Universität Leipzig
Wirtschaftswissenschaftliche Fakultät
Institut für Wirtschaftsinformatik
Professur Softwareentwicklung für Wirtschaft und Verwaltung

Bachelorarbeit

Freie wissenschaftliche Arbeit
zur Erlangung des akademischen Grades Bachelor of Science Wirtschaftsinformatik an der Wirtschaftswissenschaftlichen Fakultät der Universität Leipzig

Alternative Steuerungen für das VR-Labor

Betreuender Hochschullehrer: Prof. Dr. Ulrich Eisenecker
Bearbeiter: Christoph Jobst

Eingereicht am: 23. September 2010

Abstract

Für das Virtual Reality-Labor des Instituts für Wirtschaftsinformatik der Universität Leipzig sollen Alternativen zu den bestehenden Steuerungsmöglichkeiten gefunden, an die Anlage angeschlossen und evaluiert werden. Es werden Vorgehensweisen dargelegt, die den Anschluss neuer Geräte ermöglichen sowie deren Anwendung am Beispiel der Nintendo Wii-Remote gezeigt.

Schlüsselwörter

Device, Emulation, VRPN, Wii

Gliederung

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

3D	dreidimensional
CAD	Computer-aided Design
DDK	Driver Development Kit
DLL	Dynamic Link Library
HID	Human Interface Device
INFITEC	Interferenzfiltertechnologie
LED	Light-emitting Diode
LPT	Line Printing Terminal
PIE	Programmable Input Emulator
PPJoy	Parallel Port Joystick
ROM	Read-only Memory
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
VDP	Visual Decision Platform
VR	Virtual Reality
VRPN	Virtual Reality Peripheral Network

1 Einleitung

Die dreidimensionale (3D) Darstellung wird immer dann gern verwendet, wenn ein umfassender Eindruck des zu betrachtenden Gegenstandes vermittelt werden soll. Sie findet immer häufiger Einzug in Fernseher der Privathaushalte und Kinos. Firmen nutzen sie für die Erstellung und Veranschaulichung ihrer Maschinen und Projekte. In der Videospielbranche ist sie schon seit Jahren nicht mehr wegzudenken. Selbst Software kann auf diese Weise abgebildet werden. Nutzer sollen dabei aber oft kein statisches Bild vor sich haben, sondern auch mit den Darstellungen interagieren können.

1.1 Motivation

Virtual Reality (VR) entführt ihre Nutzer in eine digitale Welt. Sie sollen bestmöglich in diese Umgebung integriert werden und realitätsnah mit ihr interagieren können. Das Ziel ist ein immersiver Eindruck des Gezeigten. Dem Steuergerät kommt dabei eine besondere Bedeutung zu, da es der Interaktionspunkt des Nutzers mit der virtuellen Realität ist und dessen Erlebnis und Produktivität entscheidend beeinflusst. Darum werden intuitiv und effektiv einsetzbare Steuergeräte benötigt, die einfache Metaphern und alltägliche Bewegungsabläufe umsetzen. Es gibt viele konkurrierende Techniken. Einige der interessantesten davon sind in der Wii-Remote vereint. Sie ist vom Hersteller Nintendo als Spielcontroller konzipiert worden und ermöglicht auch Gelegenheitsspielern eine Nutzung ohne lange Eingewöhnungsphase. Mit natürlichen Bewegungen und Gesten können Spieler ihre virtuellen Figuren unter anderem bowlen oder Tennis spielen lassen. Daher soll untersucht werden, welche Möglichkeiten sich damit für die Interaktion mit einer virtuellen Realität im Bereich dreidimensionaler Modellvisualisierung anbieten und wie diese nutzbar gemacht werden können.

1.2 Zielstellung

Für das Virtual Reality-Labor des Instituts für Wirtschaftsinformatik der Universität Leipzig sollen Steuerungsalternativen erschlossen werden, indem weitere Eingabemöglichkeiten gefunden, an die bestehende Anlage angeschlossen und evaluiert werden. Beispielhaft soll dabei die Nintendo Wii-Remote untersucht werden. Im Labor kommt die Visualisierungssoftware *Visual Decision Platform* (VDP) der Firma IC:IDO zum Einsatz. Diese ermöglicht die Betrachtung von und Interaktion mit dreidimensionalen

Modellen am Computermonitor sowie auf einer mehrelementigen Projektionsfläche (*Powerwall*). Beide Varianten sollen mit der Wii-Remote bedient werden können. Letztere zum einen über Synchronisation des Desktops mit der Projektionswand, zum anderen über die ursprünglich von IC:IDO vorgesehene Programmierschnittstelle. In diesem Zuge soll auch das Potential dieser Fernbedienung beurteilt werden. Es wird darauf eingegangen, mit welchen Methoden und Werkzeugen ein Anschluss an die Anlage erfolgen kann und deren praktische Anwendung auch im Hinblick auf weitere Steuergeräte erläutert.

1.3 Grundlagen der Steuerung und Interaktion

Der Nutzer eines Systems, das eine interaktive, dreidimensionale Darstellung bietet, muss mit den ihm gegebenen Möglichkeiten in diesem virtuellen Raum navigieren können. Dies geschieht über sechs Freiheitsgrade. Diese sind die Translationsbewegungen entlang der Achsen *x*, *y* und *z* sowie die zugehörigen Rotationen (Abbildung 1). Die Rotation um die vertikale Achse wird als *yaw*, um die aus Nutzersicht von links nach rechts verlaufende Achse als *pitch* und um die nach vorn in den Raum verlaufende Achse als *roll* bezeichnet. Eine Bewegung wird als Vektor und als Abfolge von Rotationen beschrieben. Auf diese Weise können jeder beliebige Punkt und jeder Blickwinkel eingenommen werden.

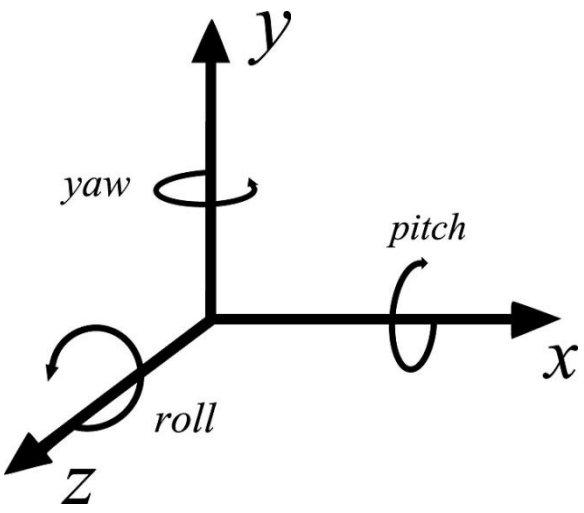

Abbildung 1: Die sechs Freiheitsgrade

Grundbaustein von Interaktionen ist die Ereignissteuerung. Auf eine gezielte Aktion in der physischen Welt folgt als Konsequenz eine vorhersehbare Reaktion in der virtuellen Umgebung. Alle Ereignisse werden vom Nutzer über das Steuergerät initiiert. Zur Auslösung der Ereignisse werden verschiedene Kombinationen aus digitalen und analogen Techniken genutzt.

Die am häufigsten eingesetzten Techniken sind:

- Buttons
- Sprache
- Lagesensor
- Schieberegler
- Analog-Sticks
- Geschwindigkeitsmessung
- Positionserfassung im Raum
- Berührungssensitive Oberfläche

Die Verfolgung bewegter Objekte im Raum wird als Tracking bezeichnet. Es gibt mehrere Techniken, nämlich optisch, magnetisch und die Nutzung von (Ultra-)Schall.
Analoge Signale haben gegenüber digitalen den Vorteil, nahezu stufenlos variierbar zu sein. Sie müssen jedoch digitalisiert werden, bevor eine Nutzung durch die Software möglich ist. Um den immersiven Eindruck zu verstärken, sollten flüssige Bewegungen als Eingabe möglich sein und vom System ebenso interpretiert und umgesetzt werden. Über hohe Abtastraten lässt sich dies auch mit digitalen Techniken erreichen, sodass im Idealfall kein Unterschied zu analogen Eingaben spürbar ist.

1.4 Vorhandenes Steuersystem

Analog statt digital galt bei der Auswahl der Standardsteuerung, welche zur Virtual Reality-Anlage des Instituts gehört. Es handelt sich dabei um ein optisches Trackingverfahren, bei dem durch Reflektion infraroten Lichts die Position des Eingabegerätes errechnet wird.
Die VDP ermöglicht die Darstellung von dreidimensionalen Inhalten an einer *Powerwall*. Am oberen Rand dieser Projektionsfläche befinden sich vier Infrarotkameras, um deren Objektive jeweils ein Infrarot-LED-Panel (engl. *light-emitting diode*) angebracht ist. Das Steuergerät IC:Control ist ein Stab, an dem sich, neben zwei Buttons, fünf Reflektoren befinden. Es gibt eine kabellose und eine kabelgebundene Variante. Der Anschluss an den PC erfolgt über die serielle Schnittstelle (EIA-232), die auf die ursprünglich parallele Schnittstelle (LPT, engl. *line printing terminal*) des Steuergerätes adaptiert wurde. Die Aufnahmen der Infrarotkameras werden an den eigens dafür vorgesehenen Tracking-Rechner gesendet, der die Daten konsolidiert und auswertet. Dazu müssen mindestens zwei der Kameras freien Blick auf die Reflektoren haben. Anhand der Reflektionsmuster wird

die Position vor der Projektionsfläche sowie der Haltungswinkel des Stabes errechnet. Die Berechnungen erledigt dabei die *DTrack*-Software der A.R.T. GmbH [A.R.T., 2010]. Anschließend werden die Werte an den Hauptrechner gesendet, von dort mittels *Virtual Reality Peripheral Network*[1] (VRPN) an die drei Rechner für die Projektorsteuerung weitergeleitet und letztlich auf den Cursor an der *Powerwall* abgebildet. Dies erfolgt mit absoluten Werten, wodurch eine bestimmte Positionierung des Stabes immer die gleiche Position und Haltung des Cursors auf der Projektionsfläche bedingt. Für absolut gewertete Eingaben ist ein fester Bewegungsradius an der Projektionsfläche vorgegeben. Daher gibt es auch keine Zurücksetzfunktion (*reset*) für den Cursor, denn er kann nicht aus dem projizierten Bild verschwinden. Die Betrachtungsperspektive kann zudem von der Kopfhaltung beeinflusst werden, da die Interferenzfilterbrille (INFITEC-Brille) des Hauptnutzers ebenfalls durch Reflektoren getrackt wird. Die Brille ist somit auch ein Steuergerät, dessen Funktionsumfang sinnvoll eingeschränkt wurde. Brille und Stab werden vom System anhand der spezifischen Reflektoranordnung unterschieden.

Eine zweite Steuerungsoption ist die Synchronisation der Projektionswand mit der CAD (engl. c*omputer-aided design*)-Anzeige des IC:IDO-Software-Hauptrechners. Diese Methode bietet alle inhaltlichen Funktionen der Powerwallsteuerung und erweitert diese durch die nun zugänglichen CAD-Funktionen der Desktopsteuerung. Die Eingaben erfolgen über Tastatur und Maus. Allerdings ist es nicht möglich, den projizierten Cursor zu nutzen. Außerdem haben Desktop und Projektion unterschiedliche Referenzsysteme für die Umsetzung der Bewegungsbefehle. Die Auswahl der VDP (*ImmersiveWorkspace*) als Referenzpunktsystem bei gleichzeitiger Nutzung eines Steuersystems für die *Powerwall*, das nicht von der Orientierung des 3D-Cursors abhängig ist, behebt dies.

Die dritte und letzte Alternative bietet das Pluginsystem der VDP [IC:IDO, 2006, S. 118]. Weitere Eingabegeräte können durch Hinzufügen einer entsprechenden DLL (engl. *dynamic link library*) oder Shared-Library zur Software des Hauptrechners ausgelesen werden. Dieses System arbeitet, wie auch das optische Trackingsystem, mit VRPN und erlaubt auf diesem Weg die Kontrolle des projizierten Cursors. Über ein Plugin kann im Virtual Reality-Labor bereits ein Logitech-Gamepad genutzt werden. Dessen Konfiguration ist allerdings nur wenig intuitiv und durch die Kabelgebundenheit wird das Erleben der virtuellen Realität beeinträchtigt.

