Mein Praxissemester bei der **IBM Deutschland Research & Development GmbH**

Markus Schnalke
MatNr: 039131

Dies ist der Bericht zu meinem zweiten Praxissemester im Studiengang Wirtschaftsinformatik an der Hochschule Ulm. Ich absolvierte dieses Praktikum in Böblingen bei der ***IBM Deutschland Research & Development GmbH****.*

Inhaltsverzeichnis

1 Einleitung

1.1 Vorwort

Als Fachhochschulstudent darf ich im Laufe meines Studiums zwei Praxissemester belegen. Dies ist eine großartige Möglichkeit um das in den Vorlesungen gelernte theoretische Wissen in praktisches Wissen und Erfahrung umzusetzen. Zudem erhält man so Einblicke in die Abläufe in Unternehmen.

Mein erstes Praxissemester habe ich bei der *MAKA — Max Mayer Maschinenbau GmbH* in Nersingen absolviert. Während den sechs Monaten dort habe ich sehr viel gelernt, das mir in den nachfolgenden Semestern nützlich war.

Für mein zweites Praxissemester wählte ich eine etwas andere Art von Unternehmen. Wo ich bei *MAKA* in einem mittelständischen Industrieunternehmen in einer kleinen IT-Abteilung arbeitete, wollte ich nun in einem großen IT-Unternehmen in mitten hunderten von Entwicklern tätig sein. *IBM* bot genau das.

Zum Inhalt. In diesem Bericht werde ich als erstes das Unternehmen, das Labor in Böblingen und die Abteilung in der ich aktiv war, vorstellen. Anschließend beschreibe ich das Projekt und seine Rahmenbedingungen. Danach gehe ich auf die von mir geleisteten Arbeiten ein. Als nächstes beschreibe ich unsere Auftritte innerhalb des Unternehmens und an der Öffentlichkeit. Abschließend werde ich noch ein paar Eindrücke aus dem Praxissemester aufgreifen und ein Fazit ziehen.

2 Das Unternehmen

Die drei Buchstaben "*IBM*" und das Logo mit den den blauen Streifen (Abbildung 2.1) sind fast jedem ein Begriff. Dass das Unternehmen "mit Computern zu tun" hat, ist bekannt, was aber genau die *IBM* macht, weiß die Mehrheit der Bevölkerung nicht.

In den folgenden Seiten möchte ich dieses Unternehmen, bei dem ich nun ein halbes Jahr gearbeitet habe, vorstellen.

Abbildung 2.1: Das seit 1972 verwendete Unternehmens-Logo.

2.1 Geschichte

Das Unternehmen wurde 1896 von Herman Hollerith, damals noch unter dem Namen *Tabulating Machine Company*, in Broome County, New York gegründet. Zu dieser Zeit wurden Maschinen zum Auslesen von Lochkarten produziert.

Knapp dreißig Jahr später, 1924, war die Produktpalette gewachsen und gefertigt wurden nun die verschiedensten Geräte für den Gebrauch im Unternehmen. In Folge dessen änderte das Unternehmen seinen Namen in *International Business Machines*.

Als die ersten Computer auftauchten war *IBM* am forderster Front mit dabei und hatte schnell eine dominierende Stellung im Bereich der Mainframes[1] inne.

Ab den 80er Jahren wandelte sich der Markt —Computer waren klein und für Privatpersonen erschwinglich geworden— und das Unternehmen hatte mit dem *IBM-PC* eine Antwort auf den *Apple II* der sehr erfolgreich war. Ein Jahrzehnt später verlor die *IBM* dann aber ihre Vorherrschaft auf dem Gebiet der Personal Computer an die Konkurrenz. Im Jahr 2005 verkaufte das Unternehmen seine PC-Sparte dann komplett an den chinesischen Hersteller *Lenovo*.

Heutzutage liegt die Priorität des Unternehmens stärker auf Dienstleistungen wie Beratung.

[1] Großrechner — komplexe Computersysteme mit sehr großer Robustheit, Verlässlichkeit und Leistung

2.2 Unternehmensgrundsätze

Schon 1935 stellte der damalige Direktor Thomas J. Watson die Position des Unternehmens bezüglich der Gleichberechtigung von Frauen und Männern klar: “Männer und Frauen werden für gleiches Geld die gleiche Arbeit verrichten. Sie werden gleich behandelt, die gleiche Verantwortung übernehmen und die gleichen Entwicklungschancen erhalten.”. 1953 folgte von höchster Stelle der Grundsatz, dass Mitarbeiter allein nach ihren Fähigkeiten beurteilt werden sollen, “ohne Rücksicht auf Rasse, Hautfarbe oder Glaubensbekenntnisse”. Sexuelle Orientierung wurde 1984 hinzugefügt.

Auf Grund seiner innovativen Unternehmenskultur kann *IBM* ein gutes Arbeitsklima und eine sehr geringe Kündigungszahl wegen Unzufriedenheit aufweisen.

Um die Grundsätze der *IBM* zu formulieren wurde ein Aufruf für Vorschläge an alle Mitarbieter des Unternehmens gemacht. Die Einsendungen wurden gesammelt, zusammengefasst und destilliert. Das Ergebnis sind folgende drei Sätze:

- Engagement für den Erfolg jedes Kunden
- Innovationen, die etwas bedeuten – für unser Unternehmen und für die Welt
- Vertrauen und persönliche Verantwortung in allen Beziehungen

Diese Gedanken sollen die Basis aller Tätigkeit im Unternehmen sein.

2.3 Unternehmensbereiche

Global Business Services Unternehmensberatung. Umsatzmäßig der größte Bereich.

Global Services Outsourcing und e-Business Anbieter. Der zweite große Bereich, in dem mehr als die Hälfte der Mitarbieter beschäftigt sind.

