags am Ergometer erfolgt über Sensoren (Taster oder Inkrementalgeber, siehe 3.5 und 5.6), deren Signale von einem Computer erfasst und verarbeitet werden müssen. Zum Anschluss der Sensoren sind sogenannte GPIO-Pins (General Purpose Input Output Anschlüsse) notwendig, die z.B. von „SoC“s, nicht aber von üblichen Desktop-PCs, bereitgestellt werden. Der Raspberry Pi ist ein solches Ein-Chip-System. Er wurde als als Lern- und Experimentiersystem entwickelt [raspberrypi.org (2015)] und besitzt ausreichend Leistung um ein vollwertiges Betriebsystem, wie Linux, auszuführen. Die Kombination aus GPIO-Pins, einem Netzwerkanschluss und die Ausführung von Linux schafft die notwendige Grundlage um die Sensoren des Lenksystems in die Fahrradsimulation zu integrieren. Das hier verwendete Modell B besitzt folgende technische Eigenschaften [raspberry projects.com (2015)]:

- Arbeitsspeicher: 512MB RAM

- Anschlüsse: 2 USB Ports, Ethernet Port, 3.5mm Jack Audioausogang, HDMI, Composite Video, SD-Karten Slot
- Prozessor: Broadcom BCM2835 SoC mit einem ARMv6 Prozessor mit 700MHz und einer Videocore 4 GPU

5.3. Eingesetzte Software

5.3.1. Qt Framework

Alle Module wurden gemäßt der Projektvorgabe in Qt programmiert. Qt ist ein Framework für C++ das viele Klassen zur Verfügung stellt, die die Programmierung erleichtern. Ein gutes Beispiel ist die Klasse QString mit der man Strings ähnlich gut wie in Java verarbeiten kann. Mit dem Qt-Creator steht eine grafische Entwicklungsumgebung zur Verfügung die das Framework voll integriert und mit dem GUI-Builder auch die Erstellung von grafischen Benutzeroberflächen ermöglicht. Die Bereitstellung für verschiedene Betriebssysteme wie Windows, Linux und Mac OS X, aber auch Android oder iOS, erlaubt die Entwicklung von plattformunabhängiger Software. Prominente Beispiele sind Adobe Photoshop Elements, TeamSpeak, Google Earth sowie die grafischen Benutzeroberflächen von VLC und VirtualBox [wikipedia.org (2015)]. Qt ist mit der zugrunde liegenden Programmiersprache C++ gut für die Entwicklung von hardwarenaher Software geeignet, was die Entwicklung auf dem Raspberry Pi begünstigt.

5.3.2. ARM Crosscompiler

Wie alle „SoC“s besitzt der Raspberry Pi, im Vergleich zu Desktoprechnern, eine stark eingeschränkte Rechenleistung. Qt und der Qt-Creator lassen sich zwar auf dem Raspberry Pi installieren und nutzen, doch dauert das Übersetzen selbst einfacher Programme lange. Dies kann man umgehen indem man sich eine Crosscompile-Umgebung einrichtet mit der die Programme auf einem schnellen Rechner, aber für die Zielarchitektur, kompiliert werden. Hier kommt ein speziell auf den Raspberry Pi abgestimmter GNU C Compiler [popcornmix (2015)] für ARM-Prozessoren zum Einsatz. Die Einrichtung erfolgte nach dieser Anleitung: qtproject.org (2015).

5.4. Erweiterung des Nachrichtenprotokolls

Die erste Protokollversion ist in der Lage Sensordaten und einfache Anweisungen, wie „Level laden“, zu übertragen. Zusätzliche Anforderungen machen eine Überarbeitung und Erweiterung des Protokolls notwendig, die im Folgenden dargestellt wird.

Protokollerweiterung

Eine der Anforderungen ist die Speicherung aller versandten Nachrichten in einer Datenbank. Dies soll über reine Analysezwecke hinaus sogenannte Replays, also das erneute Abspielen eines Nachrichtenverlaufs eines bestimmten Zeitraums oder eines bestimmten Levels, ermöglichen. Das Nachrichtenformat muss dazu um einen Zeitstempel erweitert werden, damit sowohl die chronologische Reihenfolge als auch der zeitliche Abstand der Nachrichten zueinander eindeutig sind. Die Zeitstempel werden in Unixzeit angegeben. Die Unixzeit ist ein Zähler, der die Anzahl der vergangenen Sekunden seit dem 01.01.1970 00:00 Uhr UTC angibt. Über Zeitfunktionen lässt sich dieser Wert auf jedem Betriebssystem ausgeben. Eine Umrechnung in andere Zeitzonen ist möglich. Im Projekt werden alle Zeitstempel als GMT+1 angegeben. Eine Genauigkeit im Sekundenbereich reicht an dieser Stelle aber nicht aus, denn allein das Lenkermodul erzeugt 100 Nachrichten pro Sekunde. Mit der Zeitfunktion „gettimeofday“ erhält man zusätzlich einen Zähler für die Mikrosekunden. Zusammen mit dem anderen Sekundenzähler lässt sich daraus ein Mikrosekunden genauer Zeitstempel zusammenstellen. Bei der Verarbeitung solcher Zeitstempel muss auf die Wahl des Datentyps geachtet werden. 32 Bit Datentypen sind für die Zeitstempel zu klein, weshalb beim Parsen der Nachrichten konsequent 64 Bit Datentypen eingesetzt werden müssen. Die zum Generieren von ActiveMQ Nachrichten entwickelte JSON-Klasse enthält eine Methode „getTimestamp“ (Listing 5.1), die aus den Sekunden und Mikrosekunden einen String, umgerechnet auf die Zeitzone GMT+1, zusammensetzt und zurück gibt. Ein Beispiel ist der String „1423756202248925“, der in menschenlesbarem Format „12.02.2015 16:50:02.248925“ entspricht.

```
1  QString JsonTools::getTimestamp( void )
2  {
3      struct timeval  tv;
4      struct timezone tz;
5      struct tm       *tm;
6      quint64 offset;
7      quint64 value;
8
9      gettimeofday(&tv, &tz);      // get current time in unix format
10     tm = gmtime(&tv.tv_sec);     // get broken-down time for current time zone
11     offset = tm->tm_gmtoff;      // local time offset
12     value = (tv.tv_sec+offset) * 1000000 + tv.tv_usec;
13
14     return QString::number(value);
15 }
```

Listing 5.1: getTimestamp Methode aus JSON-Klasse

Eine weitere Anforderung ist das Versenden von Game Events wie LevelStarted, LevelFinished und das Ergebnis eines Levels. Diese Events sind im EmoBike Game als Objekte hinterlegt. Die Zerlegung dieser Objekte in Key-Value-Paare, wie es Version 1 des Protokollformats verlangt,

ist umständlich und unflexibel. JSON erlaubt die automatische Serialisierung von Objekten. Dabei werden die Eigenschaften eines Objekts in eine Liste umgewandelt die Key-Value-Paare, Arrays oder weitere Objekte enthalten kann. Im folgenden wird dies als Map bezeichnet.

Bei der Definition von Version 1 blieben die Steuernachrichten unberücksichtigt weshalb dort der Workaround mit den leeren Strings im dritten Key-Value-Paar angewandt wurde (Abschnitt 4.4). In der neuen Version werden jetzt auch solche Nachrichten unterstützt. Über einen zusätzlichen Schlüssel „datatype“ wird in jeder Nachricht das Format der Daten angegeben. Insgesamt gibt es drei Datentypen: KEYVALUE, MAP und NONE. Der letztgenannte Datentyp kann von den Steuernachrichten verwendet werden. Eine Nachricht lässt sich logisch in Kopfbereich (Header) und Nutzlast (Payload) unterteilen. Der Header, bestehend aus „sender“, „timestamp“, „type“ und „datatype“, muss in jeder Nachricht vorhanden sein, während sich die Nutzlast entsprechend des angegebenen Datentyps unterscheidet. Bei KEYVALUE wird genau ein Key-Value-Paar erwartet, bei MAP ein Objekt, das dem Typ der Nachricht („type“) entspricht und bei NONE ist die Nachricht hinter dem Datentyp zu Ende. Abbildung 5.4 zeigt die Liste mit den erweiterten Nachrichten.

Nachricht	Funktion	sender	type	datatype	data	Format (C-Syntax)
Richtung	Richtungsausschlag	raspberry1	direction	KEYVALUE	-1.00 bis 1.00	%1.2f
Referenz	Referenzsignal	raspberry1	reference	NONE	-	-
Geschwindigkeit	Geschwindigkeit in Meter pro Sekunde	ergometermodule	speed	KEYVALUE	-28.0 bis 28.0	%2.1f
Drehzahl	Umdrehungen pro Minute	ergometermodule	rpm	KEYVALUE	-200.00 bis 200.00	%3.2f
Belastung	Widerstand in Watt	any	resistance	KEYVALUE	20 bis 1000	%d
Minimale Belastung	Widerstand in Watt	any	resistanceMin	KEYVALUE	20 bis 1000	%d
Maximale Belastung	Widerstand in Watt	any	resistanceMax	KEYVALUE	20 bis 1000	%d
Steigung	Steigung in Grad	unity	slope	KEYVALUE	0.00 bis 360.00	%3.2f
Gangschaltung	Gänge schalten	raspberry2	gear	KEYVALUE	1 bis 28	%d
Bremse	Geschwindigkeit/Widerstand anpassen	raspberry2	break	KEYVALUE	0 bis x	%d
Reset	Level neu starten	controlcenter	reset	NONE	-	-
Load	Level laden	controlcenter	level	NONE	Level1-5	String
Messdaten	Ergebnis Slalomlevel	unity	result	MAP	coins (0 bis 100000) time (in msec) retries	%d %d %d
Level geladen	Level wurde geladen	unity	levelLoaded	MAP	-	-
Level gestartet	Startlinie wurde überschritten	unity	levelStarted	MAP	-	-
Level gestoppt	Ziellinie wurde überschritten	unity	levelFinished	MAP	-	-
Münze eingesammelt	Coin wurde erreicht	unity	coinCollected	KEYVALUE	0 bis 100000	%d
...	weitere Unity Funktionen	unity	...	...	...	...

Abbildung 5.4.: Liste mit den erweiterten Nachrichten

5.5. Implementierung Ergometer Modul

Das Ergometer Modul ist die Schnittstelle zwischen dem Ergometer und dem Messagebroker. Es übersetzt Nachrichten zwischen dem EmoBike Nachrichtenprotokoll und dem Ergometer-

protokoll. Das ist notwendig damit andere EmoBike Komponenten über Netzwerknachrichten im EmoBike Protokollformat mit dem Ergometer kommunizieren können. So lassen sich gezielt Parameter auf dem Ergometer setzen oder Statusinformationen abfragen.

Das Format der EmoBike Nachrichten wurde schon beschrieben. Beim Eintreffen neuer Nachrichten sorgt ein Parser dafür, dass die notwendigen Informationen wie der Typ der Nachricht und der Datenwert extrahiert werden. Aus diesen Informationen kann dann eine Nachricht im proprietären Protokollformat von Daum Electronic generiert werden. Für das Protokollformat gibt es vom Hersteller ein Datenblatt [Daum Electronic GmbH (2005)], das den Aufbau beschreibt und eine Liste der möglichen Befehle enthält. Das Protokoll unterstützt sowohl serielle als auch TCP/IP Verbindungen. Bei letzterem werden die seriellen Datenpakete in TCP-Pakete eingebettet. Das hat zur Folge, dass bei der Ansteuerung über Netzwerk zunächst die seriellen Datenpakete zusammengebaut und anschließend in TCP-Pakete verpackt werden müssen.