[1] Siehe dazu Abschnitt 4

2 Alternative Steuergeräte

2.1 Überblick

Es gibt viele etablierte Speziallösungen für unterschiedlichste Ansprüche. In Flugsimulatoren erhalten zukünftige Piloten ihre Grundausbildung. Am *Ausbildungsgerät Schießsimulator Handwaffen/Panzerabwehrhandwaffen* (AGSHP) trainiert die Bundeswehr. Virtual Reality hilft auch Leben zu retten: Ärzte können mit dem *Sensable Phantom* oder dem *da Vinci Surgical System*[2] Eingriffe planen, simulieren und mit Letzterem praktisch durchführen. Bis auf das *Sensable Phantom* wäre ein Einsatz solcher Geräte im Virtual Reality-Labor des Instituts für Wirtschaftsinformatik aber deutlich überdimensioniert und nicht für den dort vorgesehenen Einsatzzweck, Softwarevisualisierungen kollaborativ zu explorieren, tauglich. Daher werden nur kleinere Geräte weiter betrachtet, die ein Ersatz für die IC:Control werden können.

Unter Berücksichtigung der gegebenen Laborausstattung bieten sich Geräte mit optischem Tracking oder gänzlich von Tracking unabhängigen Methoden an.

Eine Option sind berührungssensitive Oberflächen mit kapazitiver Messung des Drucks. Dadurch sind die drei Translationsbewegungen abgedeckt. Rotationen können mittels verschiedener Streichbewegungen einer Mehrfingersteuerung durchgeführt werden.

Weiterhin sind Datenhandschuhe[3] einsetzbar. Bewegungen der Finger werden über Dehnmessstreifen oder Lichtbrechungsmessung, Bewegungen der Hand über ein optisches oder magnetisches Trackingsystem erfasst. Mit ihnen können dreidimensionale Objekte auf einer Projektionswand auf natürliche Weise gegriffen und bearbeitet werden. Gestensteuerung ist ebenfalls umsetzbar, beispielsweise die Objektauswahl durch Zeigen. Die Handschuhe können auch eine Kraftrückkopplung (engl. *force feedback*) besitzen.

Ausschließlich mit Gesten funktioniert das Motion-Tracking. Noch wird es vorrangig zur Aufnahme von Bewegungsabläufen eines dreidimensionalen Körpers genutzt. Microsoft entwickelt augenblicklich das Eingabegerät *Kinect* (Projektname *Natal*), mit dessen Hilfe keine Steuergeräte in Nutzerhänden mehr nötig sein sollen [Microsoft, 2010]. Alles wird über Spracheingaben und die Bildauswertung einer Tiefensensorkamera realisiert. *Kinect* soll zunächst für die Microsoft-Spielkonsole Xbox 360, später auch für Windowscomputer erscheinen. Der Preis wird bei 150€ liegen, vorläufiges Erscheinungsdatum ist der 10.

[2] Siehe http://www.sensable.com und http://www.intuitivesurgical.com, letzter Abruf jeweils am 19.06.2010.

[3] Siehe http://www.5dt.com/ - letzter Abruf am 19.06.2010.

November 2010. Das SDK (engl. *software development kit*) ist bislang nur für registrierte Spielentwickler verfügbar.

Die übliche Computersteuerung aus Tastatur und Zwei-Tasten-Maus ist ebenfalls für VR geeignet, wenn auch eingeschränkt. Eine Maus kann zwei Translationen durch Verschieben und eine durch das Mausrad abdecken. *Yaw*, *pitch* und *roll* können in beliebiger Verbindung mit einer gedrückten Maus- oder Tastaturtaste ausgeführt werden. Mehr Möglichkeiten bieten 3D-Mäuse[4]. Sie sind im CAD und VR-Bereich weit verbreitet, weil sie die intuitive Navigation in den sechs Freiheitsgraden unterstützen. Die Desktopgeräte basieren auf Kipp-, Dreh- und Druckeingaben. Es gibt auch tragbare Varianten mit Tracking, zu denen das vorhandene IC:Control gehört.

Ein Gerät, das als solche 3D-Maus genutzt werden kann, wird im Folgenden vorgestellt.

2.2 Nintendo Wii-Remote

Die Nintendo Wii-Remote (Abbildung 2) ist das Hauptsteuergerät für die Wii-Spielkonsole von Nintendo und erschien 2006. Die erstmals in einem Spielcontroller verbaute Technologiekombination eröffnete komplett neue Interaktionsweisen mit den Spielen.

Abbildung 2: Wii-Remote (ähnlich [WC, 2010])

[4] Siehe http://www.spacecontrol.de und http://www.3dconnexion.com - letzter Abruf jeweils am 19.06.2010. Als Ideenschmiede für neue 3D-Mauskonzepte hat sich die Universität Weimar hervorgetan. Siehe http://www.uni-weimar.de/cms/medien/vr/research/hci.html - letzter Abruf am 13.06.2010.

2.2.1 Hardware

Die Wii-Remote ist eine Fernbedienung, die über Bluetooth mit dem zu steuernden Gerät kommuniziert. Die maximale Entfernung zum Empfänger beträgt zehn Meter. In ihr ist ein Beschleunigungssensor sowie ein Lagesensor verbaut, der *roll* und *pitch* erfasst. An der Frontseite befindet sich eine Infrarotkamera, welche die Lichtsignale der Infrarotleiste trackt. Ihr Sichtradius beträgt circa 45°. *Yaw* ist nur in Verbindung mit der Infrarotleiste messbar. Diese Leiste wird von Nintendo zwar als Sensorleiste (*Sensor-Bar*) bezeichnet, beinhaltet allerdings nur zwei Infrarot-LED-Panel. An der Wii-Remote sind neun Buttons und ein Steuerkreuz vorhanden. Einer der Buttons befindet sich im Batteriefach und einer fungiert als Power-Knopf. Somit sind sieben von ihnen einsetzbar. Ein kleiner Motor im Inneren ermöglicht Kraftrückkopplung. Ein Lautsprecher ist auch verbaut, kann aber nur Töne minderer Qualität hervorbringen. Vier blaue LEDs zeigen verschiedene Zustandsmodi, wie Batteriestärke oder Spielernummer an. Weiterhin befindet sich an ihr ein Erweiterungsport. Darüber kann diverses Zubehör angeschlossen werden. Beispielsweise das *Motion Plus*, was der Wii-Remote einen Gyrometer und einen verfeinerten Lagesensor bereitstellt. Das *Motion Plus* besitzt selbst einen Erweiterungsport und kann als transparenter Intermediär zwischen der Wii-Remote und zusätzlichen Erweiterungen genutzt werden. Da die Wii-Remote für viele körperbetonte Spiele gedacht ist, hat Nintendo sie zudem mit einer optionalen Handgelenksschlaufe und einer Gummihülle ausgestattet.

Für das Labor des Instituts wurde folgende Hardware beschafft:

- Wii-Remote
- Sensorleiste (batteriebetrieben, alternativer 12V Netzanschluss)
- *Motion Plus*
- Ladegerät und 2 Akkus
- USB (engl. *universal serial bus*) Bluetooth-Stick

2.2.2 Verbindungsherstellung

Die Wii-Remote arbeitet mit Bluetooth, ist aber standardmäßig nicht für Bluetooth-Empfänger sichtbar. Dies muss zur Verbindungsherstellung kurzzeitig aktiviert werden. Das geschieht über den Button im Batteriefach, komfortabler geht es durch gleichzeitiges Drücken der Buttons *One* und *Two*. Dieser Zustand dauert 20 Sekunden an und kann durch Halten der Buttons beliebig verlängert werden. Er wird durch das Blinken mindestens einer

LED angezeigt, wobei die Anzahl der Lichter gleichzeitig die Restkapazität der Batterie wiedergibt. Am Computer wird das Gerät als *Nintendo RVL-CNT-01* erkannt und muss ohne Verbindungsschlüssel angeschlossen werden. Es sollte sich keine aktive Wii-Konsole in der Nähe befinden, da sie sich sonst dort zu registrieren versucht.

Über die Software von *BlueSoleil*, die im Lieferumfang des Bluetooth-Sticks enthalten war, vollzieht sich der Anschluss einfach und auch nach einem Neustart von Rechner oder Wii-Remote zuverlässig. Doch der genutzte Bluetooth-Treiber installiert zugleich viele Platzhaltergeräte als Joysticks. Dadurch nutzt der VRPN-Server auf dem Hauptrechner einen nicht vorhandenen Joystick anstatt der Wii-Remote oder des angeschlossenen Gamepads und arbeitet nicht mehr sinnvoll mit *human interface device* (HID)-Geräten. Für die lokale Nutzung ohne VRPN ergeben sich dagegen keine Probleme.

Weniger zuverlässig, dafür aber mit VRPN kompatibel, ist der Standardtreiber für Bluetooth von Windows XP. *BlueSoleil* muss vor seiner Nutzung vollständig entfernt worden sein. Je nach Treiberversion und genutztem Bluetooth-Stick kommt es mit dem XP-Treiber zu Problemen. Bei manchen Konfigurationen wird die Wii-Remote sofort nach dem Anschluss wieder getrennt. Dies kann auf folgendem Weg behoben werden: Zunächst wird die *services.msc* über den Ausführen-Dialog gestartet. Dadurch wird die lokale Systemdienstliste aufgerufen. Unter diesen Diensten befindet sich der *Bluetooth Support Service*. In dessen Eigenschaften muss auf das lokale Systemkonto umgestellt und der Datenaustausch zwischen Service und Desktop erlaubt werden. Weiterhin fordert Windows den Nutzer beim Anschluss der Wii-Remote in unregelmäßigen Abständen zu einem Rechnerneustart auf. Das kann ignoriert werden. Bei weiteren Problemen muss auf Treiber des Stickherstellers ausgewichen werden, wobei auch das keine Garantie bietet.

Während des Anschlusses identifiziert sich die Wii-Remote durch Deskriptorsegmente in deren *read-only memory* (ROM) als HID-Gerätetyp. Sie fällt somit in die HID-Geräteklasse und wird über das Bluetooth HID-Profil eingebunden. Dieses leitet direkt auf die USB HID-Klasse weiter. USB HID stellt eine einheitliche Schnittstelle für den Datentransfer vom Steuergerät zum HID-Klassentreiber und umgekehrt bereit. Auf diese Weise stehen die Eingabedaten allgemein zur Verfügung und können von einer Anwendung ausgelesen werden [USB, 2001, S. 10].

Unter Windows können die Werte dank der *DirectInput*-Bibliothek unmittelbar entgegengenommen und von der implementierenden Anwendung verarbeitet werden. Dies verringert Reaktionszeit und Rechenlast.

Da die Wii-Remote nicht für den Einsatz am Computer konzipiert wurde, sind die übertragenen Daten zwar lesbar, aber nicht spezifiziert und müssen zur Nutzung erst

manuell zugeordnet werden. Daher funktioniert die Wii-Remote beispielsweise ad hoc nicht als Mausersatz und manche Bluetooth-Treiber verweigern den Dienst.
Die Verbindungstrennung gestaltet sich einfach. An der Wii-Spielkonsole schaltet sich die Wii-Remote selbstständig nach mehreren Minuten Inaktivität ab und der Power-Knopf wird für die Konsole genutzt. Am Computer trennt er die Bluetooth-Verbindung der jeweiligen Wii-Remote. Da das Gerät dauerhaft im Bereitschaftsmodus ist, empfiehlt sich zudem ein Entfernen des Akkus[5].

2.2.3 Anwendungsmöglichkeiten

Das Erweiterungssystem macht aus der Wii-Remote einen unerschöpflichen Pool von Innovationen. Sie kann beispielsweise in spezielle Spielgitarren oder in den Griff eines Tennisschlägers eingesteckt werden. Wird sie in ein Lenkrad eingebaut, kann dank des Lagesensors ein virtuelles Fahrzeug realitätsnah gesteuert werden. Ebenfalls kann der *Nunchuk*, ausgestattet mit zwei Buttons, einem Analog-Stick und eigenem Beschleunigungssensor, direkt an die Wii-Remote angeschlossen werden, wodurch beide Hände eingesetzt werden können. Das *Balance Board*, eine Kombination aus Waage und verschiedenen Druckpunkten, wird von Nintendo für Gymnastik- und Wintersportspiele genutzt und kommuniziert ebenfalls über Bluetooth. All diese Geräte lassen sich nicht nur in Spielen einsetzen, sondern auch im industriellen VR-Bereich. Ein Problem bei der Erkundung virtueller Welten ist etwa, dass der Nutzer bewegungslos an einem fixen Punkt verharrt und sich seine Umgebung dennoch verändert. Die alleinige Verwendung von Handsteuergeräten ist nur bedingt realitätsnah. Mit der Technik des *Balance Boards* können durch Veränderung der Körperhaltung eine Spielwelt oder Modelle durchflogen werden, während das Handgerät die restliche Interaktion übernimmt.
Weitere beispielhafte Verwendungen für die Wii-Remote und ihr Zubehör am Computer[6]:

- Kopf- und Fingertracking
- Zeigegerät für Präsentationen
- Interaktive Tafel mit Infrarot-LED-Stift
- Umsetzung realitätsnaher Steuerungsmetaphern

In den folgenden Abschnitten wird gezeigt, wie eine einzelne Wii-Remote als 3D-Maus für die Anlage im VR-Labor nutzbar gemacht werden kann.