Systems and Technology Group Entwicklung und Vertrieb von IT-Infrastruktur. Darunter fallen die Serversysteme (System i, System x, System p, System z), die Storagesysteme (TotalStorage) und die Drucksysteme.

Software Group Softwarelösungen. Aufgeteilt in *Information Management* (DB2), *Lotus* (Office und Groupware), *Rational* (Entwicklung), *Tivoli* (Service Management) und *Websphere* (Anwendungsintegration und -infrastruktur).

Global Finance Weltweit größter IT-Finanzdienstleister. Dazu gehört Leasing von Soft- und Hardware.

2.4 IBM International

IBM ist weltweit in über 170 Ländern vertreten. Der Stammsitz des Unternehmens ist Armonk, New York in den USA.

Insgesamt werden über 380 Tausend Mitarbeiter beschäftigt. Damit ist das Unternehmen das größte IT-Unternehmen weltweit.

Der Umsatz liegt knapp unter 100 Milliarden US Dollar, wovon der Gewinn etwa ein Zehntel beträgt.

2.5 IBM Deutschland

In Deutschland war die *IBM* ab 1910 unter dem Namen *DEHOMAG* vertreten. Seit 1949 verkehrt die Gesellschaft unter den bekannten drei Buchstaben.

Die *IBM Deutschland GmbH* beschäftigt etwa 22.000 Angestellte in rund 40 Standorten; darunter auch der Entwicklungsstandort Böblingen.

Die Geschäftsführung setzt sich seit Mai 2007 aus Martin Jetter (Vorsitzender), Christian Diedrich (Finanzen), Christoph Grandpierre (Personal), Matthias Hartmann (Global Business Services), Thomas Fell (Mittelstand) und Michael Diemer (Global Technology Services) zusammen. Den Vorsitz des Aufsichtsrat führt Hans Ulrich Maerki.

2.6 Labor Böblingen

Das Forschungs- und Entwicklungslabor in Böblingen ist eines von weltweit 30 IBM-Entwicklungszentren, und außerhalb Nordamerikas eines der größten.

Gegründet wurde die *IBM Deutschland Entwicklung GmbH*, die das Labor betreibt, 1953. Seitdem ist der Standort Böblingen ein wichtiges Standbein des Entwicklungsnetzwerkes der *IBM*. Das Labor befasst sich sowohl mit Software- als auch mit Hardware-Entwicklung.

Die Hardwareentwicklung befasst sich unter anderem mit dem *System z*, das der Nachfolger des *S/390* und das Flaggschiff der IBM-Hardware ist. Diese besonders ausfallsicheren Server stellen die unternehmenskritische IT-Infrastruktur vieler großer Unternehmen dar. Das Labor war auch maßgeblich an der Entwicklung des *Cell-Prozessors* beteiligt, der in Kooperation mit Sony und Toshiba realisiert wurde. Mit der weltweiten Verantwortung für die Architektur, das Design und die Implementierung von Linux auf *IBM zSeries* ist das Böblinger Entwicklungszentrum das größte Linux Entwicklungszentrum weltweit.

Bei der Software deckt es drei der fünf IBM-Softwarebereiche ab. Das sind WebSphere, Tivoli und Information Management. Weitere Kernkompetenzen der Software-Entwicklung sind Spracherkennungstechnologien sowie Produkte und Lösungen für die Bioinformatik-, Automobil- und Finanzbranche.

Im Juli 2008 wurde die *IBM Deutschland Entwicklung GmbH* im Rahmen der *One IBM*-Initiative in *IBM Deutschland Research & Development GmbH* umbenannt.

2.7 Die Abteilung

Innerhalb des Labors war ich der Abteilung *Open Systems Design and Development* (OSDD) zugehörig.

Die Mission der Abteilung ist das Design, die Entwicklung und die Integration von *Cell*- und *PowerPC*-basierten Blades[2] für die *IBM BladeCenter* Umgebung.

Abbildung 2.2: Abteilungs-Logo

Die Abteilung ist in drei Fokusteams mit den angeführten Aufgaben aufgeteilt.

- **Open Systems Hardware**
  Hardware-Architektur, Board-Design und Board-Entwicklung
- **Open Systems Integration**
  Hardware-Bringup, System-Integration und Engineering-Test
- **Open Systems Firmware Development**
  Firmware-Architektur, -Design und -Entwicklung

Ich war dem Bereich *Open Systems Hardware* angehörig, wobei unser Projekt an sich sehr unabhängig vom sonstigen "daily business" war. Wir hatten eher eine Sonderstellung.

[2] Besondere flache Bauform für Server

3 Das Projekt

Das Projekt an dem ich gearbeitet habe, trägt die Bezeichnung "Real Time Showcase mit dem Cell/B.E. Mikroprozessor: Implementierung einer optisch gesteuerten Produktionseinheit".

Es war eines der *IBM Speedteams* in diesem Jahr. Die offizielle Beschreibung dafür lautete folgendermaßen:

> Der revolutionäre Superchip Cell wurde in Kooperation von Sony, Toshiba und IBM für Sonys Playstation 3 entwickelt. Er wird derzeit in der Playstation 3 und dem IBM Blade Server QS21 eingesetzt. Aufgrund seiner herausragenden Architektur eignet sich der Cell Chip in besonderer Art und Weise für Echtzeit-Datenverarbeitung. Ziel dieses Projekts ist es einen Showcase zu implementieren, der die Leistungsfähigkeit des Cell/B.E.s für Echtzeit-Aufgaben unter Beweis stellt. Hierbei werden vier Roboterarme von zwei Kameras gesteuert. Das Speedteam wird die Ablaufsteuerung der Roboterarme implementieren, damit diese gemeinsam eine Aufgabe erledigen können. Die Kameras werden dabei die Roboterarme kontrollieren. Die Ansteuerung der Roboterarme ist bereits vorhanden, so dass sich dieses Projekt voll auf die Cell/B.E. Programmierung, die optische Datenverarbeitung mit OpenCV und die Algorithmen zur Ablaufsteuerung konzentriert. Als Student haben Sie die Möglichkeit - im wahrsten Sinn des Wortes - Ihre Programmierung anzufassen. Sie Programmieren auf dem interessantesten Prozessor der derzeit auf dem Markt ist und zudem gewinnen Sie Einblick in die Echtzeit-Datenverarbeitung.