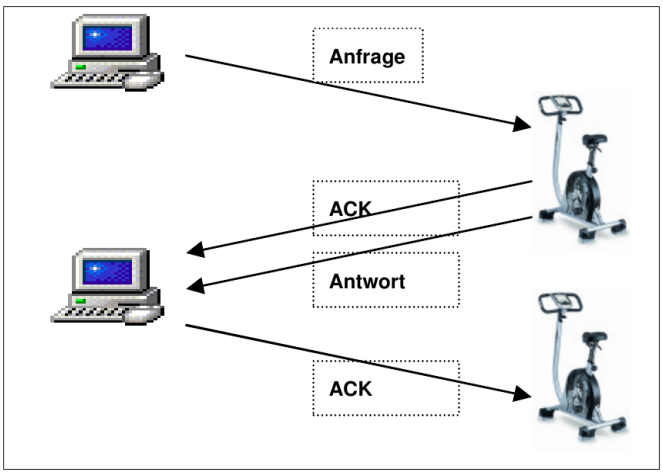


Abbildung 5.5.: Synchrone Kommunikation zwischen Ergometer und Netzwerkclient (Grafik: Daum Electronic GmbH (2005))

Es handelt sich um ein synchrones Protokoll, was bedeutet, dass jedes Datenpaket mit einem ACK-Paket (Acknowledge), bzw. im Falle einer falschen Prüfsumme mit einem NAK (No Acknowledge), quittiert werden muss. In letzterem Fall wird das Datenpaket erneut gesendet. Trifft nach einem Timeout von 10 Sekunden kein ACK/NAK ein wird ein Neuversand des Datenpakets initiiert. Nach dem fünften Versuch wird das Senden abgebrochen (siehe Abbildung 5.5).

5.5.1. Aufbau der Datenpakete

Die Datenpakete (Abbildung 5.6) bestehen aus ASCII-Zeichen (American Standard Code for Information Interchange. Zeichensatz mit 7-Bit-Kodierung). Das erste und das letzte Zeichen ist in jedem Datenpaket identisch, sie makieren Anfang und Ende. SOH steht für „Start Of Header“ und ETB steht für „End of Transmission Block“. Der auf das Zeichen SOH folgende Header ist immer exakt drei Zeichen lang und besteht aus einem Buchstaben und zwei Ziffern. Der Header gibt den Typ der Nachricht an. Die aktuelle Drehzahl wird beispielsweise mit „S21“ abgefragt. Inhalt und Länge der Dateneinheit hängen vom Typ der Nachricht ab. Die letzten zwei Zeichen vor ETB enthalten eine Checksumme, die die Dezimalkodierung aller Zeichen aus dem Header und dem Dateneinheit aufaddiert und darauf modulo 100 anwendet. Durch die Überprüfung der Checksumme kann der Empfänger Übertragungsfehler erkennen und ein erneutes Senden des Pakets veranlassen.

SOH	Header			Dateneinheit			Prüfsumme		ETB
1	2	3	4	5	…	n	n+1	n+2	n+3
0x01	s1	z1	z2	Daten			(Prüfsumme)		0x17

Abbildung 5.6.: Serielles Datenpaket des proprietären Ergometerprotokolls (Grafik: Daum Electronic GmbH (2005))

Es gibt drei Arten von Datenpaketen: Anfragen, Befehle und Antworten. Bei einer Anfrage ist der Datenbereich leer. Der Typ der Nachricht gibt an welche Daten vom Ergometer zurück gemeldet werden sollen. Diese Daten stehen im Datenbereich der vom Ergometer gesendeten Antwortnachricht. Das Format unterscheidet sich dabei je nach Anfrage und muss dem Datenblatt entnommen werden. Das Beispiel von oben sendet „S21“ mit einem leeren Datenbereich. Laut Datenblatt enthält die Antwort eine Fließkommazahl mit drei Vorkommastellen und einer Nachkommastelle. Zusammen mit dem Komma ist der Datenbereich fünf Zeichen lang.

Zum Setzen eines Parameters auf dem Ergometer dient das Befehlsdatenpaket. Der zu setzende Wert wird im Datenbereich angegeben. Das Ergometer antwortet auf jedes Befehlsdatenpaket mit einem Antwortdatenpaket. Dieses enthält bei erfolgreicher Verarbeitung den gleichen Wert wie das Befehlsdatenpaket. Sollte der angeforderte Wert nicht einstellbar sein wird der nächstmögliche Wert eingestellt und im Antwortpaket gesendet. Falls kein nächstmöglicher Wert ermittelt werden kann enthält das Antwortpaket keinen Datenbereich.

5.5.2. Funktionsumfang

Ursprünglich sollten benötigte Informationen wie Geschwindigkeit, Drehzahl, Belastung (Tretwiderstand) und Gang in regelmäßigen Intervallen beim Ergometer nachgefragt und über den Messagebroker weitergeleitet werden. Bei Testläufen zeigte sich jedoch, dass die Antworten des Ergometers im Bereich von zehntel Sekunden eintreffen. Bei parallelen Anfragen schaukelt sich die Verzögerung in einem selbstverstärkenden Effekt schnell so weit hoch, dass keine Antworten mehr ankommen und das Ergometer nicht mehr benutzbar ist. Eine Anfrage beim Hersteller brachte keine Lösung für das Problem, so dass die Anzahl der Abrufe reduziert werden musste. Übrig blieben das Abfragen der Geschwindigkeit und das Setzen der Belastung. Die Geschwindigkeit wird in einem Thread alle 50 ms abgefragt und an das Game weitergeleitet, das daraufhin die Geschwindigkeit des Bikes in der digitalen Welt anpasst. Tests haben gezeigt, dass 50 ms ein guter Wert ist, der das Ergometer nicht überfordert, ein schnelles Feedback über das Display gibt und ein mit dem Fahrradfahren vergleichbares realistisches Reaktionsverhalten aufweist. Es können parallel noch Befehlsnachrichten an das Ergometer gesendet und dort verarbeitet werden. Es werden folgende Funktionen unterstützt:

- Regelmäßiges Abfragen der Geschwindigkeit und Senden an das EmoBike Game
- Empfangen einer Steigung, berechnen einer neuen Belastung und diese am Ergometer setzen
- Empfangen einer Belastung und diese am Ergometer setzen

Mit der Steigung kann die virtuelle Umgebung des EmoBike Games (Abbildung 5.7) Einfluss auf den Tretwiderstand des Ergometers nehmen und ein vom Fahrradfahren erwartetes Verhalten erzeugen. Der Steigungswert wird vom Game über den Messagebroker an das Ergometermodul gesendet, das aus diesem Wert und der aktuellen Belastung einen neuen Belastungswert berechnet und diesen am Ergometer setzt. Der Algorithmus zur Berechnung des neuen Widerstands ist einfach gehalten. Die Steigung kann Werte von 0 bis 359 Grad annehmen wobei der Bereich von 0 bis 179 Grad ein positive Steigung repräsentiert und der Bereich von 359 bis 180 Grad eine negative Steigung. Jedes Grad verändert die Belastung um 10 Watt. Beispiel: Aktuelle Belastung 150 Watt, Änderung der Steigung von 0 auf 3 Grad. Die neue Belastung beträgt:

$$150\ \text{Watt} + (3 * 10\ \text{Watt}) = \underline{180\ \text{Watt}}$$

Physikalisch ist diese Berechnung nicht korrekt. Weder die Gewichtskraft, die sich am Hang in Hangabtriebskraft und Normalkraft aufteilt, noch Haft- und Gleitreibung werden hier berücksichtigt. Als Reaktion auf eine Steigungsänderung reicht dieses Verfahren im ersten

Schritt dennoch aus. Einen Hang hinaufzufahren ist anstrengender als in der Ebene zu fahren. Das simuliert ein realistisches Verhalten wie man es erwartet.

Abbildung 5.7.: Level mit Steigung

Langfristig ist geplant diese Logik in ein Physikmodul auszulagern und um weitere Funktionen zu ergänzen. Das Ziel ist es den Datenaustausch mit dem Ergometer so gering wie möglich zu halten. Im Idealfall beschränkt sich dieser auf das Abrufen der Drehzahl und das Setzen der Belastung. In dem Modul können dann auch weitere Parameter berücksichtigt werden, so dass sich die Geschwindigkeit beispielsweise aus der Drehzahl und dem aktuellen Gang ergibt. Ebenso könnte die Belastung aus Anfangsbelastung, Steigung, Gang und Untergrund berechnet werden. Mit diesem Ansatz ist eine spätere Erweiterung um zusätzliche Parameter gegeben und man kann das träge Antwortverhalten des Ergometers umgehen.

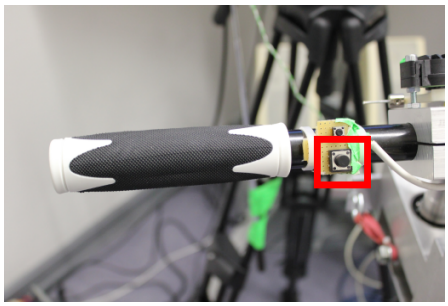
5.6. Implementierung Lenker Modul

Die Voraussetzung zum Lenken ist die Erweiterung des Ergometers um eine entsprechende Vorrichtung, die die Lenkbewegungen erkennt und an das EmoBike Game weiterleitet. Die Entwicklung verlief in zwei Phasen. Zunächst wurde der vorhandene starre Lenker an den beiden Außenseiten um Sensoren erweitert. Zum Vergleich wurden sowohl digitale Taster

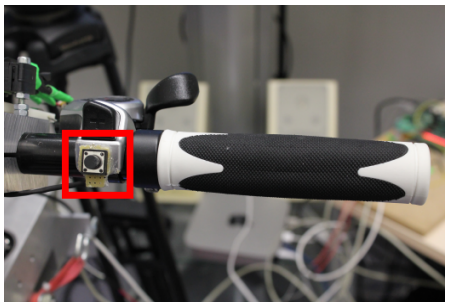
als auch analoge Drucksensoren montiert. Die Taster lassen sich mit dem Daumen betätigen wohingegen die Drucksensoren auf dem Griff befestigt sind und einen Druck auf den Lenker registrieren. In Phase zwei wurde der starre Lenker durch ein drehbares Modell ersetzt. Über einen am unteren Ende der Lenkstange befestigten Inkrementalgeber wird die Drehbewegung erfasst. Alle Sensoren sind an einen Raspberry Pi angeschlossen der die Signale abnimmt, einen Richtungswert ermittelt und diesen über den Messagebroker an das EmoBike Game sendet. Die folgenden Abschnitte beschreiben die unterschiedlichen Implementierungen und gehen auf die Vor- und Nachteile ein.