---

[5] Um die Schutzhülle dabei nicht zu beschädigen, ist die Bedienungsanleitung der Wii-Remote zu beachten.
[6] Der Programmierer Johnny Chung Lee hat einige sehr interessante Projekte für die Wii-Remote erstellt. Näheres dazu hier: http://johnnylee.net/projects/wii/ - letzter Abruf am 23.07.2010.

3 Emulation

Werden Systemzustände nachgeahmt und einem anderen oder dem gleichen System vorgetäuscht, so wird dies Emulation genannt. Im Rahmen dieser Arbeit wird diese Technik genutzt, um die Standardsteuerung für die VDP durch die Wii-Remote zu ersetzen. Der Vorgang bedarf einiger Vorbereitungen und kann im VR-Labor auf zwei Weisen umgesetzt werden.

Zur Erleichterung der Arbeit wurde auf zwei Programme zurückgegriffen. Zum einen GlovePIE (engl. *programmable input emulator*) und zum anderen PPJoy. Beide Programme werden von jeweils nur einem Programmierer als Freizeitprojekt betreut.

3.1 GlovePIE

GlovePIE (Abbildung 3) bietet eine Entwicklungsumgebung für Skripte, die direkt im integrierten Editor erstellt und interpretiert werden können. Das Programm war ursprünglich zur Anwendung am *P5-Dataglove* konzipiert, wurde aber kontinuierlich erweitert und so zu einem universell einsetzbaren Emulator. Der Entwickler Carl Kenner hat die Anwendung für die kommerzielle und die nicht kommerzielle Nutzung freigegeben und dabei nur wenige Auflagen erteilt. So darf das Programm beispielsweise nicht für militärische Zwecke genutzt werden. Die komplette Lizenz befindet sich in [Kenner, 2010, S. 6]. Die aktuelle Version ist 0.43.

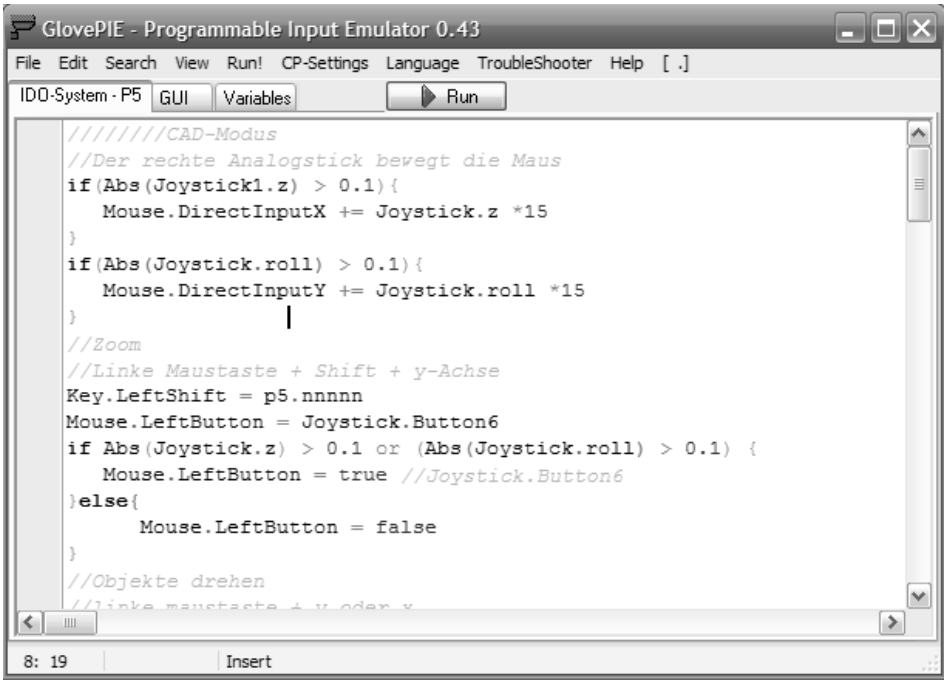

Abbildung 3: GlovePIE Oberfläche

Programmieren gestaltet sich in GlovePIE sehr intuitiv und der Code ist leicht verständlich. Die Skriptsprache ist Turing-vollständig. Elemente aus diversen Programmiersprachen sind in die PIE-Sprache eingeflossen. Das spiegelt sich unter anderem in den Regeln zur Kommentierung und Zuweisung wider [Kenner, 2010, S. 9].

```
//  Kommentar
/*  Mehrzeiliger
    Kommentar */
%   Kommentar (MATLAB)
'   Kommentar (BASIC)
REM Kommentar (BASIC)
//Gedrückte linke Maustaste melden bei betätigen von Button1 am Joystick1
Mouse.LeftButton = Joystick1.Button1
Mouse.LeftButton := Joystick1.Button1
Joystick1.Button1 => Mouse.LeftButton
```

Listing 1: Sprachübergreifendes Programmiermodell

Ähnlich verhält es sich mit Schleifen und Auswahlkonstrukten. GlovePIE ist flexibel genug, um mit vielen unterschiedlichen Programmierstilen umgehen zu können. Es ist robust und kann fehlerhafte Zeilen im Code bei der Interpretation überspringen. GlovePIE kann auch Code generieren. In einer grafischen Oberfläche werden Eingabe und zu emulierende Ausgabe durch Nutzerangaben ermittelt und daraus ein simples Skript erstellt. Der PIE-Code wird mit einer festlegbaren Wiederholungsrate sequentiell ausgeführt.

GlovePIE kann folgende Ausgabesignale erzeugen:

- Maus
- Fake Cursor (mehrere Mauszeiger gleichzeitig)
- Töne (*Musical Instrument Digital Interface* - MIDI)
- Tastatur
- Virtueller Joystick

Das Programm kann dafür Eingaben von über 40 Geräten nutzen. Unter anderem:

- Diverse Joysticks und Gamepads
- Wii-Remote und Zubehör
- Datenhandschuhe
- Trackingsysteme

Von besonderer Relevanz für diese Arbeit ist die Einbindung der Wii-Remote. In der Skriptsprache wird sie über die umgangssprachliche Bezeichnung *Wiimote* angesprochen. Die PIE-Form lautet: *wiimote[Nummer].[zu nutzende Daten oder Button]*.
Ist nur ein Gerät angeschlossen, so kann die Nummerierung entfallen. Die Anzahl ist in GlovePIE auf sieben Stück pro Bluetooth-Empfänger begrenzt [Kenner, 2010, S. 75].
Beim Start eines Skriptes werden die genutzten Wii-Remotes von GlovePIE initialisiert. Dies zeigen die beiden äußeren LEDs an. Leuchten die LEDs in der Mitte, wird zusätzlich das *Motion Plus* kalibriert. In diesem Zeitraum sollte die Wii-Remote nicht bewegt werden. Der Vorgang ist beendet, sobald nur noch eine LED aktiv ist. Sind weniger Wii-Remotes verbunden als im Skript genutzt werden, bietet GlovePIE eine automatische Verbindungsherstellung an, sofern der Microsoft-Treiber genutzt wird.
In GlovePIE wird für die Wii-Remote ein *x,y,z*-Koordinatensystem genutzt:

- *x*-Achse: *pitch*
- *y*-Achse: *yaw*
- *z*-Achse: *roll*

Roll, *pitch* und *yaw* können die in Abbildung 4 beschrieben Werte liefern. Mit *SmoothRoll* und *SmoothPitch* stehen auch geglättete, aber dafür weniger exakte, Werte zur Verfügung.

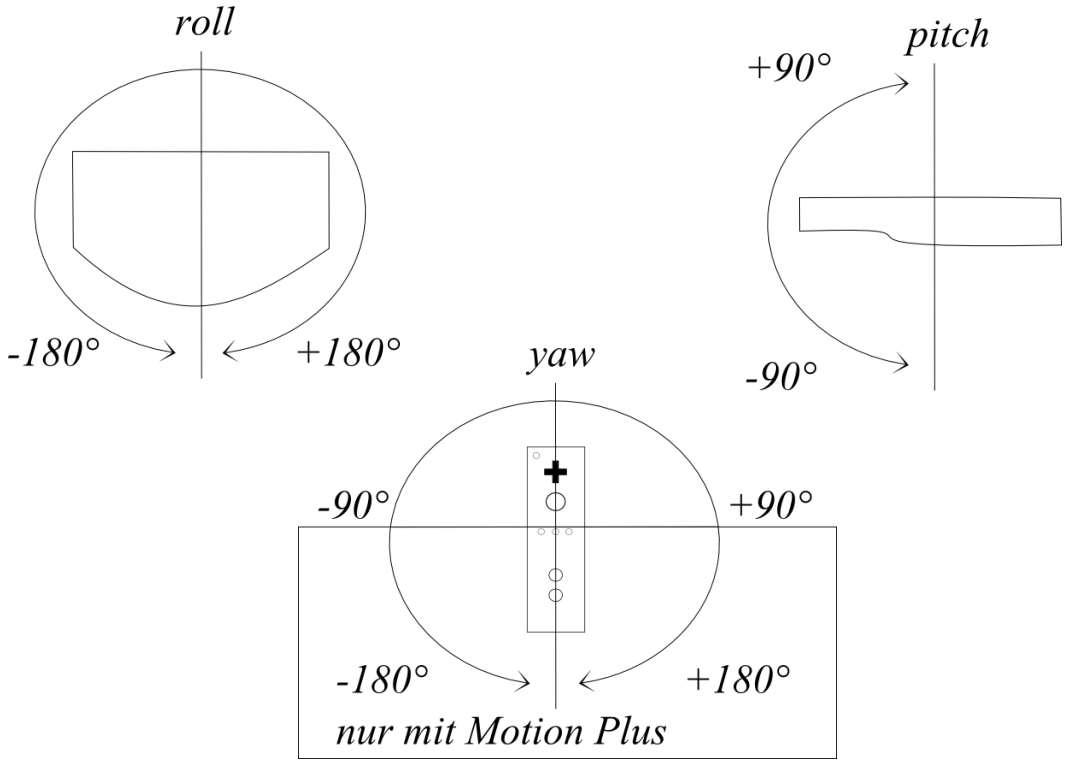

Abbildung 4: *roll, pitch* und *yaw* – Messwerte in GlovePIE

Mit angeschlossenem *Motion Plus* kann durch den Gyrometer zudem die Rotationsgeschwindigkeit dieser drei Bewegungen in Grad pro Sekunde gemessen werden (*YawSpeed, RollSpeed, PitchSpeed*). Die erweiterten *yaw*-Werte sind eine Schätzung auf Grundlage von *YawSpeed*. Die weiteren Werte des *Motion Plus* werden mit *wiimote[Nummer].MotionPlus.[GyroRoll | GyroPitch | GyroYaw]* ausgelesen. Im Gegensatz zu *roll*, *pitch* und *yaw*, sind diese Werte nicht begrenzt. Eine zweifache Linksdrehung um die *z*-Achse liefert beispielsweise *roll* = 0° und *GyroRoll* = -720°.
Die Buttons sind gemäß ihrer Beschriftung beziehungsweise Positionierung am Steuerkreuz benannt (*A, B, Plus, Minus, One, Two, Home, Up, Down, Left, Right*).
Weiterhin ist in der Wii-Remote ein Sensor für die Gravitationsbeschleunigung verbaut. Für dessen Werte stehen *gx*, *gy* und *gz*, die Kurzform als Vektor heißt *g*. Befindet sich die Wii-Remote im freien Fall gilt *g* = [0, 0, 0], weil der Sensor keine Gravitationskraft mehr misst. Wird sie ruhig und flach in der Hand gehalten, so wirkt von unten eine, der Erdanziehungskraft genau entgegengesetzte, Beschleunigung in Höhe von 9,81m/s² auf sie ein. Zu diesem Zeitpunkt ist *gy* = 1. Um *g* in der Einheit Meter pro Quadratsekunde zu erhalten, werden *RawAccX*, *RawAccY*, und *RawAccZ* genutzt.
Diese Messwerte verwendet GlovePIE, um die Beschleunigung ohne Gravitationseinwirkung zu errechnen (*RelAccX*, *RelAccY*, *RelAccZ*). *Rel* steht dabei für relativ, da die Werte sich an der augenblicklichen Haltung orientieren. Das *x,y,z*-Koordinatensystem bewegt sich also in diesem Fall mit der Wii-Remote.
In Verbindung mit der Infrarotleiste kann die Wii-Remote als Zeigegerät genutzt werden. Sie kann bis zu vier LEDs gleichzeitig tracken. Üblicherweise werden in den Infrarotleisten mehrere LEDs als Panel verbaut, um auch in mehreren Metern Entfernung einen ausreichend großen Lichtpunkt für die Infrarotkamera zu bieten. Prinzipiell kann aber jede Lichtquelle genutzt werden, die genug infrarotes Licht abstrahlt und sich so von der Umgebung abhebt. Weitere geeignete Lichtquellen wären somit unter anderem Kerzen oder die LEDs von handelsüblichen Fernsehfernbedienungen. Anderes Licht, zum Beispiel von blauen LEDs, kann wegen des Infrarotfilters vor der Kamera nicht benutzt werden. Von der Kamera erfasste Lichter werden zur Lokalisierung der Fernbedienung in Relation zur Lichtquelle genutzt. Dafür reicht ein einzelner Lichtpunkt aus. Befindet sich der Lichtpunkt etwa im rechten unteren Bereich des Sichtfeldes, so ist die Wii-Remote entweder im linken oberen Quadranten vor der Lichtquelle oder zeigt zumindest dorthin. Diese zwei Fälle können dank des Lagesensors unterschieden werden.
Die Kamera arbeitet mit einer Auflösung von 1024x768 Pixeln. GlovePIE bietet die Messwerte entweder als Position einzelner Lichtpunkte im Pixelraster an (*dot1x*, *dot1y*,