Begonnen wurde das Projekt im April 2007. Seitdem arbeiten Studenten daran.

Der erste Student hatte die Grundlagen gelegt indem er die low-level Kommunikation mit dem Roboter, die Inverse Kinematik[1] und ein erstes Framework für die Programmierung des Cell-Prozessors erstellt hat. Zum Einsatz kamen Lynx6-Roboterarme, die im Abschnitt 3.2.2 noch genauer beschrieben werden.

Sein Nachfolger arbeitete während seiner Masterarbeit vor allem an den mathematischen Berechnungen der Inversen Kinematik in allgemeiner Form. Seine Arbeit bezieht sich auf einen Industrie Roboter mit sechs Freiheitsgraden, der leider nicht beschafft werden konnte. Somit simulierte er seine Berechnungen nur.

[1] Berechnung der Gelenkwinkel in Roboterarmen zum Erreichen eines bestimmten Punktes

Student Nummer drei, und damit mein Vorgänger und Kollege, verweilte ein ganzes Jahr im Unternehmen, programmierte während seiner Zeit ein umfassendes Framework um die ganzen low-level Angelegenheiten zu abstrahieren. Dies erleichterte die Programmierung ungemein. Zudem band er *OpenCV* zur visuellen Erkennung mit ein.

Ich war nun der vierte Student der am Projekt arbeitete. Einen Monat nach mit kam noch eine weitere Studentin hinzu.

Als unser spanischer Teampartner, mein Vorgänger, im Juni seine Zeit im Projekt beendete, kam ein neuer Student hinzu. Dieser war von MIT in Cambridge. Es lässt sich also sagen, dass unser Projekt stark international geprägt var. Unsere Kommunikation war folglich größtenteils in Englisch.

3.1 Zielsetzung

Ziel des Projekts ist es, ein Robotsteuerungs Framework zu erstellen, das auf der Cell/B.E. basiert.

Üblicherweise werden Roboter über Digitale Signalprozessoren (DSPs) gesteuert. Diese erfordern spezielle Fachkenntnisse in der Programmierung und jeder Roboter wird über einen eigenen DSP gesteuert. Mit Hilfe der Cell-Architektur soll eine Alternative geschaffen werden, die es ermöglicht auf einfache Weise Robotersteuerungen zu entwickeln. Als Werkzeuge sollen die Programmiersprache *C* und das Betriebssystem *GNU/Linux* zum Einsatz kommen, da diese vom Cell unterstützt werden und zudem in der Industrie bekannt sind.

Die Echtzeitfähigkeit die bisher für DSPs sprach kann mit dem Cell-Prozessor ebenso erreicht werden. Seine hohe Skalierbarkeit und die geringen Latenzen ermöglichen es dann sogar eine Vielzahl von Robotern, eventuell sogar ganze Fertigungsstraßen, mit nur einem Cell-Chip zu steuern.

Dass dies realistische Vorstellungen sind, soll nun gezeigt werden, indem ein Showcase mit vier Robotern und einer visuellen Komponente an einem System erstellt wird.

3.2 Verwendete Technologie

Unser Entwicklungssystem war ein angepasster IBM *QS21* Cell/B.E. Blade-Server, welcher zwei Cell-Prozessoren mit je einem Gigabyte Arbeitsspeicher beherbergt. An diesem waren vier Lynx6-Roboterarme von Lynxmotion[2] angeschlossen, und außer dem noch eine *mvBlueFOX* Kamera von *Matrix Vision*[3]. Als Betriebsystem lief die PowerPC-Version von *Fedora 7* GNU/Linux mit einem angepassten Kernel.

[2] Website: http://lynxmotion.com
[3] Website: http://matrix-vision.com

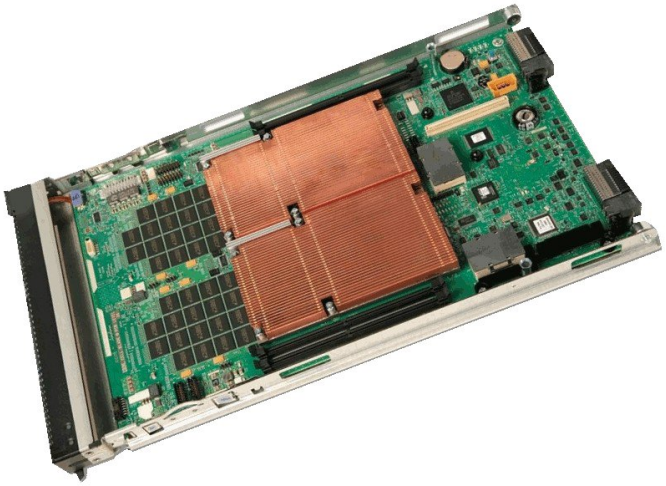

Abbildung 3.1: QS21 Blade-Server mit zwei Cell-Prozessoren

3.2.1 Cell-Prozessor

Ausgeschrieben *Cell Broadband Engine Architecture* genannt, werde ich hier meist die Kurzformen *Cell/B.E.* oder einfach nur *Cell-Prozessor* verwenden.