5.6.1. Version 1: Lenker mit Tastern

An jeder Lenkerseite ist neben dem Griff ein Taster (Abbildungen 5.8a und 5.8b) montiert, der bei Betätigung einen Lenkausschlag in die jeweilige Richtung auslöst. Die Konvention mit dem EmoBike Game besagt, dass der Wertebereich von -1.0 bis +1.0 reicht. Der Wert 0 entspricht der Mittelstellung, -1.0 bedeutet einen Maximalauschlag nach links und +1.0 einen Maximalausschlag nach rechts. Der Maximalauschlag beträgt 30 Grad von der Mittelstellung. Die Zwischenschritte sind auf drei Nachkommastellen begrenzt. Digitale Taster kennen nur zwei Zustände: gedrückt und nicht gedrückt. Entsprechend wird ein betätigter Taster als Vollausschlag gewertet, Zwischenschritte werden hier nicht generiert. Ein Halten des Tasters bewirkt ein Halten des Ausschlags. Die digitalen Taster wurden versuchsweise durch analoge



(a) Linker Taster



(b) Rechter Taster

Drucksensoren ersetzt um eine feinere Lenkung durch zusätzliche Signalzustände zu erreichen. Abhängig vom Druck auf die Sensoren sollte die Lenkung ausschlagen. Durch die sehr flache Beschaffenheit der Sensoren war es jedoch nicht möglich den Druck kontrolliert auszuüben. Eine Steuerung des virtuellen Fahrrads war darüber nicht möglich weshalb die Idee verworfen und die Entwicklung der drehbaren Lenkvorrichtung vorangetrieben wurde.

5.6.2. Version 2: Lenker mit Inkrementalgeber

Die Messung der Drehbewegung des Lenkers erfolgt über einen Inkrementalgeber (Kübler 2400), auch Drehgeber genannt. Dieser sitzt am unteren Ende der Lenkstange und liefert zwei Signale über die sich die Drehrichtung bestimmen lässt. Die beiden Signale A und B sind gegeneinander um 90 Grad phasenverschoben und erzeugen eine Kennlinie wie in Abbildung 5.9. Die Phasen a bis d sind im Gray-Code kodiert, d.h. dass sich bei jeder Änderung des Zustands nur ein Bit ändert (siehe Tabelle 5.1). Durch die Überwachung der Flanken und die Speicherung des letzten Zustands lässt sich die Drehrichtung ermitteln.

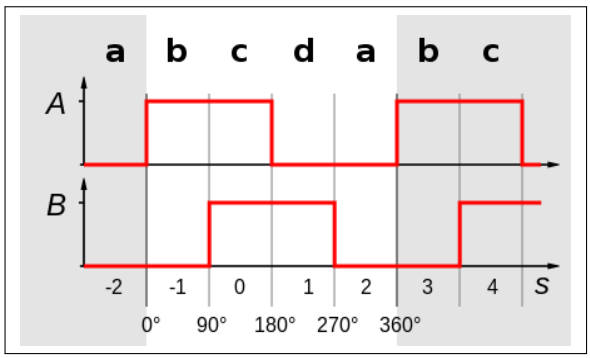


Abbildung 5.9.: Kennlinie der beiden Signale A und B eines Inkrementalgebers und den Phasen a bis d (Originalgrafik: Wikipedia)

Beispiel:
Der Drehgeber befindet sich in Phase b. Das Programm merkt sich diese Phase als den letzten Zustand. Signal A hat den Wert 1 und Signal B hat den Wert 0. Tritt jetzt eine steigende Flanke an Signal B auf wechselt der aktuelle Zustand zu c (A=1, B=1). Definiert man vorwärts laufende Zustandswechsel (hier b zu c) als Drehung nach Rechts wurde soeben eine Rechtsdrehung erkannt. Tritt hingegen an Signal A eine fallende Flanke auf wechselt der aktuelle Zustand zu a (A=0, B=0). Es wurde ein rückwärts laufender Zustandswechsel b zu a erkannt und somit eine Drehung nach Links.

Die Erkennung aller Signaländerungen bzw. Zustandswechsel spielt eine zentrale Rolle. Der eingesetzte Inkrementalgeber erzeugt bei einer vollen Umdrehung 4096 Zustandswechsel. Der Drehbereich des Lenkers liegt bei 60° vom linken Maximalauschlag bis zum rechten Maximalauschlag. Mit einer schnellen Drehbewegung über den vollen Drehbereich in einer Zeit von ca. 100 ms (geschätzt) ergibt sich eine Anzahl von 6827 Zustandswechsel pro Sekunde bzw.

A	B	Phase
0	0	a
1	0	b
1	1	c
0	1	d

Tabelle 5.1.: Signaltabelle im Graycode und zugehörige Phasen

6,827 Wechsel pro Millisekunde (4096 / (360°/60°) * (1000ms/100ms) = 6826,67). Diese Flut an Signalen muss verlustfrei empfangen und verarbeitet werden. Andernfalls kann das Programm die Richtungsänderung nicht bestimmen. Wenn beispielsweise der letzte gespeicherte Zustand Phase a war und der neu erkannte Zustand ist Phase c ist es nicht möglich zu bestimmen ob Phase c über Phase b oder über Phase d erreicht wurde, also eine Drehbewegung nach links oder nach rechts vollführt wurde. Als Konsequenz wird dieser unbestimmte Zustand verworfen und das Programm verliert seine Synchronizität mit dem Lenker. Bei wenigen Verlusten ist diese Verschiebung kaum wahrnehmbar und lässt sich über das sognannte Nullsignal korrigieren. Das Nullsignal wird vom Inkrementalgeber genau ein mal pro Umdrehung gesendet. Justiert man dieses Signal auf die Mittelstellung des Lenkers und kalibriert es beim ersten Übertreten kann danach die Lenkrichtung bei jedem weiteren Übertreten zurückgesetzt werden.

Mit der eingesetzten WiringPi Library [Henderson (2014b)] können GPIO Pins an Interrupts gebunden werden. Interrupts sind ein geeignetes Mittel um asynchron auftretende Ereignisse zu überwachen. Die Interrupts lassen sich so konfigurieren, dass sie auf fallende, steigende oder beide Flankenwechsel an den GPIO Pins reagieren. Die Signale A und B sind jeweils an einen eigenen Pin am Raspberry Pi angeschlossen und lösen bei jedem Flankenwechsel einen Interrupt aus. Interrupts werden in der Interrupt Service Routine (ISR) abgearbeitet. Die ISR liest die Signalpegel aus und ermittelt die aktuelle Phase. Zusammen mit der zwischengespeicherten Phase aus dem letzten Aufruf ergibt sich die Drehrichtung.

In den Praxistests zeigten sich starke Perfomanceprobleme. Während bei langsamen Lenkbewegungen die Steuerung korrekt funktioniert, kommt es bei schnellen oder ruckartigen Bewegungen zu vielen Signalverlusten. Schnelle Bewegungen erzeugen im Inkrementalgeber eine große Anzahl von Signaländerungen von der jede einzelne auf dem Raspberry Pi einen Interrupt auslöst. Interrupts können nicht gepuffert werden. Befindet sich ein Interrupt zur Bearbeitung in der ISR kann maximal ein weiterer Interrupt [Henderson (2014a)] erkannt werden, der abgearbeitet wird sobald die laufende ISR beendet wurde. Alle weiteren in diesem

Zeitraum auftretenden Interrupts gehen verloren, was zu zwei Problemen führt. Zum einen gehen Signale verloren was bedeutet, dass der Lenker seine Stellung ändert ohne dass diese Informationen registriert werden. Die Synchronizität zwischen dem Lenker am Ergometer und dem virtuellen Lenker im EmoBike Game geht verloren (dies bezeichnet man auch als „driften“). Zum anderen können ungültige Phasenwechsel auftreten, die verworfen werden müssen, da keine Richtung bestimmt werden kann (siehe Beispiel oben). Dies führt ebenfalls zu einem Synchronizitätsverlust.

Neben der geringen CPU-Leistung des Raspberry Pi ist hauptsächlich die im Userspace ausgeführte ISR für die langsame Interruptverarbeitung verantwortlich. Die vielen notwendigen Kontextwechsel[1] zwischen Userspace und Kernelspace zum Erkennen eines Interrupts, der Bestätigung des Interrupts und der Abarbeitung der ISR kosten viel Zeit. Nach einiger Recherche war klar, dass andere GPIO Libraries keine nennenswerten Geschwindigkeitsvorteile bringen würden. Die Verlagerung der Interruptverarbeitung direkt in den Kernelspace [Hamlin (2013)] dagegen kann im Idealfall die Laufzeit der ISR auf 1/10 der ursprünglichen Laufzeit reduzieren. Notwendig dafür ist ein Linux Kernelmodul zur Signalauswertung und ein Userspace Programm, das die Informationen über das Netzwerk an den ActiveMQ sendet. Die Entwicklung eines Kernelmoduls ist jedoch nicht trivial und zeitintensiv, insbesondere weil ein Fehler im Kernelmodul zum Absturz des Betriebssystems führt. Weil die Lenkung durch den starken Drift aber nicht zu benutzen war und Optimierungen am Userspace Programm nicht die notwendige Performanceverbesserung versprachen, fiel die Entscheidung die Zeit zur Entwicklung des Kernelmoduls aufzuwenden.

5.6.3. Entwicklung des Kernelmoduls

Für die Entwicklung eines Kernelmoduls werden die nicht im Standardimage enthaltenen Kernel Header benötigt. Sie müssen exakt zu dem auf dem Raspberry Pi laufendem Kernel Image passen. Leider sind die über die Paketverwaltung nachinstallierbaren Kernel Header nicht immer auf dem aktuellen Stand. Das ist auch der Fall bei der hier verwendenten Version 2014-06-20 der Raspberry Pi Linux Distribution Raspbian. Der installierte Kernel trägt die Versionsnummer 3.12.22+ wohingegen die Header noch die Versionsnummer 3.12.9 tragen. Das hat zur Folge, dass ein neues Kernel Image mit den passenden Headern kompiliert werden muss. Wie in Abschnitt 5.3.2 erwähnt, ist das Kompilieren von Quellcode auf dem Raspberry Pi im Vergleich zu anderen Computern langsam, weshalb auch an dieser Stelle eine Crosscompile-

[1] Bei einem Kontextwechsel wird der Registerinhalt des aktuell bearbeiteten Programms gesichert, zeitweise ein anderes Programm ausgeführt und anschließend der Registerinhalt wieder zurückgespielt [vgl. Tanenbaum und Bos (2014)].

Umgebung eingerichtet wird. Damit lassen sich sowohl der Kernel als auch das Kernelmodul auf einem schnellen Desktopsystem kompilieren und anschließend auf dem Raspberry Pi installieren. Die Einrichtung erfolgte nach dieser Anleitung [eLinux.org (2015)]. Die verwendete Kernelversion lautet 3.12.28+.