dot2x, …, dot4y) oder bereits auf einen Bereich von Null bis Eins umgerechnet in einem Koordinatensystem (*PointerX, PointerY, PointerXY),* dessen linke obere Ecke der Nullpunkt von *x* und *y*-Achse ist. Es kann gemeldet werden, ob ein bestimmter Lichtpunkt aus dem Sichtfeld verschwindet (*dot1vis*,…) und die Größe eines Lichtpunktes ermittelt werden (*dot1size*,…). Mit der Lichtpunktgröße ist somit auch ein Wert für eine *z*-Achse nutzbar, welche die Entfernung zur Lichtquelle widerspiegelt. Diese Methode ist allerdings nicht sehr genau. Besser funktioniert dies unter Einbeziehung von zwei Lichtquellen, deren Abstand zueinander bekannt sein muss. Werden beide Lichter von der Wii-Remote erfasst, kann mittels Triangulation die Entfernung zwischen Lichtquelle und Wii-Remote berechnet werden. Den Abstand misst GlovePIE in Metern und schreibt ihn in den Vektor *position[3]*. Die vier blauen LEDs der Wii-Remote (*led1*,…) und das *force feedback* (*rumble*) können mittels *true* und *false* einzeln geschaltet werden.

Gestensteuerung wird ebenfalls unterstützt. Ereignisse, wie *SwingUpDownStop*, *CurveRightUp* und *UpsideDownOnShoulder*, werden vom Programm erkannt und liefern zur weiteren Verarbeitung boolesche Werte.

In GlovePIE werden Statusinformationen oder Textausgaben zu Debugzwecken über ein Textfeld ausgegeben (Abbildung 5). Mittels *debug = wiimote.exists* wird darin angezeigt, ob die Wii-Remote erfolgreich angeschlossen wurde und von GlovePIE ausgelesen werden kann. Dazu muss diese Zeile in den Editorbereich geschrieben und die Skriptinterpretation über den RUN-Button gestartet werden. Sollen mehrere Informationen angezeigt werden, sind diese mit einem + zu verbinden.

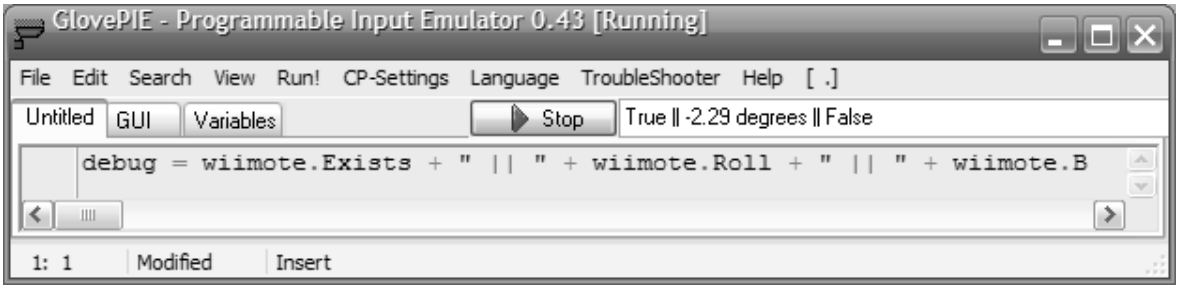

Abbildung 5: Debuginformationen anzeigen in GlovePIE

Die PIE-Sprache hat noch weitere Inhalte, wie bitweise Verschiebung und Spracherkennung, welche aber für die angestrebte Emulation an der VDP nicht relevant sind[7].

[7] Für den vollen Funktionsumfang siehe vorläufige 0.43 Programmdokumentation [Kenner, 2010].
Das Programm kann von http://www.glovepie.org bezogen werden - letzter Abruf am 17.07.2010.

3.2 PPJoy

Das zweite Programm zur Umsetzung der Emulation ist PPJoy. Es steht für nicht kommerzielle Zwecke frei zur Verfügung. Die aktuelle Version ist 8.46.

Seit November 2009 sind neue PPJoy-Versionen unter 64-bit-Systemen lauffähig[8]. Somit funktioniert das Programm auch auf dem Hauptrechner, unter einem 64-bit Windows XP, im VR-Labor stabil. Vor der Installation wurde dies in einer virtuellen Maschine mit dem entsprechenden Betriebssystem getestet.

Der südafrikanische Programmierer Deon van der Westhuysen will mit seinem Programm alte Gamepads und Controller wieder nutzbar machen. Eingabegeräte für den Commodore 64, Atari, ZX Spectrum, die Playstation, frühe Konsolen von Nintendo oder Sega und auch selbst gebaute Geräte, können an heutigen Rechnern wieder genutzt werden. Dazu ist ein paralleler Anschluss und etwas handwerkliches Geschick notwendig. Oft haben alte Geräte einen speziellen Adapter. Die Pins der Controlleranschlüsse werden daher per Draht mit den Kontakten des LPT verbunden und PPJoy stellt einen Treiber bereit, damit die neue Hardware auch als Eingabegerät akzeptiert und erkannt wird. Misslingt dies, versucht PPJoy ein Mapping der Eingabesignale auf einen virtuellen Joystick, der vom System nicht von einem physisch vorhandenen Joystick unterschieden werden kann. Die virtuellen Joysticks können beliebig konfiguriert werden. Bis zu acht Achsen, 32 Buttons und zwei Steuerkreuze sind pro Gerät erstellbar. Die Achsen wiederum können verschiedenen Eingabemethoden zugeordnet werden, beispielsweise *dial, rudder, slider, wheel, x*-Achse und *x*-Rotation [Westhuysen, 2004].

Ein solches virtuelles Gerät kann auch unabhängig von vorhandener Hardware angelegt werden. Andere Anwendungen können diese Geräte dann zur Emulation von Joystickeingaben nutzen.

In GlovePIE lautet die Zugriffsform: *ppjoy[Nummer].[Button oder Achse]*.

Während der Installation von PPJoy erscheinen einige Warnungen, dass der Treiber nicht signiert sei und ein Risiko für das System darstelle. Derartige Warnungen können ignoriert werden. Der Treiber ist lediglich aus finanziellen Gründen nur *test signed*. Digitale Treibersignaturen werden ab $499 pro Jahr verkauft, was Westhuysen nicht für ein Freizeitprojekt aufbringen möchte. Laut Westhuysen würde der einzige Effekt darin bestehen, dass die Systemwarnungen bei der Installation entfallen und das Programm unter

[8] Versionen ab 8.4 bewirken bei Windows 2000 und älter einen Bluescreen beim Systemstart, da seitdem das DDK von Windows 7 für die Weiterentwicklung von PPJoy genutzt wird. Nähere Informationen zur Entwicklung und eine Downloadmöglichkeit für das Programm befinden sich in Westhuysens PPJoy-Blog. Siehe http://ppjoy.blogspot.com/ - letzter Abruf am 08.09.2010.

Windows Vista und Windows 7 nutzbar ist, ohne das Betriebssystem vorher in den Testmodus versetzen zu müssen[9]. Das aktuelle Testrelease aktiviert diese Umstellung bereits automatisch. Die Emulation mit der Wii-Remote kann nach erfolgreicher Installation von GlovePIE und PPJoy beginnen.

3.3 IDO:ImmersiveWorkspace

Das Modul *IDO:ImmersiveWorkspace* der IC:IDO-Software ist für die Navigationskontrolle zwischen Desktop und *Powerwall* zuständig. Es kann über den Menüpunkt *Modules* des Desktopprogrammes aufgerufen werden und ist in vier Rubriken aufgeteilt. Der Reiter *Cluster* dient der Auswahl eines Steuergerätes im Rahmen der *Cluster Configuration* und im Reiter *Navigation* kann die Synchronisation des Desktops mit der *Powerwall* aktiviert und konfiguriert werden. Eine ausführliche Anleitung mit Erklärungen zu allen Optionen befindet sich in [IC:IDO, 2008, S. 74-75]. Die Reiter *Appearance* und *Tracking* sind für die Emulation nicht relevant.

3.4 Umsetzung der Desktopsteuerung

Zunächst soll die Wii-Remote zur Steuerung des Desktoprechners verfügbar gemacht werden. Sobald Maus und Tastatur mit ihr gesteuert werden können, ist auch eine Navigation durch die dreidimensionalen Modelle mit dem Desktopprogramm möglich. Zusätzlich können Navigationsbewegungen am Desktop durch Synchronisation auf die *Powerwall* übertragen werden.

In GlovePIE kann der Mauscursor über zwei Arten manipuliert werden. Zum einen über seine Position auf dem Bildschirm (*CursorPosX, CursorPosY*) und zum anderen durch Emulation der physischen Bewegungen einer Maus (*DirectInputX, DirectInputY*). Bei *CursorPos* ist die linke obere Ecke des Bildschirmes der Nullpunkt. *X*-Achse und *y*-Achse entsprechen der Bildschirmauflösung abzüglich eines Pixels. *DirectInput* misst die Bewegungen der Maus in *dots per inch* (dpi). Diese Unterscheidung ist wichtig, da manche Programme einen eigenen Mauszeiger erstellen und die Eingabesignale mittels *DirectInput* einlesen. Einige Anwendungen frieren den Mauszeiger in der Bildmitte ein, blenden ihn dabei aus und übertragen die Eingabesignale ohne Cursor auf den Bildschirminhalt.

[9] Näheres dazu ist unter http://ppjoy.blogspot.com/2010/03/about-signed-drivers.html in Westhuysens Blog zu finden - letzter Abruf am 08.09.2010.

Die IC:IDO-Software nutzt die Systemmaus. Dadurch kann mit dem Pixelraster des Bildschirms gearbeitet werden. Die Hardware der Wii-Remote bietet zur Umsetzung der zweidimensionalen Bewegungen des Mauscursors mehrere Möglichkeiten.

- Steuerkreuz (oder beliebige andere Buttons)
- Horizontale Bewegungen: *roll*, vertikale Bewegungen: *pitch*
- Cursorpositionierung mit Infrarotleiste und Wii-Remote als Zeigegerät
- Bewegung mittels Beschleunigungssensoren: *g, RelAcc*
- Gyrometer im *Motion Plus*: *PitchSpeed* und *YawSpeed*

```
var.Geschwindigkeit = 10 //Zahl entspricht Pixeln pro Frame
if wiimote.Left then mouse.CursorPosX -= var.Geschwindigkeit
if wiimote.Right then mouse.CursorPosX += var.Geschwindigkeit
if wiimote.Up then mouse.CursorPosY -= var.Geschwindigkeit
if wiimote.Down then mouse.CursorPosY += var.Geschwindigkeit
```

Listing 2: Cursorsteuerung mit Buttons

Die Variante in Listing 2 nutzt die ergonomischen Möglichkeiten der Wii-Remote nicht aus und die Geschwindigkeit des Cursors ist statisch. Die Fernbedienung soll einhändig geführt und außer *B* alle Buttons mit dem Daumen betätigt werden, welcher hier aber am Steuerkreuz gebunden wird. Um Funktionen auszulösen, müsste daher oft die zweite Hand genommen werden. Listing 3 zeigt eine Umsetzung ohne Buttons und mit dynamischer Geschwindigkeitsanpassung.

```
var.G = 40 //Geschwindigkeit bei voller Auslenkung
mouse.CursorPosX += MapRange(Wiimote.Roll, (-90), 90, (-var.G), var.G)
mouse.CursorPosY -= MapRange(Wiimote.Pitch, (-90), 90, (-var.G), var.G)
```

Listing 3: Cursorsteuerung mit *roll* und *pitch*

Die Funktion *MapRange(value, b1, b2, c1, c2)* dient hier der Umrechnung der in Grad gemessenen Werte von *roll* und *pitch* auf ein Pixelmaß für die Cursorbewegung. Der erste Übergabeparameter gibt die zu messende Größe an. Darauf folgen paarweise der erlaubte Eingabebereich und der Zielbereich nach der Umrechnung. Der Cursor wird sich entsprechend des Auslenkungswinkels bewegen, bis die Auslenkung beendet ist. Die Wii-Remote muss bei der Variante in Listing 3 also gerade gehalten werden, wenn keine Bewegung des Cursors gewollt ist. Der Einbau von Pufferzonen kann das zwar erleichtern, dennoch bleibt die Rückkehr in die horizontale und vertikale Ausgangslage nötig.