Dieser Chip wurde, zwischen 2001 und 2005, in einer Kooperation von Sony, Toshiba und IBM entwickelt. Der Hauptteil der Entwicklungsarbeit wurde dabei von *IBM* übernommen.

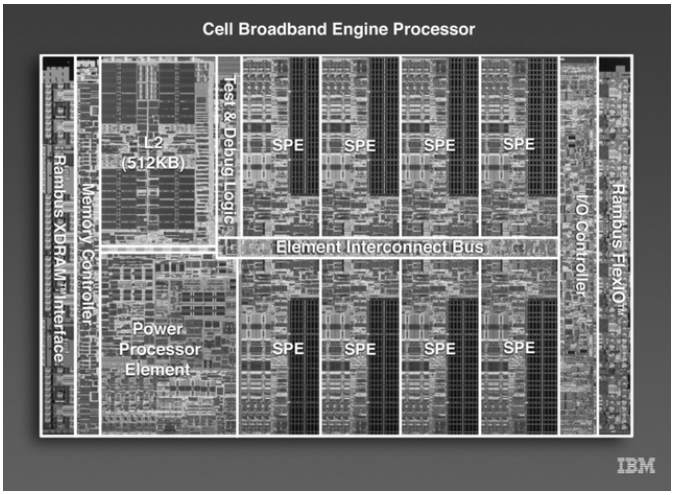

Abbildung 3.2: Cell/B.E. Chip

Bei der Cell/B.E. handelt es sich um eine heterogene Multicore-Architektur. Das bedeutet, dass der Prozessor aus mehreren Kernen besteht, die (im Gegensatz zu den x86-Multicores aber) aus verschiedenen Kerntypen bestehen. Der Cell verfügt über einen

PowerPC-Kern (PPE/PPU) und acht sogenannten Synergistic Prozessor Elemente (SPE/SPU). Die PPE ist ein vollwertiger 64-bit PowerPC Kern. Er kann in herkömmlicher Weise verwendet werden, so kann darauf zum Beispiel ein Betriebsystem oder eine beliebige Anwendung laufen. Die SPEs dagegen sind für große Rechenleistung optimiert, Datentransfer-Operationen sind bei ihnen eher langsam.

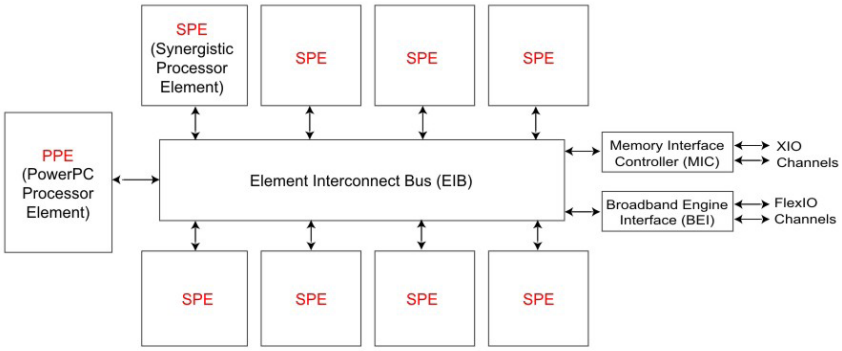

Abbildung 3.3: Schematischer Aufbau der Cell/B.E.

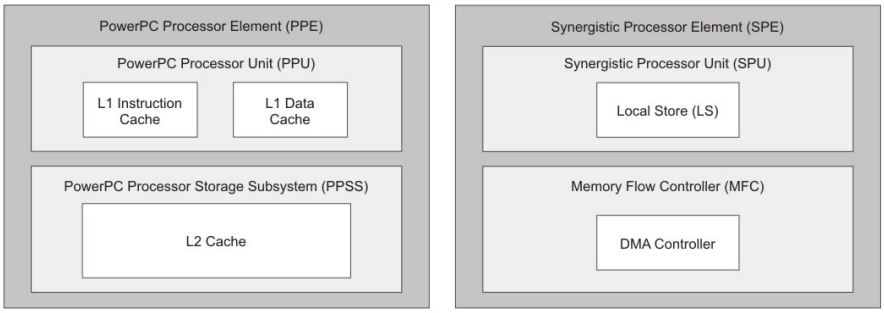

Abbildung 3.4: Schematischer Aufbau von PPE und SPE

Üblicherweise übernimmt die PPE die Kontrolle und verteilt die Arbeit auf die einzelnen SPEs die dann unabhängig von einander arbeiten. Die Ergebnisse fließen anschließend an die PPE zurück.

Zur Kommunikation zwischen den einzelnen Kernen stehen drei verschiedene Wege zur Verfügung, welche alle über den Element Interconnect Bus (EIB) abgewickelt werden.

- **Mailboxen** Sie sind Hardwareimplementierungen von Nachrichtenwarteschlangen. Jede SPE hat drei davon. Nachrichten haben eine feste Größe von 32 Bit.
- **Signale** Wie auch Nachrichten umfasst ein Signal ebenfalls 32 Bit, jedoch existieren keine Warteschlangen, so dass pro Signalkanal genau ein definierter Zustand aktiv sein kann. Jede SPE hat zwei Signalkanäle.

- **DMA-Transfers** Hierbei wird den SPEs ermöglicht auf den allgemeinen Hauptspeicher zuzugreifen, ebenso kann die PPE so auf den lokalen Speicher einer SPE zugreifen. DMA[4]-Transfers können mit bis zu 16 Kilobyte an Daten übertragen werden. Die Transfers werden vom Memory Flow Controller (MFC) der SPE durchgeführt und können somit parallel zu den aktuellen Instruktionen ablaufen.