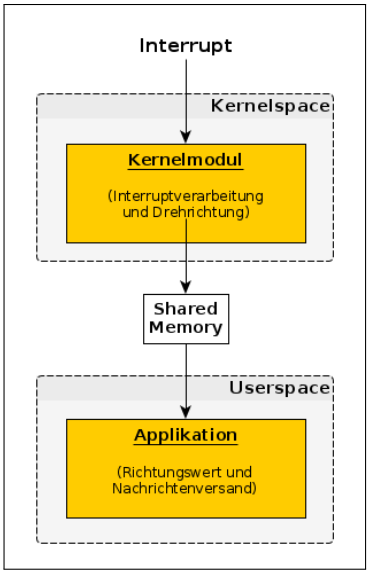


Abbildung 5.10.: Aufgabenverteilung Lenkermodul

Das neue Lenkermodul ist aufgeteilt auf zwei Programme: das Kernelmodul und die Userspace Applikation. Die Aufgabenverteilung ist in Abbildung 5.10 dargestellt. Das Kernelmodul verarbeitet die Signale vom Inkrementalgeber. Wie zuvor werden Interrupts mit den GPIO Pins verknüpft und beim Auslösen der Interrupts die ISR aufgerufen, aber diesmal direkt im Kernelkontext. Die ISR ermittelt über die aktuelle und die zuletzt gespeicherte Phase die Drehrichtung und berechnet einen Richtungswert. Dieser wird als Ganzzahl im gemeinsamen Speicher abgelegt. Der gemeinsame Speicher (Shared Memory) dient zum Datenaustausch mit der Userspace Applikation. Hintergrund für die Speicherung als Ganzzahl ist der aufwendige Umgang mit Fließkommazahlen im Kernelspace. Die Folge ist, dass die Userspace Applikation die Ganzzahl in eine Fließkommazahl umrechnen muss bevor sie an den ActiveMQ übergeben wird.

Der aktuelle verwendete Lenkbereich beträgt 60 Grad. Der Inkrementalgeber erzeugt 4096 Zustandswechsel bei einer vollen Umdrehung. Der Wertebereich für die Maximalausschläge liegt bei -1.0 bis +1.00. Daraus ergibt sich für jeden Zustandswechsel eine Richtungsänderung von ca. 0.003 ((|-1.0| + 1.0) / (4096 / (360°/60°)) = 0.0029). Wenn jetzt jede dieser Änderungen

direkt an den ActiveMQ gesendet werden würde ergäbe das für den vollen Lenkbereich eine Anzahl von ca. 680 Nachrichten. Was bei langsamen Drehbewegungen kein Problem darstellt, kann bei schnellen Drehbewegungen zu einer Nachrichtenflut auf dem ActiveMQ führen. Deshalb wird nicht jede Richtungsänderung versendet, sondern ein Timer verwendet, der alle 10 ms den aktuellen Richtungswert an das EmoBike Game sendet. Das erzeugt eine zwar eine permanente Last auf dem ActiveMQ, verhindert aber eine Überschwemmung mit Nachrichten.

In den Tests mit dem verbesserten Lenkermodul zeigte sich, dass die Interruptverluste gegen Null gehen. Über einen Zeitraum von einigen Minuten mit vielen hektischen Lenkbewegungen konnten nur zwei verlorene Interrupts festgestellt werden. Ein derart geringer Drift ist zu vernachlässigen. Allerdings erhöht sich der Drift proportional zur Zeit, weshalb sich die Abweichung über einen großen Zeitraum hinweg bemerkbar machen kann. Mit dem bereits erwähnten Nullsignal lässt sich dieses Verhalten jedoch korrigieren. Das Problem der Signalverluste wurde mit der Implementierung des Kernelmoduls gelöst.

5.6.4. Latenzmessungen

Ein weiteres Problem betrifft die Verzögerung (Latenz) vom Ausführen einer Lenkbewegung bis zur Darstellung auf dem Bildschirm. Gerade bei schnellen Bewegungen ist erkennbar, dass der Lenker im EmoBike Game die Zielstellung erst mit einiger Verzögerung erreicht. Für den Menschen ist eine Verzögerung ab 100 ms wahrnehmbar [vgl. Nielsen (1993), zit. in: Bernin (2011)], so dass die Reaktionszeit beim Lenken darunter liegen muss. Der ActiveMQ als Ursache war naheliegend da das Publisher/Subscriber Konzept keine synchrone Datenverbindung zwischen Raspberry Pi und EmoBike Game bietet. Stattdessen ist das Game darauf angewiesen vom ActiveMQ zeitnah über neue Nachrichten informiert zu werden wodurch ein Overhead entsteht. Messungen ergaben jedoch, dass die Nachrichtenlaufzeit über den ActiveMQ im Durchschnitt nur 7.3 ms beträgt (siehe Abbildung 5.11). Das liegt weit unter der Grenze von 100 ms. Gemessen wurde der Zeitraum, den eine Nachricht benötigt, um vom Raspberry Pi an den ActiveMQ gesendet und von dort auf dem selben Raspberry Pi empfangen zu werden. In der Dokumentation zum ActiveMQ konnten keine Hinweise darauf gefunden werden, dass ein lokaler Zwischenspeicher die Nachrichtenzustellung beschleunigt, wenn Sender und Empfänger auf dem gleichen System laufen. Die Messwerte können daher als realistisch angesehen werden.

Für die weitere Analyse erschien es sinnvoll zunächst die Gesamtlatenz vom Versand einer Nachricht vom Raspberry Pi bis zu einer Reaktion auf dem Display zu messen. Um präzise Ergebnisse zu erhalten musste der Systemaufbau so angepasst werden, dass die Latenzen auf

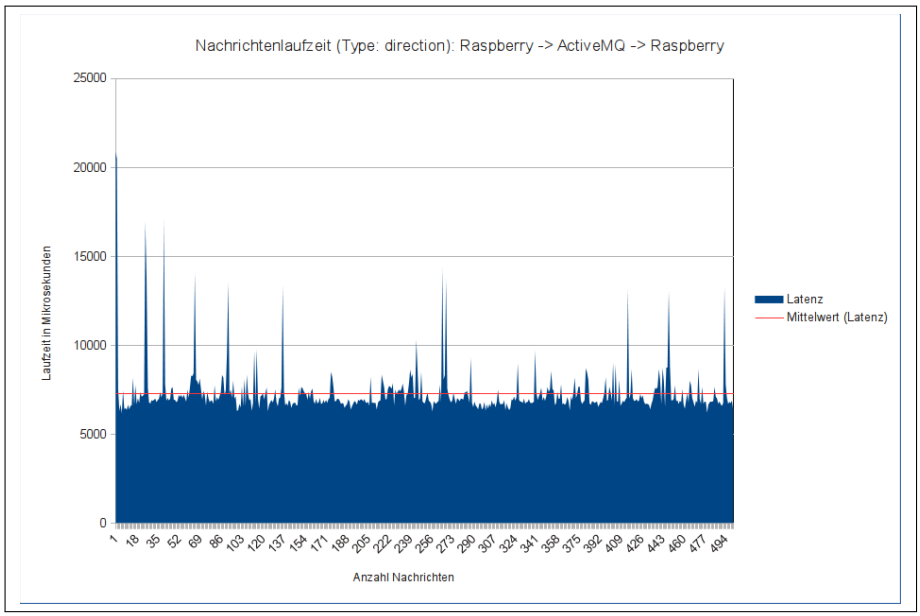


Abbildung 5.11.: Messung der ActiveMQ Nachrichtenlaufzeit

einem einzigen System gemessen werden können. Das EmoBike Game wurde zuvor um einen speziell präparierten Level erweitert, der am oberen Bildschirmrand einen schwarzen Kasten anzeigt. Beim Eintreffen einer festgelegten Nachricht wechselt die Farbe des Kastens zu weiß. Ein am Bildschirm befestigter Fototransistor erkennt den Farbwechsel und schaltet durch. Der Sensor ist mit dem Raspberry Pi verbunden, der das Signal erfasst und die Differenz zwischen dem Versand der Nachricht und der Reaktion des Fototransistors bildet. Die Ergebnisse zeigt die Abbildung 5.12.

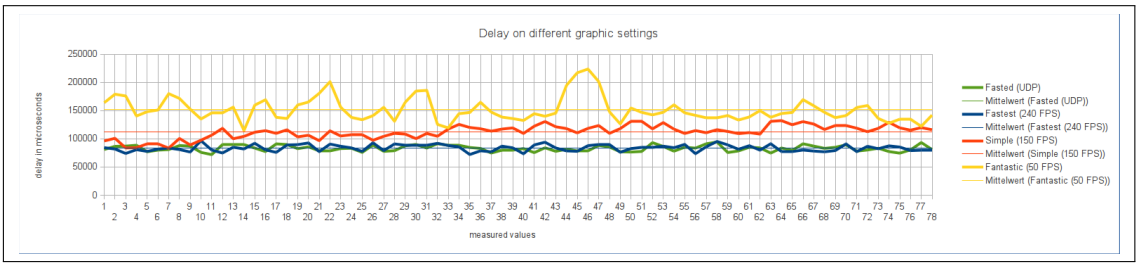


Abbildung 5.12.: Messung der Nachrichtenlatenz vom Versand bis zur Darstellung

Die Messungen wurden mit unterschiedlichen Grafikdetailstufen durchgeführt, die einen direkten Einfluss auf die Anzahl der Frames pro Sekunde (FPS) haben. Mit der Einstellung

„Fastest“ liegt die Latenz bei ungefähr 80 ms. Wird die Detailstufe angehoben auf „Simple“ erhöht sich die Latenz auf über 110 ms und mit der maximalen Detailstufe „Fantastic“ liegt die Latenz bei 150 ms. Die grüne Kurve zeigt die Latenz wenn die Richtungsdaten über eine direkte UDP-Verbindung zwischen Raspberry Pi und EmoBike Game versendet werden. Diese Kommunikationsverbindung wurde zwischenzeitlich implementiert um den ActiveMQ als Ursache definitiv auszuschließen. Die nahezu identischen Ergebnisse mit der Einstellung „Fastest“ bestätigen die Annahme, dass der ActiveMQ nicht für das Latenzproblem verantwortlich ist.

Die Ergebnisse bestätigen außerdem die subjektive Wahrnehmung beim Abfahren des präparierten Levels. Das Reaktionsverhalten des Lenkers wirkt auf den Probanden bei einer geringen Detailsstufe synchron und bei höheren Detailstufen asynchron. Dies passt auch zu der Theorie, dass eine Verzögerung ab 100 ms vom Menschen wahrgenommen wird. Die Frames pro Sekunde scheinenen hier der wichtige Faktor zu sein. Je höher die FPS desto besser das Lenkverhalten. Obwohl für das menschliche Auge bereits ein Wert von 25 FPS für eine flüssige Darstellung ausreicht scheint das für den Lenker nicht der Fall zu sein. Mit der Einstellung „Fastest“ werden im Testlevel durchschnittlich 240 FPS erreicht. Dieser Level enthält weniger Objekte als die regulären Level, die mit den gleichen Einstellungen weniger FPS schaffen. Die Option, für alle Level die niedrigste Detailstufe zu wählen, um ein gutes Lenkverhalten zu erreichen, schließt sich daher aus. Stattdessen muss die Anzahl der FPS dauerhaft in allen Levels erhöht werden. Mit dem Austausch der vorhandenen Grafikkarte durch ein schnelleres Modell konnte die Anzahl der FPS signifikant erhöht und die Problematik des verzögerten Lenkverhaltens vorerst beseitigt werden.