Besser wäre eine Steuerung, die den Cursor nur während einer Bewegung des Eingabegerätes verändert und ihn entsprechend bei Bewegungslosigkeit verharren lässt.

```
if Wiimote.PointerVisible
  //Die Bildschirmauflösung des Desktops im VR-Labor: 1680x1050
  mouse.CursorPosX = Wiimote.PointerX * screen.Width
  mouse.CursorPosY = Wiimote.PointerY * screen.Height
end if
```

Listing 4: Wii-Remote als Zeigegerät

Die Umsetzung in Listing 4 eignet sich hervorragend zur Nutzung der Wii-Remote als Zeigegerät bei Präsentationen. Der Cursor wird an die Stelle der Bildfläche gesetzt, zu der die Wii-Remote in Relation zur Lichtquelle zeigt. Für einen realistischen Eindruck sollte die Lichtquelle daher direkt unter oder über der genutzten Bildfläche angebracht sein. Die nicht exakte Zeigerichtung, die sich automatisch dadurch ergibt, dass der Zeigemittelpunkt nicht die Bildschirmmitte, sondern die Lichtquelle ist, fällt mit zunehmender Entfernung zu ihr nicht mehr auf. Die optimale Distanz zwischen Wii-Remote und Infrarotleiste liegt bei zwei bis vier Metern.

Nachteil dieser Methode ist der Richtungszwang. Die Wii-Remote muss stets auf die Lichtquelle zeigen. Sie darf weder verdeckt, noch zu weit entfernt sein. Andere Lichtquellen können für Interferenzen sorgen.

Eine Umsetzung mit den Beschleunigungssensoren (*g*, *RelAcc*), wie in Listing 5 gezeigt, ist prinzipiell auch möglich, aber nicht praktikabel. Die Wii-Remote kann zur Cursorsteuerung nicht durchgehend beschleunigt werden. Die Masseträgheit sorgt zusätzlich für einen unberechenbaren Cursor. Für die Gestensteuerung sind diese Messwerte hingegen sehr nützlich, um Stöße und Schwünge zu erkennen. Damit sind die Möglichkeiten einer zubehörlosen Wii-Remote ausgereizt.

```
var.G = 40 //Geschwindigkeit bei maximal zulässiger Beschleunigung
//Erlaubter Eingabebereich für MapRange sind 10 Meter pro Quadratsekunde
mouse.CursorPosX -= MapRange(Wiimote.RelAccX, (-10), 10, (-var.G), var.G)
mouse.CursorPosY += MapRange(Wiimote.RelAccY, (-10), 10, (-var.G), var.G)
```

Listing 5: Cursorsteuerung mit den Beschleunigungssensoren

Das *Motion Plus* eröffnet noch eine weitere Variante. Der Gyrometer liefert mit *YawSpeed* einen Wert für die horizontale und mit *PitchSpeed* einen Wert für die vertikale Bewegung.

Listing 6 zeigt eine Steuerungsvariante mit Gyrometernutzung. Diese Steuerung erwies sich als die angenehmste und wird daher weiter ausgebaut.

```
var.G = 40
var.YawSpeed = wiimote.MotionPlus.YawSpeed
var.PitchSpeed = wiimote.MotionPlus.PitchSpeed
mouse.CursorPosX += MapRange(var.YawSpeed, (-90), 90, (-var.G), var.G)
mouse.CursorPosY -= MapRange(var.PitchSpeed, (-90), 90, (-var.G), var.G)
```

Listing 6: Cursorsteuerung mit dem Gyrometer

Für optimale Steuerungsbedingungen sind noch Anpassungen nötig. Mit dem Code in Listing 6 ist die Cursorbewegung nur zufriedenstellend, wenn die Wii-Remote nicht gekippt wird. Die Bewegungsgeschwindigkeit wird relativ zur Wii-Remote gemessen. Liegt sie etwa um 90° gekippt auf der Seite, ergibt sich *YawSpeed* nunmehr aus der Vertikaldrehung. Benötigt wird somit eine Anpassung der Cursorbewegung in Abhängigkeit von der Haltung der Wii-Remote, wie in Listing 7 gezeigt.

```
if wiimote.Roll < 90 and wiimote.Roll > -90 {
     var.Oben = 1  // Buttons zeigen nach oben
     var.xyAusgleich = MapRange(Abs(wiimote.Roll), 90, 0, 1, 0)
} else {
     var.Oben = -1 //Buttons zeigen nach unten
     var.xyAusgleich = MapRange(Abs(wiimote.Roll), 90, 180, 1, 0)
}
//...
var.YawY = var.Oben * var.YawSpeed * var.xyAusgleich
var.PitchX = var.Rechts * var.PitchSpeed * var.xyAusgleich
var.YawX = -var.Rechts * var.YawSpeed * var.xyAusgleich
var.PitchY = var.Oben * var.PitchSpeed * var.xyAusgleich
mouse.CursorPosX += var.Oben * var.YawSpeed - var.YawY + var.PitchX
mouse.CursorPosY -= var.Oben * var.PitchSpeed - var.PitchY + var.YawX
```

Listing 7: Ausgleichsrechnung für die Gyrometernutzung

Über zwei Auswahlanweisungen wird die Haltung geprüft. Dabei gilt *var.Rechts* = 1, wenn *roll* größer als 0° ist und *var.Oben* = 1, wenn *roll* zwischen -90° und 90° liegt. Für alle anderen Fälle erhalten die Variablen den Wert (-1). Die Variable *var.xyAusgleich* beinhaltet den Ausgleichsfaktor, der sich aus dem Auslenkungswinkel ergibt.

Die Werte der tatsächlich gemessenen Bewegung werden je nach Haltung negiert. Außerdem werden die Bewegungen bei Schräglage anteilig verrechnet.
Im Praxistest erweist sich eine permanent aktive Steuerung mit dem Gyrometer für den allgemeinen Desktopbetrieb als zu empfindlich. Doppelklicks sind kaum möglich. Eine weitere Glättung der Werte bei gedrückter Maustaste erschwert dagegen unter anderem das Aufziehen eines Auswahlrahmens. Daher soll die Cursorsteuerung nur durch Halten des Zeigefingerbuttons *B* aktiviert werden, während mit dem Daumen unabhängig davon alle anderen Buttons bedient werden können. Die somit geschaffene Maussteuerung[10] muss noch um die Desktopsteuerung der IC:IDO-Software ergänzt werden. Die wichtigsten Befehle für die CAD-Steuerung fasst Tabelle 1 zusammen.

Zoom	Shift + Linke Maustaste + Mausbewegung (y)
Drehen	Linke Maustaste + Mausbewegung (x, y)
Objekte verschieben	Strg + Linke Maustaste + Mausbewegung (x, y)
Objekte selektieren	Linke Maustaste
Funktionsmenü	Rechte Maustaste
Vollbild	F11

Tabelle 1: IC:IDO-Software - Desktopsteuerung CAD [IC:IDO, 2008, S. 18]

Die verwendeten Tasten werden nun gemäß Tabelle 2 den Buttons der Wii-Remote zugewiesen. Die linke Maustaste ist die Hauptaktionstaste und Doppelklicks sind für die Navigation in der IC:IDO-Software nicht nötig. Daher wird auf eine gesonderte Aktivierung der alternativen Steuerung verzichtet und die linke Maustaste dem Button *B* zugewiesen. Der Daumen kann auf der Oberseite der Wii-Remote alle anderen Buttons für die Tastenkombinationen erreichen. Da die Visualisierung per Synchronisation auch an der *Powerwall* betrachtet werden soll, ist das Umschalten in den Vollbildmodus notwendig. Die Navigation funktioniert ansonsten nur, solange sich der Mauszeiger im Visualisierungsbereich des Desktopfensters befindet. Dies kann beim Betrachten der *Powerwall* nicht sichergestellt werden.

Wiimote.A	Shift	Wiimote.Minus	Strg
Wiimote.B	Linke Maustaste	Wiimote.Down	Rechte Maustaste

Tabelle 2: Desktopsteuerung - Buttonbelegung der Wii-Remote

[10] Ein Skript für die Wii-Remote als Mausersatz und Rechnerfernbedienung ohne IC:IDO-Softwarebezug befindet sich als wiimote_gyro_desktop.PIE auf dem beiliegenden Datenträger.

Am Desktop ist die Bewegungsfreiheit des Cursors im Rahmen der Bildschirmauflösung definiert, der verwendete dreidimensionale Raum ist jedoch unbegrenzt. Damit die Navigation nicht abrupt endet, wird der Cursor bei Erreichen des Darstellungsrandes auf dessen Mitte zurückgesetzt. Dies geschieht mit den Werten *Screen.Width* und *Screen.Height*, welche immer die aktuelle Auflösung enthalten. Die Werte werden halbiert und den *CursorPos*-Werten zugewiesen. Bei den *CursorPos*-Werten beginnt die Zählung bei Null und sie können maximal den Wert *Screen.Width* – 1 beziehungsweise *Screen.Height* – 1 erreichen.

Im Reiter *Navigation* von *IDO:ImmersiveWorkspace* kann die Projektionsdarstellung (*Immersive Workspace*) als Referenzperspektive (engl. *viewpoint reference*) für die Bewegungen ausgewählt werden. Ist die Standardsteuerung mit IC:Control eingestellt, sind alle Bewegungen relativ zur Cursororientierung, was keine winkeltreue Navigation an der *Powerwall* gegenüber der Desktopsteuerung zur Folge hat. Abhilfe verschafft die Auswahl der Orientierungsunabhängigen *3wall_gamepad*-Konfiguration im Reiter *Cluster*.

Der Aufruf der Skripte kann weiterhin erleichtert werden. *Batch*-Dateien können GlovePIE starten sowie automatisch ein Skript laden und ausführen lassen. Diese Dateien können auch in den Autostart von Windows aufgenommen werden. Damit GlovePIE im Desktopbetrieb keinen Platz in der Taskleiste belegt, kann es zudem mit dem PIE-Skriptbefehl *hidepie* versteckt werden. Mit *exitpie* wird das Skript gestoppt und GlovePIE beendet. Zur Ausführung dieses Befehls bietet sich in den Skripten für die Wii-Remote beispielsweise eine Tastenkombination oder *wiimote.exists* als Bedingung an.

Ein gravierender Nachteil der Desktopsteuerung mit Synchronisation[11] ist, dass der 3D-Cursor nicht über die Mausbewegungen bedient werden kann und es für ihn kein zweidimensionales Substitut an der *Powerwall* gibt.

3.5 Umsetzung der Powerwall-Steuerung

Der 3D-Cursor der VDP ist ein kegelförmiges 3D-Objekt, das als Interaktions- und Zeigewerkzeug in Modellvisualisierungen dient und wird über VRPN angesprochen. Für die IC:Control und Joystickgeräte stehen VRPN-Treiber über das Modul *IDO:ImmersiveWorkspace* im Reiter *Control* zur Auswahl bereit. Die Nutzung der Joystickgeräteklasse ist auf ein Logitech-Gamepad vom Typ *RumblePad 2* abgestimmt. Tabelle 3 zeigt die Belegungen des Gamepads.

[11] Das Skript liegt als IDO-System_wiimote_gyro_desktop.PIE bei.