Für DMA-Transfers empfiehlt es sich, die Technik *Double-Buffering* einzusetzen, bei der die nächsten Daten schon geholt werden, während mit den aktuellen noch gerechnet wird. Auf diese Weise kann die SPE nahezu voll ausgelastet werden.

Mailboxen und Signale werden üblicherweise vor allem für die Übertragung von Statusinformatinen verwendet.

Der Cell ist ein sehr leistungsstarker Prozessor mit einer Maximalleistung von 230 GigaFLOPS[5].

Er wird momentan vor allem in Sonys *Playstation 3* und IBM Blade-Servern verbaut. Aber auch in Supercomputern, wie dem *IBM Roadrunner*, der am 9. Juni diesen Jahres als erster Rechner mehr als ein PetaFLOPS erreicht hat, wird er eingesetzt. Zudem plant die IBM ihn künftig in ihre Mainframes zu integrieren.

3.2.2 Roboterarm

Die von uns verwendeten Roboterarme sind das Modell *Lynx6* vom Hersteller Lynxmotion. Die Roboter sind aus Lexan gefertigt und werden als Bausatz geliefert. Sie sind etwa 20 Zentimeter hoch und 40 lang.

Abbildung 3.5: Lynxmotion Lynx6 Roboterarm

Sie haben fünf Freiheitsgrade (Basisdrehung, Schulter, Ellenbogen, Handgelenk, Handdrehung) und damit einen weniger als gängige Industrieroboter oder der menschliche

[4]Direct Memory Access
[5]FLoating point Operations Per Second — übliches Maß zum Vergleich von Prozessorleistung

Arm. Die Zahl “6” in der Modellbezeichnung rührt von einem sechsten Gelenk her, das jedoch nur der Greifer ist und damit keinen weiteren Freiheitsgrad darstellt.

Die Bewegung der Gelenke wird von Servomotoren[6] (kurz “Servos”) übernommen.

Angeschlossen sind die Roboterarme über USB am Cell-Blade und per serieller Schnittstelle am Roboter, dazwischen sitzt ein USB-zu-Seriell-Konverter.

3.2.3 Kamera

Die optische Komponente wurde erst kurz vor meiner Zeit in das Projekt eingeführt. Anfangs wurde noch eine handelsübliche Webcam verwendet. Später wurde diese durch eine professionelle CCD-Kamera der Firma *Matrix Vision* ersetzt. Bei dieser handelt es sich um eine *mvBlueFOX*, die 100 Mal pro Sekunde ein Schwarz-Weiß-Bild mit ein einer Auflösung von 640x480 Pixeln liefert.

Abbildung 3.6: Matrix Vision mvBlueFOX

Zur Bilderkennung verwendeten wir die Open Source Bibliothek *OpenCV*, welche auf den Cell portiert und dafür optimiert ist.

3.3 Ausgangssituation

Als ich meine Arbeit antrat waren folgende Dinge bereits vorhanden.

- Framework das die low-level Kommunikation zwischen den SPEs regelt
- Scheduler der die einzelnen Programmmodule verwaltet und die Echtzeiteinhaltung kontrolliert
- Inverse Kinematik für den Roboterarm
- Programme mit statischen Bewegungsanweisungen für den Roboter
- Einfache dynamische Bewegungen an Hand von Gesichtserkennung mit OpenCV

[6] Motoren die bestimmte Positionen anfahren und halten können. Häufig im Modellbau eingesetzt.

Die alten Arbeiten meiner Vorgänger waren abgeschlossen und mein Teampartner, der schon ein halbes Jahr bei IBM war und während dieser Zeit das Framework ausgearbeitet hatte, feilte daran nur noch herum. Somit fand mit mir eine Art Neubeginn statt.

3.4 Der Showcase

Die neue Aufgabe war, einen Showcase zu erstellen, der auf der *Automatica*, einer Messe für Automatisierungstechnik in München, vorgeführt werden sollte.

Der Showcase sollte vier Roboterarme beinhalten, die sich einen Ball zuspielen. Der Ball sollte einen definierten Bereich nicht verlassen — dafür sollten die Roboter sorgen. Die Position des Balles sollte durch Patternerkennung im Kamerabild herausgefunden werden.

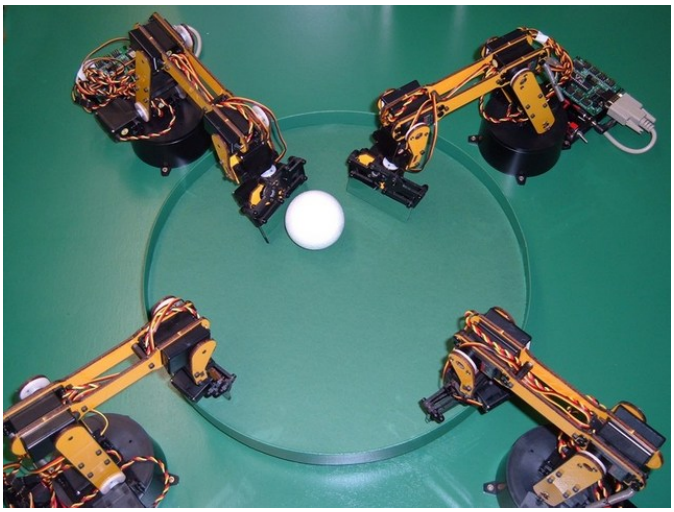

Abbildung 3.7: Am Ende sah es so aus

4 Meine Tätigkeit

In den ersten Tagen musste ich zuerst meinen Rechner einrichten. Das Betriebssystem (Red Hat Enterprise Linux 5) war zwar schon installiert, jedoch mussten diverse Programme eingerichtet werden. Des weiteren wird für die Entwicklung für den Cell ein spezielles Software-Development-Kit (SDK) benötigt. Diese besteht aus eine Vielzahl von Teilen deren Setup einige Zeit in Anspruch nahm.