5.7. Implementierung ControlCenter

Das ControlCenter ist das Operator Frontend für das EmoBike und dient der Überwachung und Konfiguration des Gesamtsystems (siehe Abbildung 5.13). Für die Überwachung muss es sich als Subscriber für die vorhandenen Topics registrieren. Anschließend kann es alle über den Messagebroker ausgetauschten Nachrichten mitlesen und anzeigen. Das ist notwendig damit die selbstgeneriereten EmoBike Nachrichten auf Fehler hin untersucht werden können. Neben Format- oder Rechtschreibfehlern innerhalb der Textnachrichten sind auch Nachrichtenabfolgen interessant. Wird durch ein bestimmtes Paket eine Anfrage getätigt sollte zeitnah ein entsprechendes Antwortpaket kommen. Durch die chronologisch korrekte Anzeige der Nachrichten lassen sich solche Verbindungen nachvollziehen und gegebenenfalls ausbleibende oder falsch adressierte Nachrichten identifzieren.

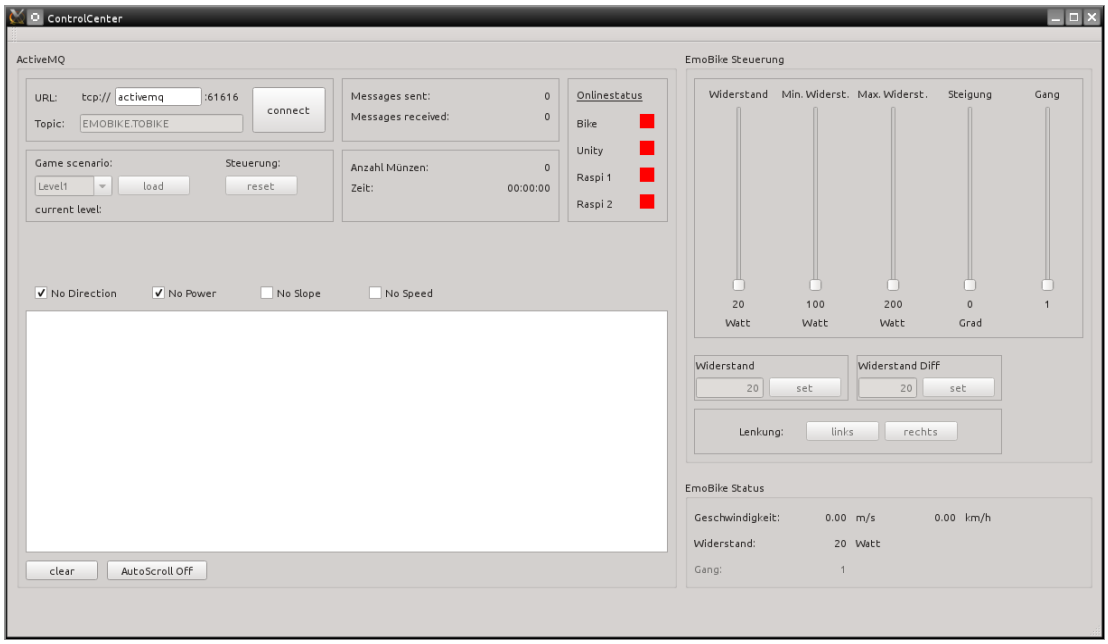


Abbildung 5.13.: Das ControlCenter

Über Checkboxen lassen sich Filter setzen, um die Anzeige der Nachrichten zu reduzieren (siehe Abbildung 5.14). Insbesondere die Nachrichten mit der Lenkrichtung kommen mit einer hohen Frequenz von 100 Hz (alle 10 ms) und füllen dadurch das Ausgabefenster so schnell, dass die Meldungen nicht mehr gelesen werden können.

Abbildung 5.14.: Checkboxen zum Filtern von Nachrichten

Zur Konfiguration und Steuerung muss sich das ControlCenter zusätzlich als Publisher beim Messagebroker registrieren. Danach lassen sich Steuernachrichten wie „Level laden“, „Widerstand setzen“, „Steigung setzen“, „Gang setzen“ oder „Geschwindigkeit abfragen“ senden. Einige dieser Funktionen dienen nur der Simulation bestimmten Verhaltens. Zum Beispiel lässt sich mit dem Verändern der Steigung testen ob der Widerstand korrekt angepasst wird. Interessant ist auch das Verhalten an den Grenzen, die in den Game Levels nicht vorkommen, also die maximale Belastung oder eine extrem große Steigung.

Mit den Reglern für den minimalen und maximalen Widerstand lässt sich der Widerstandsbereich begrenzen. Dies kann notwendig sein um eine minimale Belastung zu erreichen, die nicht unterschritten werden soll, auch wenn ein sehr niedriger Gang gesetzt wird. Die Festlegung einer maximalen Belastung verhindert, dass große Steigungen den Tretwiderstand zu stark erhöhen und das Ergometer nicht mehr bedienbar machen. Zusätzlich lässt sich dadurch die Belastung in einem konstanten Bereich halten.

Über ein Dropdown-Menü (Abbildung 5.15) lassen sich die einzelnen Levels laden. Im Experimentierstadium lassen sich damit dynamische Versuche durchführen, deren Levelabfolge zur Laufzeit angepasst werden können. Bei auftretenden Problemen kann das Level neu geladen werden. Diese Funktion ist ebenfalls hilfreich wenn aktiv an der Weiterentwicklung eines Levels gearbeitet wird und dieses Level zu Testzwecken häufig hintereinander geladen werden muss. Für spätere Versuche sollte der Ablauf zusätzlich automatisierbar sein damit die Versuchsreihen einheitlich sind. In zwei Bereichen werden Statusinformationen ausgegeben. Der

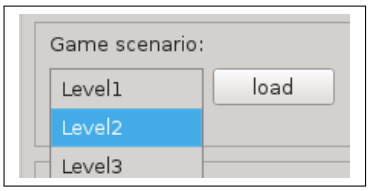


Abbildung 5.15.: Dropdown-Menü zum Laden eines Levels

Bereich für das Ergometer zeigt die aktuellen Werte für Geschwindigkeit und Widerstand an. Sie werden vom Ergometermodul versandt sobald sich ein Wert ändert. Da das ControlCenter alle über den Messagebroker laufenden Nachrichten mitliest können diese Informationen gefiltert aufbereitet werden. In den Nachrichten ist die Geschwindigkeit in m/s angegeben. Für ein besseres Verständnis wird in der GUI zusätzlich die Geschwindigkeit in km/h ausgegeben. Der aktuelle Widerstand ist in Watt angegeben.

Der zweite Bereich („Onlinestatus“) gibt Auskunft über die Netzwerkkonnektivität der wichtigsten EmoBike Systeme: Ergometer, EmoBike Game und Raspberry Pi. Das ControlCenter überprüft regelmäßig die Erreichbarkeit dieser Komponenten und gibt über grüne und rote Buttons ein visuelles Feedback. So lassen sich Probleme auf Netzwerkebene schnell identifizieren.

5.8. Evaluation

Die in der Analyse herausgearbeiteten Aufgaben (3.4) zum Aufbau einer Fahrradsimulation mit dem Ergometer als Eingabecontroller konnten alle umgesetzt und ein lauffähiger Prototyp entwickelt werden. Dazu wurde das Ergometer um einen drehbaren Lenker erweitert, dessen Bewegungen mit einem Inkrementalgeber gemessen werden. Die Signalauswertung des Inkrementalgebers erfolgt auf einem Raspberry Pi, der aus den Signalen einen Richtungswert bestimmt und diesen an das EmoBike Game sendet. Die während der Entwicklung aufgetretenen Probleme sowohl mit der Latenz als auch mit der Präzision konnten gelöst werden. Die Latenz überstieg zunächst die Grenze von 100 ms, die für den Menschen den Übergang der Wahrnehmung von Reaktionen zwischen Echtzeit und Verzögerung kennzeichnet. Nach umfangreichen Tests konnten sowohl die Netzwerkverbindung als auch das Lenker Modul als Ursache ausgeschlossen werden. Schnellere Grafikhardware für das EmoBike Game brachte die Lösung. Die Latenz konnte auf unter 100 ms reduziert und somit in den für den Menschen als Echtzeit empfundenen Bereich gebracht werden. Das zweite Problem betraf die Präzision. Die Lenkung ist dank des hochsensiblen Inkrementalgebers sehr weich und präzise, erzeugte aber bei schnellen Lenkbewegungen eine Flut von Signalen, die der Raspberry Pi nicht schnell genug verarbeiten konnte, was zu Signalverlust und damit zum Driften der Lenkung führte. Mit einer Änderung in der Implementierung des Lenker Moduls, die die Signalverabeitung vom Userspace in den Kernelspace verlagert, wurde dieses Verhalten korrigiert.

Das entwickelte Nachrichtenprotokoll ermöglicht zusammen mit einem Messagebroker die Kommunikation der EmoBike Komponenten miteinander. Mit dem Ergometer Modul wurde das Ergometer in das EmoBike System integriert und kann nun von dort gesteuert werden. Das Modul übersetzt die Nachrichten zwischen dem EmoBike Nachrichtenprotokoll und dem proprietären Herstellerprotokoll. Diese Schnittstelle ermöglicht das Setzen der aktuellen Geschwindigkeit im EmoBike Game und die Änderung des Tretwiderstand in Abhängigkeit von der virtuellen Umgebung.

Das ControlCenter bietet die Möglichkeit zur Steuerung und Konfiguration des Systems zur Laufzeit. Es kann Einfluss auf den Tretwiderstand, die Steigung und die Gänge nehmen sowie die Game Levels laden. Zusätzlich werden Informationen zum Zustand der EmoBike Komponenten angezeigt und alle über den Messagebroker vermittelten Nachrichten ausgegeben.

Erste Versuche mit der Fahrradsimulation haben gezeigt, dass sich die Probanden schnell mit der Steuerung des Ergometers zurecht finden. Die Beziehung zwischen der Interaktion mit dem Ergometer und der Reaktion auf dem Bildschirm wurde rasch hergestellt und nach einer

kurzen Eingewöhnungsphase im dafür entworfenen Einführungslevel (Abbildung 5.16) hatten die Probanden die Kontrolle über das System und konnten sich sicher durch die weiteren Level des EmoBike Games bewegen. Anhaltspunkte für Probleme mit der Steuerung gab es nicht, woraus man schließen kann, dass das mentale Modell eines Fahrrads von den Probanden auf das Ergometer übertragen wurde. Dies bestätigt ein natürliches Verhalten des Ergometers.