Button 1	Aktionstaste	Button 8	roll
Button 2	Kontextmenü	Button 9	Datenübertragungskanäle tauschen
Button 5	*-z*	Button 10	Cursor zurücksetzen
Button 6	roll	Linker Analog-Stick	*+/-x, +/-y*
Button 7	*+z*	Rechter Analog-Stick	pitch, yaw

Tabelle 3: Konfiguration Logitech-Gamepad

Diese Konfiguration dient als Emulationsgrundlage. Mit PPJoy wird ein virtuelles Gamepad erstellt, das dem Logitech-Gamepad entsprechend zwölf Buttons, zwei Analog-Sticks und ein Steuerkreuz hat. Die *DirectInput*-Signale des virtuellen Gerätes werden vom VRPN-Treiber akzeptiert und er leitet sie an die Rechner für die Projektorsteuerung weiter. GlovePIE unterstützt die virtuellen Geräte von PPJoy gesondert und kann sie über den Treiberanbieter von anderen Joysticks unterscheiden. Die Achsen der PPJoy-Analog-Sticks werden mit *ppjoy[Nummer].Analog[Nummer]* und die PPJoy-Buttons mit *ppjoy[Nummer].Digital[Nummer]* angesprochen. Die Zählung beginnt jeweils bei Null. Der *Button1* des Gamepads entspricht *Digital0*. Die Stellung eines Analog-Sticks wird in GlovePIE mit einem Wertebereich für die zwei Stick-Achsen von jeweils (-1) bis 1 gemessen. Bei maximaler Auslenkung des ersten Analog-Sticks in den linken oberen Quadranten gilt *Analog0 = Analog1* = (-1), in Normalstellung betragen sie Null. Die zuvor in den Listings 6 und 7 gezeigte Maussteuerung soll weiter Verwendung finden und muss an das Gamepad angepasst werden. Die Steuerung des virtuellen, linken Analog-Sticks wird von den vormals für *CursorPosX* und *CursorPosY* genutzten Werten übernommen, die vorher mittels *MapRange* auf den neuen Wertebereich umgerechnet werden.

Die Buttons des virtuellen Gamepads werden mit Buttons der Wii-Remote belegt. *A* wird der Aktionsbutton, *Minus* öffnet das Kontextmenü, *Home* wird für *reset* genutzt.

Um eine möglichst realitätsnahe Bedienung des 3D-Cursors zu ermöglichen, werden die Rotationsbewegungen mit *GyroRoll*, *GyroPitch* und *GyroYaw* gesteuert. Die einzelnen Messwerte von Gyrometer und Lagesensor beeinflussen sich gegenseitig während der Navigation. Es kann beispielsweise kein *PitchSpeed* ohne eine Veränderung bei *Pitch* und *GyroPitch* geben. Da dies bedeuten würde, dass der 3D-Cursor sich während der Bewegung unablässig dreht und umgekehrt, wird auch hier eine Trennung über den Button *B* vorgenommen.

Damit die Probleme der Desktopsteuerung aus Listing 3 nicht auftreten, werden auch die Messwerte des Lagesensors nachbearbeitet, wie in Listing 8 gezeigt.

```
var.RotationY = Delta(wiimote.MotionPlus.GyroYaw)
ppjoy.Analog2 = MapRange(var.RotationY, (-10), 10, (-1), 1)
```

Listing 8: Rotation um die *y*-Achse mit *Delta(GyroYaw)*

Die Funktion *Delta(x)* hat als Rückgabewert die Veränderung des Übergabeparameters seit dem letzen Frame. Ist der Wert kleiner geworden, so ist *Delta(x)* negativ. Auf diese Weise gilt immer *var.RotationY = 0*, wenn keine Bewegung stattfindet. Entsprechend wird dies für *GyroPitch* umgesetzt. Der virtuelle, rechte Analog-Stick wird anschließend mittels *MapRange* mit einem gültigen Wertebereich angesprochen.
Die Rotation um die *z*-Achse ist ein Spezialfall und muss anders implementiert werden. Für die Rotation um die Längsachse des 3D-Cursors sind am Gamepad zwei Buttons vorgesehen. Die Steuerung über boolesche Werte bedingt eine statische Geschwindigkeit. Eine ansteigende Geschwindigkeit, etwa durch längeres Halten des Buttons, ist in der VDP nicht vorgesehen.

```
if var.RotationZ < (-0.1) {
ppjoy.Digital5 = true
wait 0.4 ms //Millisekunden
ppjoy.Digital5 = false
}
```

Listing 9: Sonderfall - Rotation mit Buttons und *GyroRoll*

Die Variable *var.RotationZ* in Listing 9 wird analog zu *var.RotationY* mit *GyroRoll* berechnet. Eine Pufferzone dient dazu, dass der Button nicht unbeabsichtigt als gedrückt gilt. Wird ein Button im gleichen Frame auf *true* und dann auf *false* gesetzt, gilt er als nicht gedrückt. Daher wird dazwischen eine Wartezeit eingefügt. Für die entgegengesetzte Rotation über Button *Digital7* gilt dies entsprechend.
Für eine Translationsbewegung des 3D-Cursors entlang der *z*-Achse des dreidimensionalen Raumes ergibt sich das gleiche Problem. Sie ist am Gamepad ebenfalls durch zwei Buttons vorgesehen. Zusätzlich fehlt der Wii-Remote ein praktikabler Messwert für die Bewegungsauslösung. Von den Geschwindigkeitsmessungen verbleibt nur *RollSpeed*. Der Einsatz der Infrarotleiste ist nicht geeignet, da die Wii-Remote dazu stets auf die Lichtquelle gerichtet sein muss. Mehrere Infrarotleisten jeweils in der Mitte der Projektionsflächen anzubringen, ist keine akzeptable Lösung. Die bereits vorhandenen LEDs an der Oberseite der *Powerwall* sind für Triangulationsberechnungen zu weit voneinander entfernt. Die Kamera der Wii-Remote kann erst ab einem Abstand von drei

bis vier Metern zwei dieser Lichtquellen gleichzeitig erfassen. Die Lichtpunktgröße beträgt bis einen Meter vor einem LED-Panel den Minimalwert und ist daher zu ungenau. Lichtpunktgrößenmessung und Triangulation sind somit nicht einsetzbar. Der Beschleunigungssensor ist keine Alternative, da mit ihm nur ruckhafte Bewegungen möglich sind. Es verbleibt somit kein Messwert, um in der realen Welt eine Translationsbewegung entlang der *z*-Achse zu messen. Daher wird diese Bewegung auf zwei Weisen implementiert. Eine Variante sind die Buttons *Up* und *Down* am Steuerkreuz, die andere eine Schraubbewegung mit *GyroRoll*. Eine Rechtsdrehung bewegt den Cursor dabei nach vorn. Die Umsetzung erfolgt analog zu Listing 9. Die natürliche Handgelenkdrehung während des Schwingens kann diese Bewegung ungewollt auslösen. Die Navigation an der *Powerwall* und eine Steuerung des 3D-Cursors sind somit über Emulation des Gamepads für die Wii-Remote umgesetzt[12].

3.6 Zwischenfazit

Die Wii-Remote ist ein sehr guter Ersatz für die normale Computermaus. Alle Bewegungen und Funktionen können mit ihr auf verschiedene Weisen umgesetzt werden. Zum einen über die absolute Positionierung des Cursors mit Hilfe der Infrarotleiste, zum anderen über die internen Bewegungssensoren und die Buttons.

Für die Steuerung des 3D-Cursors an der *Powerwall* ermöglicht die Fernbedienung ihrem Nutzer einen wesentlich größeren Bewegungsspielraum vor der Projektionsfläche, da sie nicht extern getrackt wird. Dieser Vorteil ist aber zugleich auch der Nachteil der Wii-Remote. Die Nutzung von Infrarottracking ist aus praktischen Gründen an der *Powerwall* nicht sinnvoll. Dies funktioniert nur an kleineren Flächen zufriedenstellend, die einen fixen Punkt haben, auf den die Wii-Remote ausgerichtet werden kann ohne dem Cursor selbst zusätzliche Rotationswerte zu geben. Die Wii-Remote arbeitet somit nicht mit absoluten Werten, sondern mit den relativen Werten des Lagesensors und des Geschwindigkeitssensors im Zusatzadapter *Motion Plus*. Da diese Werte sich gegenseitig beeinflussen, wird die Steuerung des 3D-Cursors in Translations- und Rotationsbewegungen geteilt. Dieser Behelf hat zur Folge, dass die Ausrichtung des 3D-Cursors nicht immer der korrekten Haltung der Wii-Remote gegenüber der *Powerwall* entspricht, weil deren tatsächliche Positionierung ohne externes Tracking unbekannt ist. Ohne infrarote Lichtquellen kann die Wii-Remote keine Abstände messen. Es fehlt daher

[12] Das Skript liegt als 3wall_gamepad_wiimote.PIE bei. Zugehörige Clusterkonfiguration: 3wall_gamepad.

ein Sensorwert für die Translationsbewegung des 3D-Cursors entlang der *z*-Achse in die Tiefe des dreidimensionalen Raumes. Diese Bewegung konnte mit der Umsetzung als Schraubbewegung ersetzt werden. Das größte Problem stellt allerdings die Schnittstelle zwischen Wii-Remote und VDP dar. Einige Funktionen sind auf dem verwendeten Gamepad-Plugin den Buttons zugeordnet, was dynamische Bewegungen erschwert. Über verschiedene Behelfe kann dieser Mangel allerdings zufriedenstellend behoben werden.
Dennoch wäre eine Schnittstellenänderung oder ein direkter Anschluss der Wii-Remote an die VDP besser. Dafür sind Kenntnisse der Interna von VRPN-Anwendungen notwendig.

4 Virtual Reality Peripheral Network

Das *Virtual Reality Peripheral Network* (VRPN) ist eine Entwicklung der *University of North Carolina at Chapel Hill*. Der Initiator des Projektes Russel M. Taylor II verfolgt damit seit 1998 die Bereitstellung einer geräteunabhängigen und netzwerktransparenten Schnittstelle zu VR-Eingabegeräten[13].

4.1 Grundlagen

Werden in einer Anlage mit mehreren Displays oder Projektoren einige Computer gleichzeitig mit der Darstellung der VR-Umgebung betraut, so müssen sie alle Zugang zu den Eingaben des Steuergerätes haben. VRPN stellt einen Geräteserver zur Verfügung, bei dem sich beliebige Rechner im lokalen Netzwerk oder über das Internet registrieren können und dann mit den Eingabedaten beliefert werden. Somit entfällt eine mehrfache Verkabelung des Steuergerätes und die Darstellungsfläche kann ohne Neustart des Servers jederzeit variieren. Das VRPN-Projekt ist *public domain software* und somit gemeinfrei, da die Entwickler auf jegliches Urheberrecht verzichten. Die Offenlegung des Quellcodes hat zu vielen Verbesserungen und Ergänzungen des Projekts durch die Hilfe von Freiwilligen und Unternehmen geführt[14]. In der aktuellen Version 7.26 sind dadurch VRPN-Treiber für diverse Geräte enthalten. Das Projekt ist in C und C++ entwickelt worden. Kommende Versionen sollen unter der *Boost Software License 1.0* veröffentlicht werden.
VRPN kann auf diversen Betriebssystemen genutzt werden. Es besteht aus einer Sammlung von Geräteklassen und Geräteservern. Die Kommunikation zwischen Server

[13] Projektwebseite: http://www.cs.unc.edu/Research/vrpn/ - letzter Abruf am 12.08.2010.
[14] Auch IC:IDO unterstützt: http://lists.unc.edu/read/messages?id=5543928 - letzter Abruf am 12.08.2010.

und Client findet über einheitliche Schnittstellen statt, da jedes Steuergerät in mindestens eine von fünf allgemeinen Geräteklassen eingeordnet wird [Taylor et al., 2001, S. 2]:

- Tracker (Geschwindigkeit, Beschleunigung, Position und Haltung)
- Button
- Analog
- Dial
- ForceDevice