Als ich meinen Rechner soweit einsatzbereit hatte, nahm ich mir die umfangreiche Dokumentation zur Entwicklung von Cell-Programmen vor. Mit Hilfe von ihr sollte und wollte ich einen Überblick über die vorliegende Architektur und die Programmierung für sie gewinnen. Es gibt einen Simulator, der einen Cell-Prozessor imitiert und auf einem gewöhnlichen x86-System läuft. Mit diesem habe ich erste Testprogramme geschrieben um ein Gefühl für Cell-Programmierung zu bekommen.

Nachdem ich dann Zugriff auf den bestehenden Code hatte, studierte ich die Arbeiten meinem Vorgänger, bevor ich mich daran machte, das aktuelle Framework zu erforschen.

4.1 Kollisionskontrolle

Meine erste Aufgabe bestand darin, eine Kollisionskontrolle für die Roboterarme einzubauen. Diese in sich abgeschlossene Aufgabe war für den Anfang gut geeignet. So konnte ich ohne allzuviel Vorwissen haben zu müssen, sanft in das Projekt einsteigen.

Der Algorithmus sollte möglichst allgemein sein und nicht nur mit genau unserer Roboteranordnung funktionieren.

Das Problem Kollisionserkennung ist einfach Abstandsberechnung von Objekten. Unsere Objekte sind Roboterarme, die sich als vier aneinander hängende Linien ansehen lassen — jedenfalls aus Sicht der Kollisionserkennung. Das eigentliche Problem besteht also aus Abstandsberechnungen von Strecken (nicht Geraden) im Raum. Um nicht die komplizierte Berechnung von Streckenabständen durchführen zu müssen, habe ich jede Strecke durch eine Anzahl Punkte auf ihr ersetzt. Somit musste ich nur Punktabstände berechnen, was einfach ist; allerdings in größerer Anzahl.

Ich berechnete die Abstände jedes Kollisionspunktes eines Roboters, mit jedem Kollisionspunkt jedes anderen Roboters.[1] Da dabei, vereinfacht, jeder Punkt mit jedem verglichen wird, resultiert daraus eine quadratische Laufzeit: $O(n^2)$. Für eine größere

[1] Kollisionen innerhalb eines Roboterarms sollten durch die Kinematik abgefangen werden.

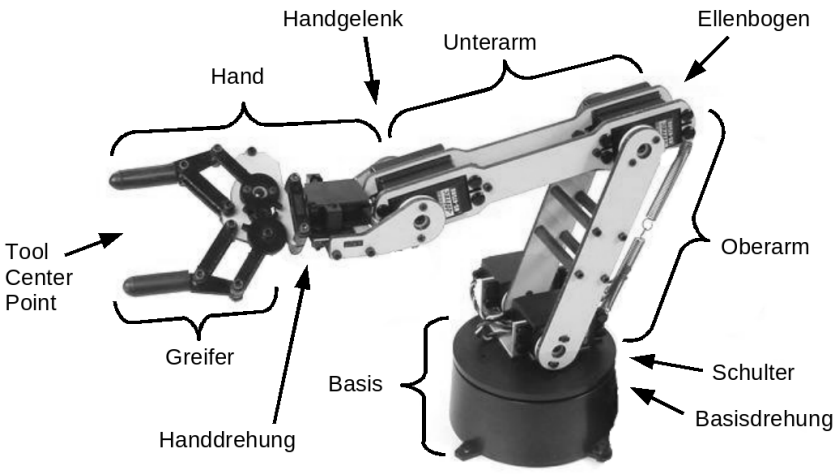

Abbildung 4.1: Terminologie des Roboterarms

Anzahl von Robotern, oder mehr Kollisionspunkten pro Strecke, sollten wir also recht schnell viel Zeit brauchen. Diese Vorraussage wurde von den Performance-Messungen meines Teamspartners bestätigt.

Mit unseren vier Robotern konnte ich 16 Kollisionspunkte pro Knochen einfügen, ohne besonders viel Zeit zu verbrauchen; 32 Punkte waren noch machbar. Ich entschied mich für vier Kollisionspunkt pro Knochen, denn dies führte zu einer voll ausreichenden Genauigkeit, wie Abbildung 4.2 zeigt.

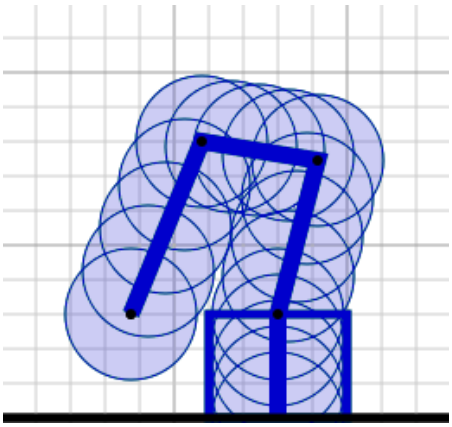

Abbildung 4.2: Hervorgehobene Kollisionszone bei vier Kollisionspunkt pro Knochen

Programmablauf Als Ausgangsdaten habe ich die Position und Ausrichtung der Roboter in der "Welt" und sämtliche Gelenkwinkel. Aus diesen Daten habe ich die Welt-Koordinaten, also Koordinaten bezogen auf das globale Koordinatensystem, aller Gelen-

ke berechnet. Mit den globalen Koordinaten führe ich die Kollisionsberechnung durch, denn diese liegen im gleichen Koordinatensystem und Abstandberechnungen sind somit einfach: $distance = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$.