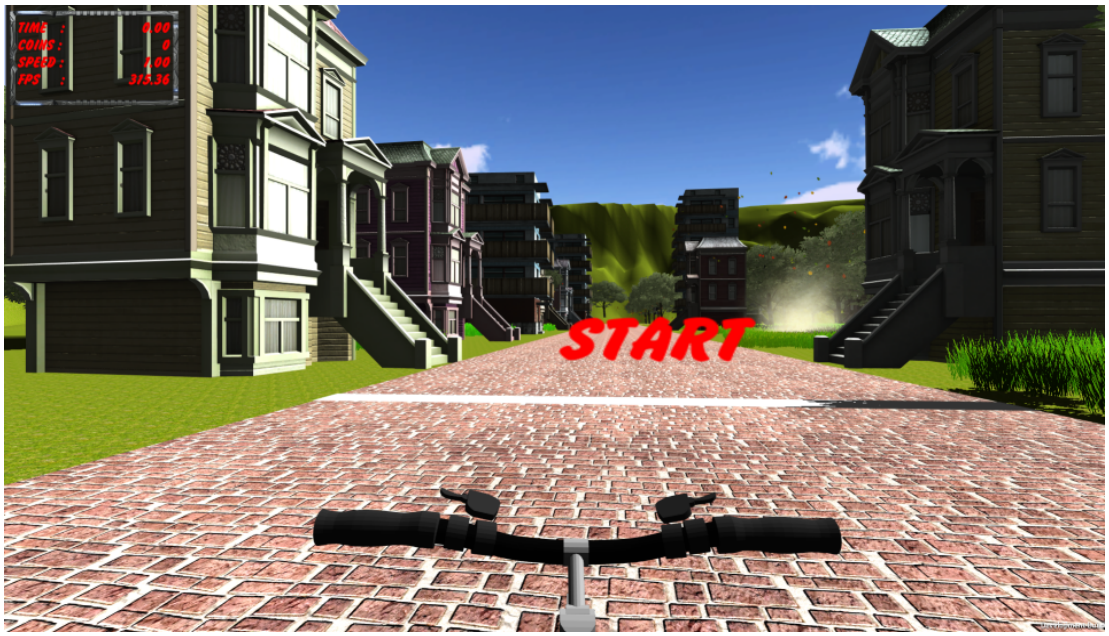


Abbildung 5.16.: Einführungslevel zur Gewöhnung an die Steuerung des Ergometers

Zusammengefasst konnte eine Fahrradsimulation mit dem Ergometer als natürlich zu bedienendem Controller aufgebaut werden. Die Implementierung einer Infrastruktur bestehend aus Messagebroker und EmoBike Nachrichtenprotokoll dient sowohl der Kommunikation zwischen Lenker Modul, EmoBike Game und Ergometer Modul als auch der Überwachung und Steuerung durch das ControlCenter. Sie bildet außerdem die Grundlage für zukünftige Erweiterungen. Geplant ist die Erweiterung des Lenkers um Bremse und Gangschaltung sowie die Entwicklung eines Physikmoduls zur Berechnung der Geschwindigkeit in Abhängigkeit von Umgebungsvariablen. Erste Versuche zeigen, dass sich die Probanden eine Seitwärtsneigung wünschen um die Lenkung weiter zu verbessern. Inwieweit das realisierbar ist muss in einem der nächsten Entwicklungsschritte untersucht werden, die aber nicht mehr Teil dieser Arbeit sind. Die Basis für die Erweiterung der Fahrradsimulation als Teil des EmoBike Projekts wurde geschaffen.

6. Fazit

6.1. Zusammenfassung

Im Rahmen des EmoBike Projekts wurde eine Fahrradsimulation mit dem Fahrradergometer als zentrales Eingabegerät entwickelt. Das Ziel war es das Benutzerinterface des Ergometers so zu erweitern, dass sowohl die Steuerung als auch das Feedback ein natürliches, von einem realen Fahrrad bekanntes Verhalten aufweisen. Mit der Entwicklung eines drehbaren Lenkers wurde eine zusätzliche Interaktionsmöglichkeit geschaffen, die mit dem Vorbild des Fahrrads kongruiert. Lenkbewegungen werden schnell und präzise verarbeitet und im EmoBike Game flüssig dargestellt, so dass die Steuerung einem realistischen Verhalten entspricht. Geschwindigkeit und Tretwiderstand lassen sich dank der entwickelten Schnittstelle zum Ergometer regulieren und geben dem Probanden sowohl optisches (schneller/langsamer Fahren im EmoBike Game) als auch haptisches (Tretwiderstand) Feedback. Die Integration eines lose gekoppelten Kommunikationssystems mit einem einheitlichen Nachrichtenprotokoll bildet die Grundlage zur strukturierten Datenübertragung innerhalb des verteilten EmoBike Systems und ermöglicht eine unabhängige Erweiterung um zusätzliche Komponenten. Erste Versuche zeigten, dass die Probanden sich schnell auf die Steuerung des Ergometers einstellen konnten. Die richtige Bedienung wurde intuitiv ohne Hilfestellung von außen erkannt, was das natürliche Verhalten des entwickelten Benutzerinterfaces bestätigt.

Eingangs wurde gesagt, dass es noch viele Bereiche gibt in denen die Interaktion zwischen Mensch und Computer verbessert werden kann. Dazu gehört die Fahrradsimulation, die nun über einen Controller verfügt, der viele Eigenschaften eines realen Fahrrads aufweist und dadurch eine Erwartungskonformität erreicht, die mit einem einfachen Gamecontroller nicht möglich ist. Die gesamte Simulation stellt sich dem Benutzer als ein einheitliches System dar, wobei das EmoBike Game die einzige Komponente ist, die klar als Computersystem identifizierbar ist. Dass dahinter ein verteiltes System aus vielen Computern mit unterschiedlichen Aufgaben steckt, bleibt dem Benutzer verborgen. Dies zeigt beispielhaft, wie sich Computer immer weiter in unsere Umgebung integrieren (Ubiquitous Computing) ohne dabei als solche

erkannt oder wahrgenommen zu werden (Disappeared Computing). Die Erweiterung des Ergometers um ein natürliches Interface realisiert dies für die Fahrradsimulation und verbessert dadurch das gefühlte Erleben des Fahrradfahrens deutlich.

6.2. Ausblick

Die Erweiterung des EmoBike Benutzerinterfaces hat gezeigt, dass Latenz und Präzision wichtige Faktoren für das realistische Verhalten einer Steuerung sind. Sie lassen sich auf andere Bereiche wie Fahrzeug- und Flugzeugsimulatoren übertragen, die ebenfalls Sensoren zur Integration von Bedienelementen einsetzen. Bei zukünftigen Entwicklungen muss untersucht werden welche weiteren Faktoren das Verhalten von natürlichen Steuerungen beeinflussen und wie man diese verbessern kann. Dies betrifft sowohl die technische Umsetzung als auch die Benutzbarkeit.

Die Herausforderung beim Entwickeln von computergestützten natürlichen Interfaces besteht aus der Abstraktion von Hardware und Interface bei gleichzeitigem Erhalt der Intuitivität. Als Beispiel sei eine Maschinensteuerung mit vielen mechanischen Knöpfen und Ventilen genannt. Zur Zentralisierung der Steuerung könnte man die einzelnen mechanischen Elemente durch computergesteuerte Aktoren ersetzen, die alle an einem zentralen Schaltpult zusammen laufen. An diesem Pult lassen sich die Aktoren einzeln auswählen und bedienen. Interessant ist die Repräsentation der Steuerung, die computertypisch mit Maus und Tastur erfolgen könnte. Eine Alternative dazu wäre die Entwicklung eines Interfaces, das der ursprünglichen Steuerung entspricht. Ein Controller bestehend aus einem Ventil und einer kleinen Anzahl von Knöpfen lässt sich durch das Computersystem zu jeder angebundenen Maschine verbinden. Anschließend lässt sich die ausgewählte Maschine mit dem Controller bedienen. An dieser Stelle wird eine Abstraktion zwischen Hardware und Benutzerschnittstelle geschaffen. Die Art der Bedienung bleibt erhalten und es wird ein natürliches Benutzerinterface geschaffen. Eine ähnliche Situation wird von Olaf Potratz [vgl. Potratz (2014)] beschrieben, der die Abstraktion jedoch in Form einer Gestensteuerung durchführt.

Die Natürlichkeit des Emobike Interfaces wurde mit dem drehbaren Lenker und der Steuerung des Tretwiderstands bereits deutlich verbessert. Es gibt aber noch viele Möglichkeiten das Interface zu erweitern. In Abschnitt 3.2 der Analse wurden bereits Bremse, Gangschaltung und eine Seitwärtsneigung aufgeführt. Abschnitt 3.3 nennt zusätzlich Fahrtwind und die Einbeziehung physikalischer Aspekte, wie das Ausrollen. Weitere Elemente wie ein Rütteln des Ergometers

bei unebenem Untergrund sind vorstellbar. Ein weiterer Ansatz wäre die Anpassung des Emo-Bike Games, das zu einer Simulation für das Verhalten im Straßenverkehr erweitert werden könnte, wodurch die Simulation als Ganzes an Natürlichkeit gewinnen würde. Denkbar wäre auch die Integration eines Kartendienstes, wie beispielsweise Google Maps, der in Kombination mit Google Street View vielleicht das Befahren realer Strecken ermöglicht.

Der wichtigste Aspekt beim Entwickeln von Natural User Interfaces wird auch zukünftig sein, zu verstehen was die Natürlichkeit eines Interfaces ausmacht, welches Verhalten erwartet wird und wie man diese beiden Faktoren in einer intuitiven Steuerung miteinander verbinden kann.