Statt die Eingaben der Geräte direkt an die Clients zu liefern, können sie auch als Input für andere Geräteobjekte genommen werden (engl. *layered device*) [Taylor et al., 2001, S. 2]. Eines dieser Objekte heißt *AnalogFly*. Es ist ein Tracker-Objekt, dessen Positions- und Haltungsdaten serverseitig von beliebigen Geräten bezogen werden. Die verwendeten Geräte sind für Clients nicht sichtbar, da diese lediglich ein *AnalogFly*-Objekt registrieren. Durch dieses VRPN-Gerätemanagement sind keine Änderungen an den Clients notwendig, wenn das Steuergerät gegen eines mit den gleichen Geräteklassen ausgewechselt wird. Zum Datenaustausch werden Callback-Funktionen benutzt. Der Client muss Objekte der gewünschten Geräteklassen anlegen und dem Konstruktor einen String übergeben, der den Namen des Gerätes am Server und dessen Netzwerkadresse beinhaltet, beispielsweise *Tracker0@localhost*. Das Objekt registriert sich dann beim angegebenen Server für das gewünschte Gerät. Optional kann zusätzlich eine bereits bestehende Verbindung und der Name einer alternativen Gerätekonfigurationsdatei übergeben werden. Wird dies ausgelassen, so hat der Konstruktor dafür NULL als Vorgabewerte. Der Client muss das Geräteobjekt abfragen (engl. *polling*), um den aktuellen Zustand des Steuergerätes zu erfahren. Diese Verfahrensweise ist effektiv, aber verursacht gleichzeitig erhöhte Last durch die häufigen Methodenaufrufe. Effizienter wäre eine Ereignissteuerung, bei der sich das Geräteobjekt automatisch meldet, sobald sich der Zustand des Steuergeräts ändert. Der Server aktualisiert den Objektzustand in der Voreinstellung mit einer Rate von 1kHz und damit oft schneller als es native Gerätetreiber am lokalen Rechner schaffen (60Hz) [Taylor et al., 2001, S. 5]. Die Daten der Eingabeelemente, wie die einzelnen Achsen eines Analog-Sticks, werden immer an der gleichen Stelle des versandten Vektors platziert. Diese Positionen werden in VRPN *channel* genannt. Für Buttons sind die Zustände *pressed* und *released* vorgesehen und werden gesondert übertragen. Die Button-Meldungen werden bei dem jeweiligen Eintreten des Ereignisses als TCP (engl. t*ransmission control protocol*)-Paket an den Client geschickt. Werte der Achsen dagegen als UDP (engl. *user datagram protocol*)-Paket, da dies eine unsicherere, aber schnellere Kommunikation

ermöglicht. Die Achsenzustände werden fortlaufend aktualisiert, Button-Meldungen sind hingegen seltener und dürfen daher nicht verloren gehen.
Um Anwendungen aus dem VRPN SDK zu kompilieren sind einige Anpassungen notwendig. Es wird als ZIP-Archiv auf der Projektseite angeboten und liegt darin als vorkonfiguriertes Projekt für *Microsoft Visual Studio 2005* vor[15]. Die fertige Konfiguration erweist sich teilweise als hinderlich, da Includeverzeichnisse und Bibliotheksreferenzen die Ordnerstrukturen der Entwickler nutzen. Und dies findet sich nicht nur in den Projekteigenschaften, sondern ebenfalls hart kodiert im Quellcode. Teile des Projekts verweisen auf Dateien, deren zugehörige SDK-Versionen nicht mehr angeboten werden. Die Kompilierung des VRPN-Projektes und der benötigten Ressourcen ist aber auch mit aktuellen SDKs möglich. Für diese Arbeit wurden folgende SDKs verwendet:

- Windows Driver Kit 7.1.0.7600
- Microsoft DirectX SDK (June 2010)

Diese Pakete stellen umfangreiche Entwicklungsbibliotheken und Werkzeuge bereit. Vom *Driver Development Kit* (DDK) werden die *Build Enviroments* und von *DirectX* die Header und Bibliotheken benötigt[16]. Es werden die Standardinstallationspfade genutzt. Um Sound und grafische Ausgaben auf den Geräten der Clients anzusteuern, sind zusätzliche Bibliotheken nötig, die im Rahmen dieser Arbeit nicht verwendet werden.
In der Projektmappe ist folgendes Includeverzeichnis bei den Projekten *vrpn*, *vrpndll*, *timecode_generator*, *vrpn_phantom*, *vrpn_server* und *vrpn_print_devices* zu ergänzen:

- $(SYSTEMDRIVE)\Programme\Microsoft DirectX SDK (June 2010)\Include

Ein weiteres Bibliotheksverzeichnis wird für *vrpndll*, *vrpn_server* und *vrpn_print_devices* zusätzlich vom Linker benötigt:

- $(SYSTEMDRIVE)\Programme\Microsoft DirectX SDK (June 2010)\Lib\x86

Ein VRPN-Treiber für die Wii-Remote ist im SDK bereits vorhanden (*vrpn_WiiMote.C*, *vrpn_WiiMote.h*). Er bezieht seine Daten über die DLL des *wiiuse*-Projektes, die beim Start eines Servers in dessen Verzeichnis vorliegen muss. Zur Kompilierung der *wiiuse.dll*

[15] Download: ftp://ftp.cs.unc.edu/pub/packages/GRIP/vrpn/ - letzter Abruf am 12.08.2010.
[16] Von der Installation weiterer Bestandteile des *Driver Kits* wird abgeraten. Das *Device Simulation Framework* beschädigt die Windows XP-Installation.

werden die HID-Bibliotheken des DDKs benötigt. Es wird ebenfalls eine bereits kompilierte Version angeboten. *Wiiuse* ist unter der LGPL-Lizenz veröffentlicht[17].

Wiiuse fehlt eine Unterstützung für das *Motion Plus* und in der Folge auch dem VRPN-Treiber. Daher ist die angestrebte Steuerung darüber nicht erreichbar. Eine Umsetzung für das Standard-VRPN soll dennoch kurz gezeigt werden.

VRPN stützt sich zur internen Konfiguration stark auf Präprozessordirektiven. Die Projekteigenschaften werden in der Headerdatei c*onfigure.h* mit bedingter Kompilierung und dem Setzen von Flags festgelegt. Um die Bereiche der Wii-Remote in den zu kompilierenden Code aufzunehmen, müssen dort die Flags *VRPN_USE_DIRECTINPUT* und *VRPN_USE_WIIUSE* gesetzt werden. In den damit aktivierten Codebereichen sind einige Anpassungen für *wiiuse* und *DirectX* notwendig.

Die für die benötigten Belange fertig angepassten Projekte der Projektmappe können nun kompiliert werden. Zum Testen werden der Server *vrpn_server* und der Client *vrpn_print_devices* verwendet (Abbildung 6). Referenzierte Nebenprojekte, beispielsweise die *vrpn.dll*, werden von Visual Studio automatisch mit kompiliert. Der entstandene Server kann alle in VRPN spezifizierten Geräte, darunter nun auch die Wii-Remote, in die dazugehörigen Geräteklassen einteilen und auslesen. Der Client gibt die Zustände des beim Aufruf angelegten Geräteobjektes aus. Für den Server wird zusätzlich eine Konfigurationsdatei, standardmäßig *vrpn.cfg*, benötigt. In dieser wird unter anderem festgelegt, welche Geräte von ihm gelesen werden sollen und wie sie benannt werden.

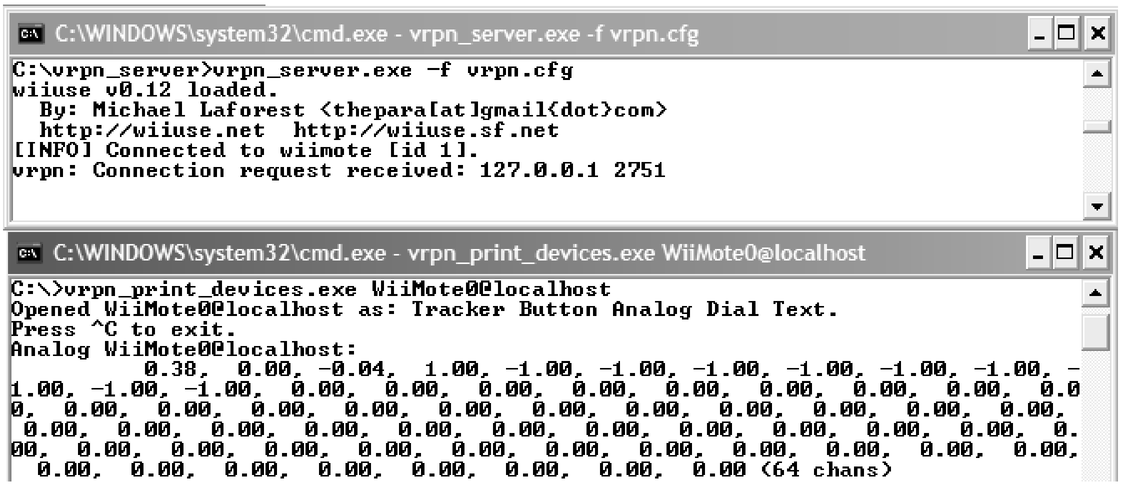

Abbildung 6: VRPN Server und Client für die Wii-Remote

---

[17] Siehe http://de.wikipedia.org/wiki/GNU_Lesser_General_Public_License - letzter Abruf am 13.08.2010. Bezugsquelle von *wiiuse*: http://wiiuse.sourceforge.net/ - letzter Abruf am 13.08.2010.

Die Zuordnung der *channel* zu einzelnen Navigationsfunktionen erfolgt durch die Abnehmerapplikation auf der Clientseite und wird von VRPN nicht spezifiziert. Daher ist über den VRPN-Code auch keine Änderung im Mapping für das Gamepad möglich.
Die Wii-Remote ist ohne *Motion Plus* kein ausreichend gutes Steuergerät für die VDP. Aber das Konzept des *layered device* bietet eine neue Alternative für den Anschluss. Nachfolgend wird untersucht, ob dieses Konzept als Schnittstelle zwischen Wii-Remote und VDP über Emulation eines erweiterten Gamepads benutzt werden kann. Dafür sind zunächst Kenntnisse der Umsetzung von VRPN durch IC:IDO notwendig.

4.2 Nutzung von VRPN durch IC:IDO

Die Software von IC:IDO basiert auf VRPN. Auf dem Hauptrechner wird der *IDO:DeviceServer* ausgeführt und liest über beigefügte VRPN-Treiber die Eingabegeräte aus. Die Eingabedaten werden dann in ein *shared memory* geschrieben, auf das die VDP über das Plugin *IDODevice* zugreift [IC:IDO, 2006, S. 117].
Weitere Treiber für den *IDO:DeviceServer* können dem VRPN-Projekt entnommen und müssen gemäß [IC:IDO, 2006, S. 118] an die Plattform angepasst werden. Die resultierende DLL oder Shared-Library wird danach durch Kopieren zu den Gerätetreibern der Software hinzugefügt[18]. Die neuen Geräteklassen müssen darauf noch in der gerätespezifisch anzulegenden *idoDeviceServer.ini* konfiguriert und bei *layered devices* die Eingabesignale den Navigationsfunktionen zugewiesen werden.
Der *IDO:DeviceServer* ist ein separater Prozess und kann über das Tool *idoDeviceSpy* einzeln gestartet werden[19]. Das Tool bietet eine grafische Oberfläche für Gerätetests, welche die Zustände aller Geräteinstanzen aus der *idoDeviceServer.ini* anzeigt.

4.3 Nutzung der Wii-Remote

Die Implementierung über Emulation eines Gamepads mit GlovePIE hat noch Optimierungspotential. Insbesondere sind dynamische Werte für eine fließende Translation auf und Rotation um die *z*-Achse wünschenswert. PPJoy hat Kapazitäten für bis zu acht Achsen, von denen bisher nur vier genutzt werden. Um diese mit Navigationsfunktionen belegen zu können, bedarf es allerdings einer Änderung im Mapping des Gamepad-Treibers. Würden außerdem die Eingaben des Lagesensors beziehungsweise der

[18] [IDOKernel-Path]\plugins\IDODevice\driverPlugins\[DriverName]\libIDODevice[DriverName](.so|.dll)
[19] C:\Programme\IDO-System\IDOExplore\2006_SP3_4\Release\idoDeviceSpy.exe

zugehörigen virtuellen Analog-Sticks als absolute Werte innerhalb des Wertebereiches interpretiert werden, könnte die Haltungsbestimmung präziser erfolgen. Für derartige Einstellungen muss die Eingabegerätekonfiguration abgeändert werden.

Das Mapping der Eingabesignale auf die einzelnen Cursorbewegungen erfolgt für *layered devices* und die dazugehörige Clusterkonfiguration in der jeweiligen *idoDeviceServer.ini*. Diese Konfigurationsdateien befinden sich im Systemkonfigurationsordner von *IDO:Explore*[20]. Die Zustände der Achsen (*channel*) und Buttons werden darin den Navigationsfunktionen zugewiesen. Welche Änderungsoptionen diese Dateien bieten, ist in keinem der vorliegenden IC:IDO-Dokumente dokumentiert.