Ergebnis Die Kollisionskontrolle stoppt das Program bevor eine tatsächliche Kollision auftritt. Die Performance der Moduls ist tragbar, könnte aber weiter verbessert werden. Die Optimierungen können in zwei Richtungen erfolgen. Zum Ersten könnten die Berechnungen für den Cell optimiert werden und somit eine deutlich höhere Rechenleistung erreicht werden. Zum Zweiten könnte die Zahl der Abstandberechnungen reduziert werden indem man zuerst schaut, ob ein Arm überhaupt in der Nähe eines anderen ist, bevor einzelne Kollisionspunkte angeschaut werden.

4.2 Visualisierung

Im Laufe meiner Arbeiten an der Kollisionskontrolle musste ich mir oft ein Bild von den Stellungen der Roboter machen um die berechneten Werte überprüfen zu können. Ich wollte möglichst unabhängig von den realen Robotern im Labor arbeiten, von denen wir zu dieser Zeit sowieso nur zwei hatten. Anfangs habe ich mir die Winkelstellungen oder die Koordinaten der Gelenke ausgeben lassen und dann unseren defekten Testarm entsprechend eingestellt. So konnte ich mir die Situation einigermaßen vorstellen. Alternativ habe ich zu Stift und Papier gegriffen um mir die Situation zu zeichnen. Dies bedeutete jedoch immer wieder den gleichen Aufwand zu betreiben, nur um die Roboterstellungen visuell vor Augen zu haben.

Bilderzeugung Wiederkehrende Aufgaben soll man automatisieren — das ist bekannt. Ich wollte mir also automatisch Bilder generieren lassen, um die stupide Arbeit auf den Computer zu übertragen. Als geeignetes Grafikformat erschien mir das *Scalable Vector Graphics*-Format, das eine Textdatei in XML ist. Somit konnte ich einfach line- und circle-Befehle vom Programm in eine Textdatei schreiben lassen. Im Vergleich zu den bisherigen Ausgaben der Gelenk-Koordinaten, war es in erster Linie nur eine andere Schreibweise. SVG-Dateien werden von gängigen Bildbetrachtern und aktuellen Browsern angezeigt.

Ich habe die SVG-Generierung als separates Modul implementiert, das bei Bedarf aktiviert werden kann und dann in jedem Programm-Cycle ein Bild der Roboterstellungen zeichnet. Die Darstellung im Bild ist zwei-dimensional, da 3D-Abbilder nur bei beweglicher Kamera sinnvoll nutzbar sind. Ich habe deshalb eine Dreitafelprojektion verwendet, die auch technischen Zeichnungen und Bauplänen bekannt ist.

Animation Wenig später waren dann selbst die Einzelbilder teilweise zu umständlich, so dass der Wunsch nach einer animierten Darstellung des Geschehens aufkam. SVG-Animationen ausgeben zu lassen wäre deutlich komplizierter geworden, und diese

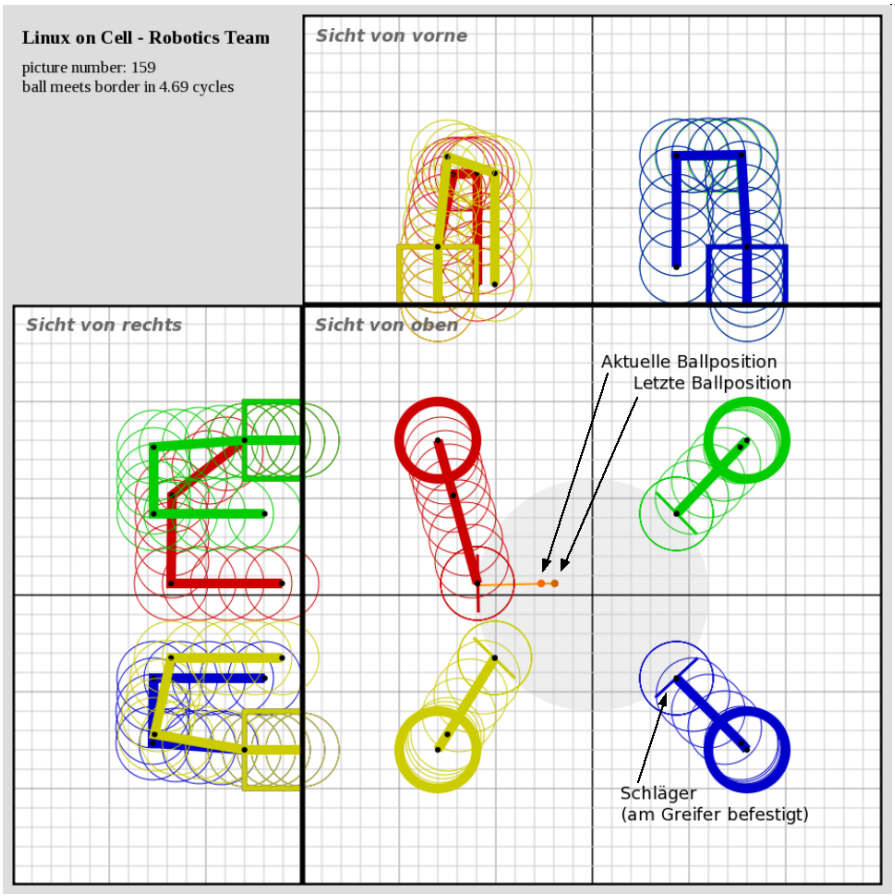

Abbildung 4.3: Eine generierte SVG-Grafik mit Beschriftungen

können auch nur von wenigen Programmen dargestellt werden. Deshalb habe ich mit der Programmsammlung *ImageMagick* aus dem SVG-Bildern ein animiertes GIF gemacht. Dieses stellte dann auch die Zeitdimension in den Bewegungen dar. Später wurden die GIFs dann durch komprimierte AVI-Filme (mit *MEncoder* erstellt) ersetzt, da diese deutlich weniger Speicher verbrauchen und schneller erzeugt werden konnten.