A. Anhang

A.1. EmoBike Fotos

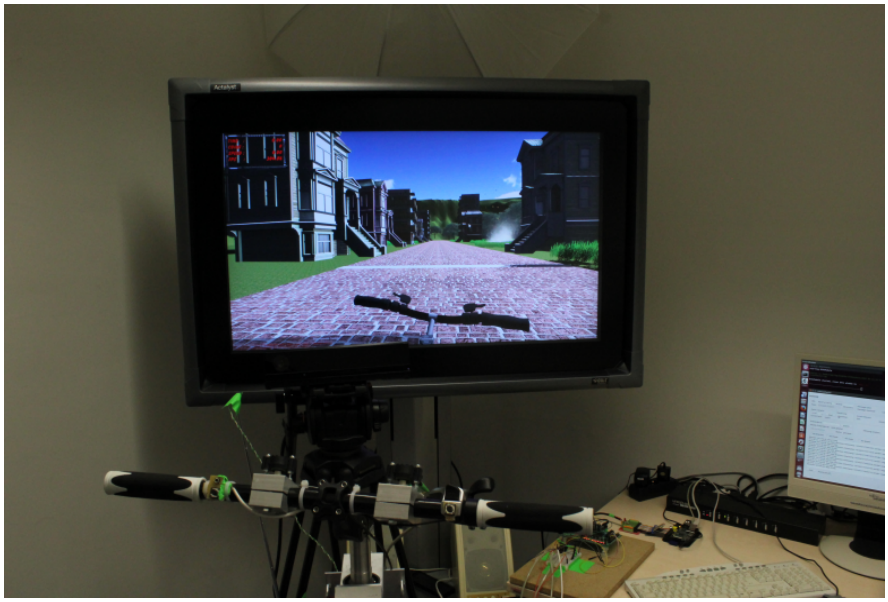


Abbildung A.1.: EmoBike Aufbau mit Lenkeinschlag nach rechts

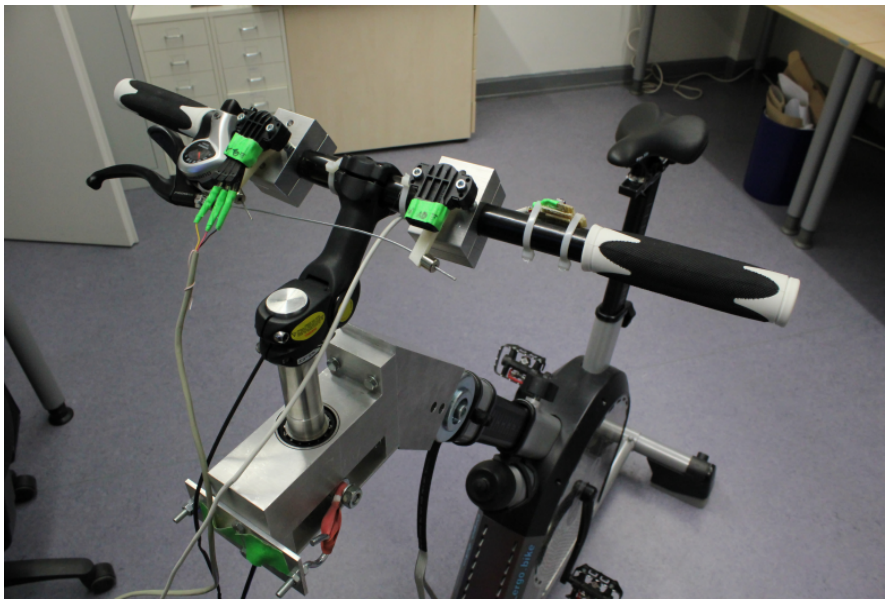


Abbildung A.2.: Lenkvorrichtung mit drehbarem Lenker

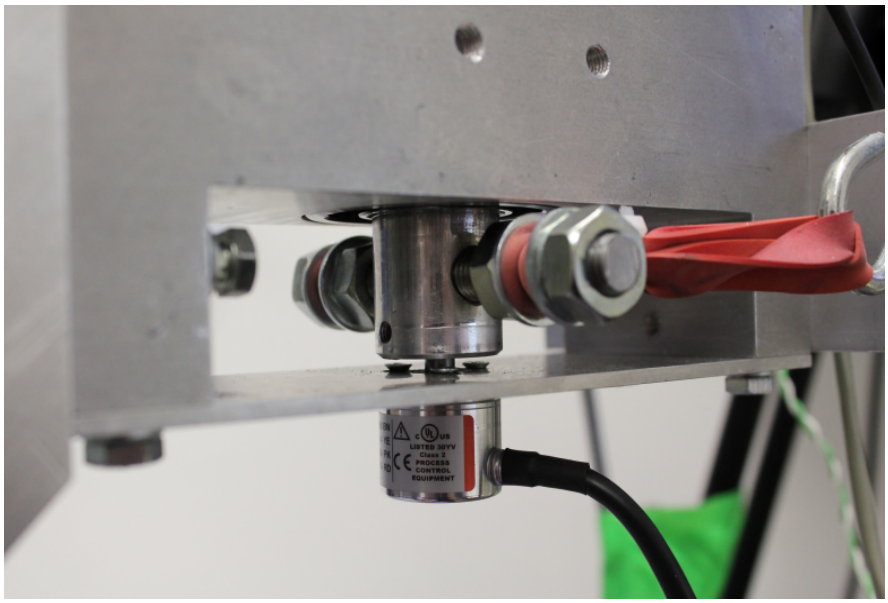


Abbildung A.3.: An der Lenkvorrichtung montierter Inkrementalgeber

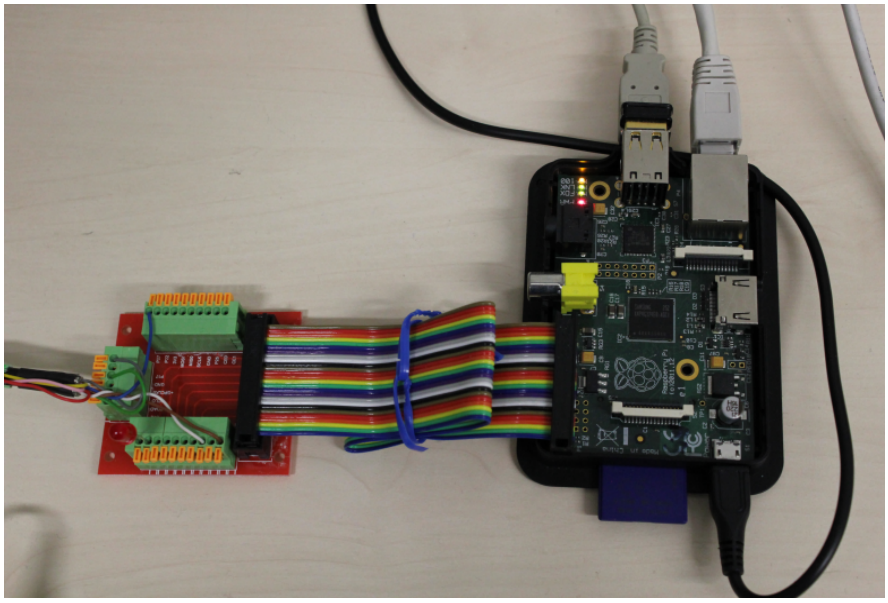


Abbildung A.4.: Raspberry Pi mit Extensionboard und dem daran angeschlossenen Kabel des Inkrementalgebers

A.2. Konstruktionszeichnungen Lenkvorrichtung

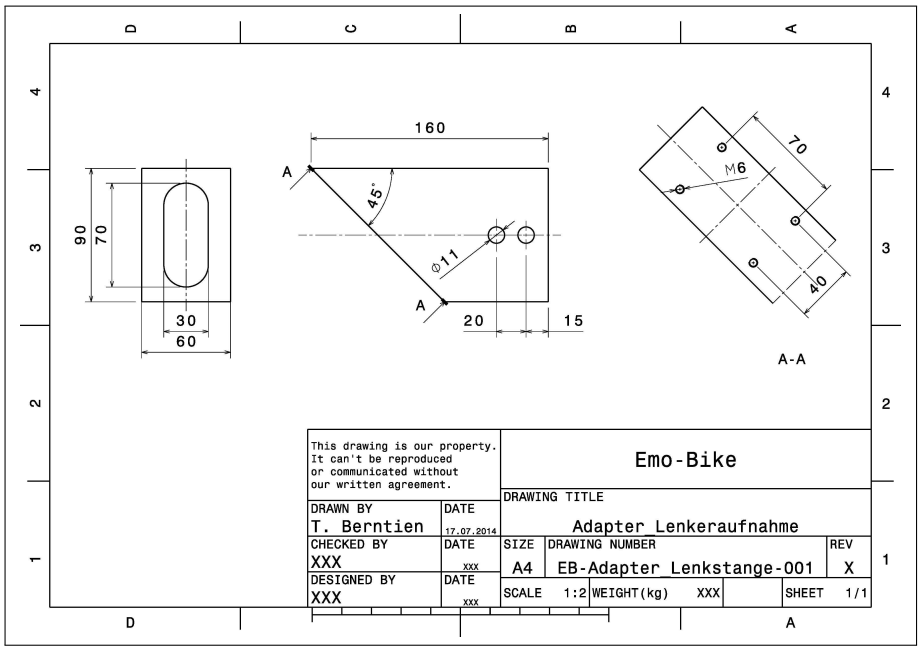


Abbildung A.5.: Adapter Lenkstange (Konstruktionszeichnung: Thomas Berntien)

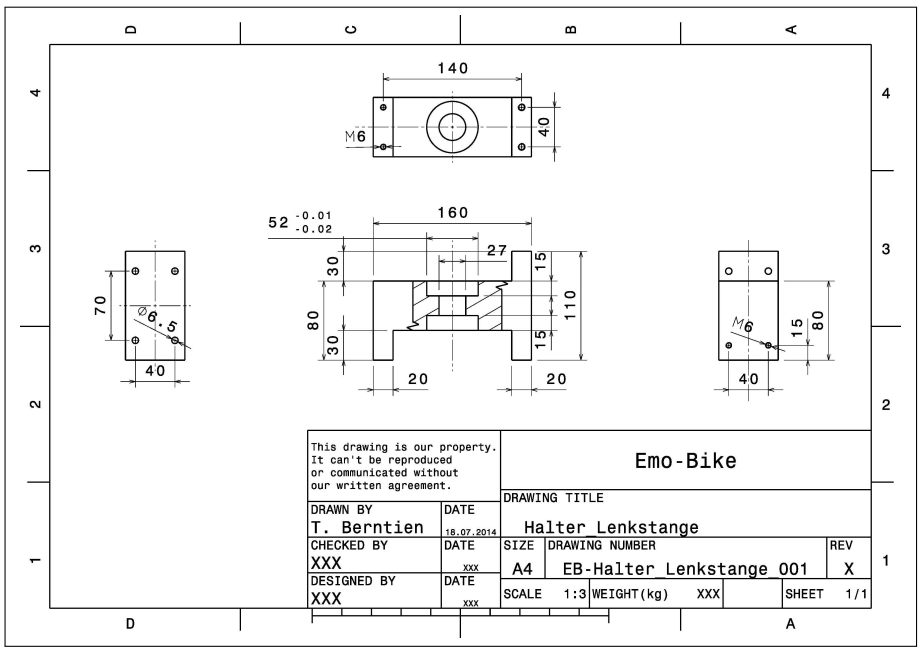


Abbildung A.6.: Halter Lenkstange (Konstruktionszeichnung: Thomas Berntien)

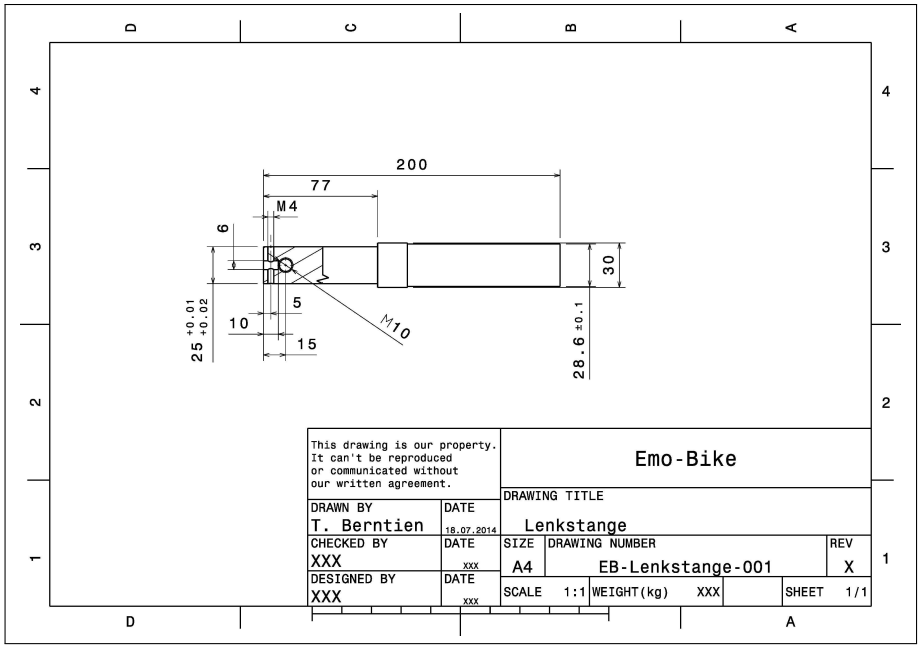


Abbildung A.7.: Lenkstange (Konstruktionszeichnung: Thomas Berntien)