In der *idoDeviceServer.ini* werden die Instanzen für die gewünschten Geräteobjekte angegeben, die beim Start angelegt werden sollen. Für das vorhandene Gamepad-Plugin sind dies *Joystick0* und *IDOAnalogFly0*. Die Zahl hinter dem Namen dient der Unterscheidung bei mehreren Geräten gleichen Typs. *Joystick0* wird *DirectXFFJoystick* als Treiber zugewiesen. Dieser bezieht seine Eingabedaten über *DirectInput* und bietet *force feedback*. Das Geräteobjekt *IDOAnalogFly* ist eine spezialisierte Version von *AnalogFly* und hat dem Gamepad entsprechend nur Belegungen für vier *channel* und somit für zwei Analog-Sticks. Die Restlichen Navigationsfunktionen sind Buttons zugewiesen. *IDOAnalogFly* wird als *layered device* zum Mittler zwischen Gamepad und VDP.

```
[Device.IDOAnalogFly0.Buttons]
Name = Joystick0
RotationAxisZ = [ 5 7 ]
TranslateAxisZ = [ 4 6 ]

[Device.IDOAnalogFly0.AxisY]
Name = Joystick0
Channel = 1
Offset = 0.0
Thresh = 0.05
Scale = 2.0
Power = 4.0
```

Listing 10: Die *idoDeviceServer.ini* des Gamepads (Auszug)

IDOAnalogFly bietet neben den in Listing 10 angegebenen Funktionen noch *AxisX*, *RotationX* und *RotationY*, denen die jeweiligen Achsen zugeordnet sind.

[20] Auf dem Hauptrechner im VR-Labor: C:\Programme\IDO-System\IDOExplore\sysConfig\immersive\

Mehr als vier Achsen sind im Geräteobjekt *IDOAnalogFly* nicht vorgesehen, wodurch weitere Einträge, wie *AxisZ* oder *RotationZ*, in der Konfigurationsdatei ignoriert werden. Auch wenn die Werte des neuen Gamepads ausgelesen werden können, so erfolgt dennoch kein Mapping auf den 3D-Cursor über *IDOAnalogFly*.
Da das *AnalogFly*-Geräteobjekt sechs Achsen unterstützt, soll es anstelle von *IDOAnalogFly* genutzt werden und die *idoDeviceServer.ini* wird entsprechend angepasst.
Zunächst werden *Joystick0* und *AnalogFly0* angelegt und die *channel* von *Joystick0* den Navigationsfunktionen für Translation und Rotation von *AnalogFly0* zugewiesen (Tabelle 4). Anschließend muss in der *avatarDevices.ini* im gleichen Verzeichnis die *AnalogFly0* als zu nutzendes Eingabegerät für *Tracker.RightHand* angegeben werden.

PPJoy	VRPN channel	Device.AnalogFly0.[Funktion]
Analog0	0	AxisX
Analog1	1	AxisY
Analog2	2	AxisZ
Analog3	5	RotationX
Analog4	3	RotationY
Analog5	4	RotationZ

Tabelle 4: *AnalogFly* als *layered device*

Zwar kann nun jede Bewegung dynamisch erfolgen, aber die Rotationen bleiben weiterhin schlecht nutzbar und beeinflussen zudem die Translationen. Die Bewegungen über *AnalogFly* werden relativ zur Haltung des Cursors ausgeführt. Ohne externes Tracking erschwert dies daher die Steuerung des relativ positionierten Cursors. Bei *IDOAnalogFly* ist dieses Problem nicht vorhanden, da nur zwei dynamische Translationen vorgesehen sind und der Cursor damit lediglich auf einer vertikalen Ebene bewegt wird, unabhängig von seiner Orientierung. *AnalogFly* kann Eingaben als absolut ansehen, allerdings gilt diese Einstellung jeweils für die komplette Instanz. Es wäre nur sinnvoll, wenn die Wii-Remote zusammen mit der Infrarotleiste genutzt werden würde, denn die sonstigen Sensoren liefern keine Informationen darüber, wo sich die Wii-Remote befindet, sondern ermitteln lediglich, wie sie sich bewegt hat. Die Infrarotleiste ist, wie bereits dargelegt, am bestehenden System nicht praktikabel einsetzbar. Darum wird *AnalogFly1* angelegt. Während *AnalogFly0* für die relativen Translationen zuständig ist, soll *AnalogFly1* die absoluten Haltungswerte vom Lagesensor nutzen. Das Tool *idoDeviceSpy* liefert zufriedenstellende Ergebnisse für die einzelnen Instanzen. Allerdings scheitert dieser Versuch bei der Datenabnahme durch die VDP, da in der *avatarDevices.ini* nur genau ein

Eingabegerät angegeben werden kann und bei einem Umlegen der absoluten *AnalogFly1*-Werte auf *AnalogFly0* die absolut-Eigenschaft überschrieben wird. Die gewünschten Verbesserungen sind somit nicht mit den gegebenen Mitteln durchführbar. Der Cursor lässt sich daher über *IDOAnalogFly* in der *3wall_gamepad*-Konfiguration angenehmer steuern.

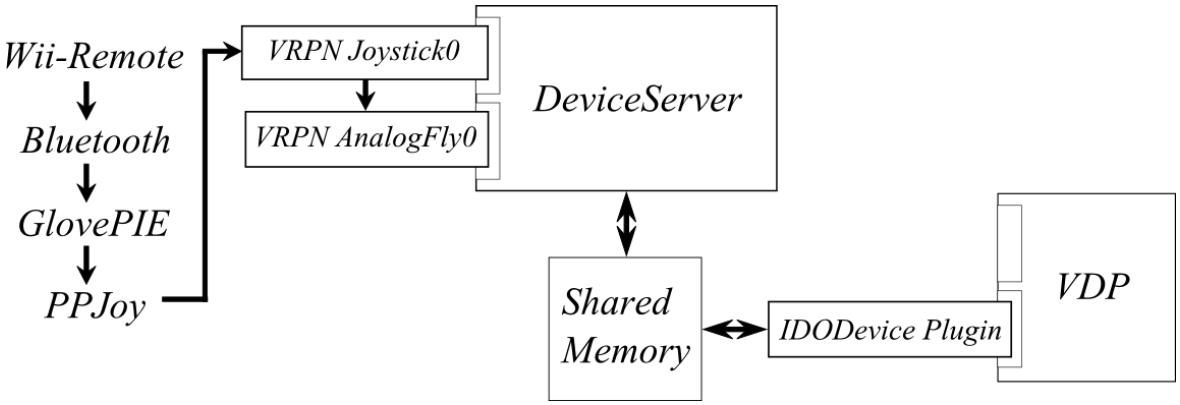

Abbildung 7: Weg der Eingabedaten (in Anlehnung an [IC:IDO, 2006, S. 117])

Da es keinen VRPN-Treiber für das *Motion Plus* gibt, müssen für alle Anwendungsfälle an der VDP die Programme GlovePIE und PPJoy zwischengeschaltet werden (Abbildung 7). Die neu erstellte Konfiguration mit *AnalogFly* kann in der Geräteauswahl von *IDO:ImmersiveWorkspace* ausgewählt werden (Abbildung 8) [21].

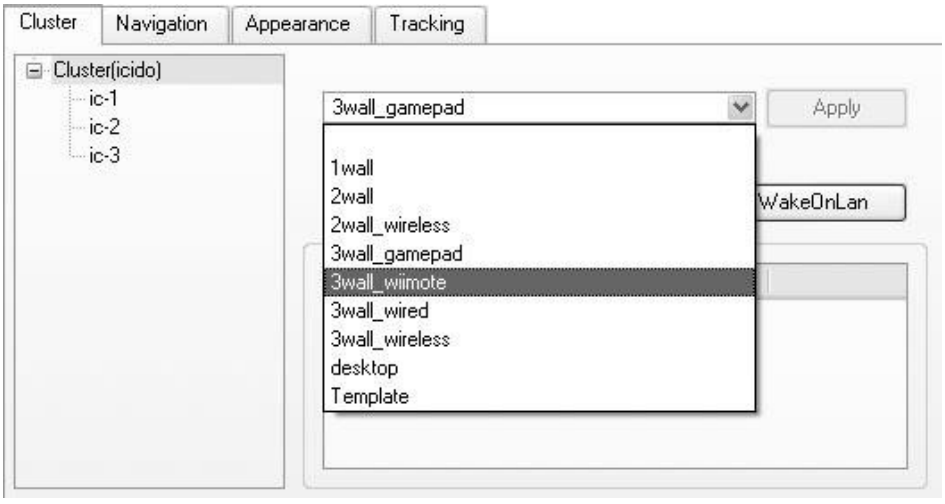

Abbildung 8: Auswahl der Wii-Remote als *AnalogFly-device*

[21] Das zugehörige Skript für GlovePIE liegt als 3wall_wiimote.PIE bei. Die vorhandene PPJoy-Konfiguration funktioniert für alle gezeigten Emulationsvarianten.

5 Zusammenfassung und Ausblick

Die Wii-Remote ist ein hervorragendes Steuergerät für viele Einsatzbereiche. Im alltäglichen Desktopbetrieb ebenso wie in virtuellen Realitäten. Mit zunehmender Komplexität der Anforderungen gelangt aber selbst sie an ihre Grenzen. Bei ansteigender Größe der Darstellungsfläche und zusätzlicher Bestimmung der Orientierung des Gerätes, ist das interne Infrarottracking nicht mehr ausreichend. Behelfe über die restlichen Sensoren ermöglichen zwar weiterhin eine Nutzung, komplett intuitiv kann diese aber nicht mehr sein. In Ermangelung eines vollständigen VRPN-Treibers für das *Motion Plus* muss die Wii-Remote über die Emulation eines virtuellen Gamepads an die VDP angeschlossen werden. An der Anlage des Virtual Reality-Labors hat sich die Emulation über das vorhandene Gamepad-Plugin bewährt. Darüber kann eine Steuerung geboten werden, die der IC:Control weitestgehend ähnelt – und dies mit kostenfreier Software und Hardwarekosten für Wii-Remote, Zusatzadapter und Bluetooth-Stick von unter 50€.

Die vorhandene Anlage des Virtual Reality-Labors hat noch Entfaltungsspielraum. Die Software für das optische Tracking ist nicht an IC:IDO gekoppelt, sondern kann auch mit zahlreichen anderen Softwarelösungen, wie *VR Juggler* oder *OpenScenegraph* eingesetzt werden. Die VDP selbst stellt auch ein Handmodell als Cursor bereit, das als Avatar für Datenhandschuhsteuerungen herangezogen werden kann. IC:IDO bietet bereits einen Handschuh mit optischem Tracking über *DTrack* an.

Die Wii-Remote hat die Konkurrenz unter den Konsolenherstellern erneut entfacht. Sony und Microsoft haben eigene Steuergeräte entwickelt und zur Marktreife gebracht, die durch ihr externes Tracking ebenfalls viel Potential für virtuelle Realitäten bieten (*Playstation Move, Kinect*). Die Entwicklungen zeigen einen deutlichen Trend, weg von der reinen Bedienung eines Handgerätes, hin zu mehr Bewegungsfreiheit im Raum. Vielleicht sind in einigen Jahren generell keine Steuergeräte mehr in den Händen der Nutzer nötig und die Steuerung erfolgt nur noch mit der Kraft der eigenen Gedanken[22].

[22] Siehe Brain Computer Interface Technology, http://emotiv.com/ - letzer Abruf am 08.09.2010

Literaturverzeichnis

[A.R.T., 2010] Advanced Realtime Tracking GmbH: Interfaces: Supported Software, URL: http://www.ar-tracking.de/Supported-Software.29.0.html, letzter Abruf am 23.06.2010.

[IC:IDO, 2006] IC:IDO: IDO:Develop Programmers Guide, ICIDO GmbH, Stuttgart, 2006.

[IC:IDO, 2008] IC:IDO: IDO:Explore Benutzerhandbuch, ICIDO GmbH, Stuttgart, 2008.

[Kenner, 2010] Carl Kenner: GlovePIE 0.43 Preliminary Documentation, http://glovepie.org/, 2010.

[Microsoft, 2010] Microsoft Corporation: Introducing Kinect, URL: http://www.xbox.com/en-US/kinect, letzter Abruf am 23.06.2010.

[Taylor et al., 2001] Russel M. Taylor II et al.: VRPN: A Device-Independent, Network-Transparent VR Peripheral System, UNC, Chapel Hill, 2001.

[USB, 2001] USB Implementers' Forum: Device Class Definition for Human Interface Devices (HID), USB.org, 2001.

[WC, 2010] WiiChat: Webpagecontent, URL: http://www.wiichat.com/attachments/nintendo-wii-chat/324-wii-dashboard-wii_remote5view_0501.jpg, letzter Abruf am 23.07.2010.

[Westhuysen, 2004] Deon van der Westhuysen: PPJoy Documentation, http://ppjoy.blogspot.com/, 2004.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Diese Arbeit hat in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Leipzig, 23. September 2010

Christoph Jobst