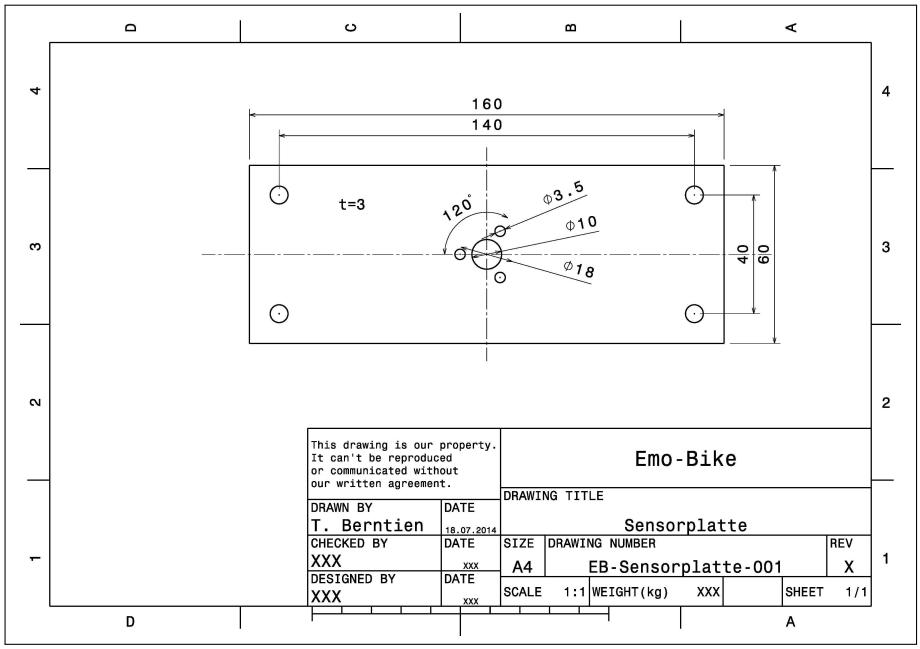


Abbildung A.8.: Sensorplatte (Konstruktionszeichnung: Thomas Berntien)

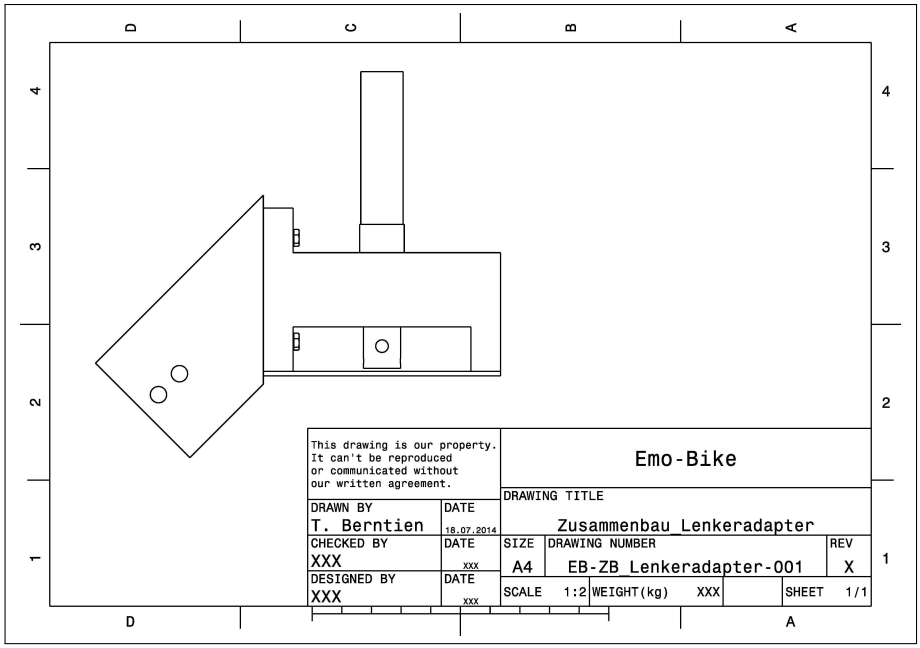


Abbildung A.9.: ZB Lenkeradapter (Konstruktionszeichnung: Thomas Berntien)

Literaturverzeichnis

[apache.org 2015] APACHE.ORG: *Apache License, Version 2.0*. Website. 2015. – URL http://www.apache.org/licenses/LICENSE-2.0. – (letzter Zugriff: 01.03.2015)

[Bærentsen 2001] BÆRENTSEN, Klaus B.: Intuitive User Interfaces. In: *Scand. J. Inf. Syst.* 12 (2001), Januar, Nr. 1-2, S. 29–60. – URL http://dl.acm.org/citation.cfm?id=372668.372680. – ISSN 0905-0167

[Barco 2015] BARCO: *Successful introduction of Barco's radar simulation software at German Water Police Academy*. Website. 2015. – URL http://www.barco.com/en/News/Press-releases/Successful-introduction-of-Barcos-radar-simulation\discretionary{-}{}{}software-at-German-Water-Police-Academy.aspx. – (letzter Zugriff: 02.03.2015)

[Bernin 2011] BERNIN, Arne: *Einsatz von 3D-Kameras zur Interpretation von räumlichen Gesten im Smart Home Kontext*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Buschmann u. a. 1996] BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: *Pattern-oriented Software Architecture: A System of Patterns*. New York, NY, USA : John Wiley & Sons, Inc., 1996. – ISBN 0-471-95869-7

[Carroll 1990] CARROLL, John M.: *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. Cambridge, MA, USA : MIT Press, 1990. – ISBN 0-262-0316390

[cdn.sparkfun.com 2015] CDN.SPARKFUN.COM: *11546-04.jpg*. Website. 2015. – URL https://cdn.sparkfun.com//assets/parts/7/4/9/7/11546-04.jpg. – (letzter Zugriff: 01.03.2015)

[conrad.de 2015] CONRAD.DE: *198118-LB-00-FB.EPS-1000.jpg*. Website. 2015. – URL http://www.conrad.de/medias/global/ce/1000_1999/1900/1980/1981/198118-LB-00-FB.EPS-1000.jpg. – (letzter Zugriff: 01.03.2015)

[Daum Electronic GmbH 2005] Daum Electronic GmbH: *Kommunikationsprotokoll für Geräte der premium- und medical-Serie*. Website. 2005. – URL www.daum-electronic.de/de/download/bedaprem/SP-KOMM-PM1.pdf. – (letzter Zugriff: 02.03.2015)

[daum electronic.de 2015] electronic.de daum: *Unsere ERGO-Bikes im Gesamtüberblick*. Website. 2015. – URL http://daum-electronic.de/download/vergleich.pdf. – (letzter Zugriff: 01.03.2015)

[eLinux.org 2015] eLinux.org: *Raspberry Pi Kernel Compilation*. Website. 2015. – URL http://elinux.org/RPi_Kernel_Compilation. – (letzter Zugriff: 01.03.2015)

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0-201-63361-2

[Hamlin 2013] Hamlin, Jay: *Raspberry Pi ISR*. Website. 2013. – URL https://sites.google.com/site/hamlinhomeprojects/projects/raspberry-pi-isr. – (letzter Zugriff: 01.03.2015)

[Henderson 2014a] Henderson, Gordon: *Priority, Interrupts and Threads*. Website. 2014. – URL http://wiringpi.com/reference/priority-interrupts-and-threads/. – (letzter Zugriff: 01.03.2015)

[Henderson 2014b] Henderson, Gordon: *WiringPi Library*. Website. 2014. – URL http://wiringpi.com/. – (letzter Zugriff: 27.02.2015)

[Hewett u. a. 1992] Hewett, Thomas T. ; Baecker, Ronald ; Card, Stuart ; Carey, Tom ; Gasen, Jean ; Mantei, Marilyn ; Perlman, Gary ; Strong, Gary ; Verplank, William: ACM SIGCHI Curricula for Human-Computer Interaction. New York, NY, USA : ACM, 1992. – Forschungsbericht. – ISBN 0-89791-474-0

[Nielsen 1993] Nielsen, Jakob: *Usability Engineering*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1993. – ISBN 0125184050

[Otto und Voskuhl 2011] Otto, Kjell ; Voskuhl, Sören: Weiterentwicklung der Architektur des Living Place Hamburg. 2011. – Forschungsbericht

[Picard 1997] Picard, Rosalind W.: *Affective Computing*. Cambridge, MA, USA : MIT Press, 1997. – ISBN 0-262-16170-2

[popcornmix 2015] POPCORNMIX: *gcc-linaro-arm-linux-gnueabihf-raspbian*. Website. 2015. – URL https://github.com/raspberrypi/tools/tree/master/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian. – (letzter Zugriff: 01.03.2015)

[Potratz 2014] POTRATZ, Olaf: *Ein Framework für physikbasierte 3D Interaktion mit großen Displays*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2014

[raspberry projects.com 2015] PROJECTS.COM raspberry: *Model B Hardware General Specifications*. Website. 2015. – URL http://www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-model-b/hardware-general-specifications. – (letzter Zugriff: 01.03.2015)

[qtproject.org 2015] QTPROJECT.ORG: *Beginner's guide to cross-compile Qt5 on RaspberryPi*. Website. 2015. – URL http://qt-project.org/wiki/RaspberryPi_Beginners_guide. – (letzter Zugriff: 01.03.2015)

[raspberrypi.org 2015] RASPBERRYPI.ORG: *What is a Raspberry Pi?* Website. 2015. – URL http://www.raspberrypi.org/help/what-is-a-raspberry-pi/. – (letzter Zugriff: 01.03.2015)

[Simutech 2014] SIMUTECH: *Fahrsimulatoren und Fahrtrainer für die PKW-Fahrerausbildung*. Website. 2014. – URL http://www.simutech.de/fahrschulsimulator_pkw.htm. – (letzter Zugriff: 02.03.2015)

[Tanenbaum und Bos 2014] TANENBAUM, Andrew S. ; BOS, Herbert: *Modern Operating Systems*. 4th. Upper Saddle River, NJ, USA : Prentice Hall Press, 2014. – ISBN 013359162X, 9780133591620

[TU Graz 2015] TU GRAZ: *Research Platform Flight Simulation*. Website. 2015. – URL http://portal.tugraz.at/portal/page/portal/TU_Graz/Einrichtungen/Institute/Homepages/i3050/flusi. – (letzter Zugriff: 02.03.2015)

[Weiser 1991] WEISER, Mark: The Computer for the 21st Century. (1991). – URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html. – (letzter Zugriff: 26.02.2015)

[wikipedia.org 2015] WIKIPEDIA.ORG: *Qt Bibliothek Verwendungsbeispiele*. Website. 2015. – URL http://de.wikipedia.org/wiki/Qt_%28Bibliothek%29#Verwendungsbeispiele. – (letzter Zugriff: 01.03.2015)

[Zagaria 2015] ZAGARIA, Sebastian: Visualisierung von Virtuellen Welten für ein Affective-Exergame im Rahmen des Emo-Bike Projektes. 2015. – Forschungsbericht

Abbildungsverzeichnis

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 3. März 2015 Jonas Hornschuh