Ergebnis Es liegt hier eine Komponente vor, die die Entwicklung des Restsystems deutlich vereinfacht hat. Mit ihr war es möglich, im Büro zu testen — es mussten nicht die Roboter im Labor bemüht werden. Zudem konnten durch den direkteren Weg, von den Basisdaten zum visuellen Ergebnis, Fehlerquellen eliminiert werden. So war es unter anderem auch hilfreich bei der Suche nach einem Bug in der Kinematik — der ja erst durch die Unterschiede zwischen SVG-Simulation und realen Roboterbewegungen sichtbar wurde.

Es freut mich um so mehr, dass ich bei Fred Brooks “The Mythical Man-Month” bestätigt finde, was ich nebenbei herausgefunden habe: Es ist sinnvoll einen Simulator

zu haben und diesen parallel mit dem eigentlichen Programm zu entwickeln.

4.3 Intelligenz-Modul

Nachdem ich meine Arbeiten an den ersten zwei Modulen soweit fertig war, haben wir im Team beschlossen, dass ich mit dem Intelligenz-Modul weiter machen sollte. Der bisherige Code war natürlich nicht ganz fertig und es war auch durchaus so gewollt, dass die einzelnen Programmteile kontinuierlich weiterentwickelt werden. Neue Funktionalitäten im einen Modul zogen neue Anforderungen in einem weiteren nach sich. So wurde der gesammte Code stückweise ausgebaut.

Nichts desto trotz begann ich dann hauptsächlich an der Intelligenz, oder Logik, zu entwickeln.

Aufgabe der Intelligenz Das Logik-Modul plant, entscheidet und gibt die resultierenden Befehle. Es beherbergt den "interessanten" Programmcode, denn hier steckt eine Künstliche Intelligenz (so simpel sie auch sei).

In unserem Fall hat die Logik die aktuelle und letzte Ballposition, sowie die Roboterpositionen zur Verfügung. Das Modul soll ausgeben was welcher Roboterarm als nächstes tun soll.

Grundgedanken Unser Showcase ist so aufgebaut, dass es immer mindestens einem Roboter möglich ist, den Ball zu spielen. In den meisten Fällen wird der Ball nur von genau einem Arm erreichbar sein, dann wird dieser ihn spielen. Wenn zwei Roboter nah genug sind, übernimmt der Arm mit der geringeren Entfernung (genannt "Master") die Kontrolle und spielt den Ball. Der andere Arm (genannt "Slave") fungiert als Unterstützung und bewegt sich relativ zum Master. Dieses Verhalten sollte uns bei unvorgesehenen Änderungen des Ballwegs nützlich sein. Die restlichen Arme fahren in eine Warte-Position. Nur wenn der Ball in der Mitte des Spielfeldes zum stehen kommt, haben mehrere Roboter die Möglichkeit ihn zu erreichen. Es wird dann der nahste Roboter aktiv werden.

Die Schussbewegung Der Ball sollte nicht nur am Paddle (= Schläger), den wir am Greifer des Arms befestigt haben, abprallen, sondern es sollte ihm wieder neue Bewegungsenergie mitgegeben werden. Es war also klar, dass die Roboter eine gewissen Schussbewegung ausführen mussten. Desweiteren war es notwendig das Paddle im richtigen Winkel auszurichten, um den Ball in eine bestimmte Richtung spielen zu können. Das Schussziel sollte in im Intelligenz-Modul gesetzt werden können. Damit wäre es dann auch möglich, dass sich nur zwei Roboter hin und her spielen. Solche "Spielereien" sollten auf jeden Fall machbar sein, denn diese machen einen Showcase erst interessant und zeigen einen Hauch menschlichen Verhaltens, was bei Robotern eine wichtige Komponente ist.

Mögliche Erweiterungen Die Intelligenz bietet Ausbaumöglichkeiten aller Art. Das geht von einer Bibliothek von Entscheidungsmöglichkeiten, über persönliche Verhaltensmuster einzelner Roboter, bis zu Gruppenverhalten, Taktik, oder gar Finten.

So zumindest die Theorie, denn ...

Die Realität Bevor wir den Aufbau für die Roboter hatten, simulierten wir nur eine mögliche Ballbahn und betrachteten das Ergebnis in den erzeugten SVG-Grafiken. Es funktionierte alles ziemlich gut. Als wir das Ballspiel dann in echt sahen war es katastrophal! Wir hatten nicht bedacht, dass sich der reale Ball keineswegs auf geraden Bahnen bewegte. Eine Billard-Kugel hätte das auf einem Marmor-Untergrund vielleicht getan. Unser Schaumstoffball, der noch nicht mal eine ebene Oberfläche hatte, bewegte sich manchmal fast willkürlich. Er änderte seine Bahn, rollte wieder zurück, blieb einfach liegen, wackelte auf der Stelle ... und unsere Intelligenz, die mit einem (idealen) simulierten Ball umgehen konnte, wusste plötzlich nicht mehr wie ihr geschah.

Es galt also diesen Unregelmäßigkeiten entsprechend zu begegnen. Sich ändernde Ballwege waren grundsätzlich kein Problem, da sie schon von Beginn an bedacht waren. Liegen gebliebene Bälle mussten neu angestoßen werden — auch dies war mit relativ wenig Aufwand machbar. Auf der Stelle wackelnde Bälle waren schon ein größeres Problem, denn einmal sieht es so aus, als ob der Ball nach rechts läuft, in nächsten Zyklus aber als wenn er nach links läuft, und so weiter. Die Roboter wissen also nicht wohin er sich denn nun bewegt. Ich habe dazu nur Ballbewegungen die zweimal in ungefähr die gleiche Richtung gehen, als tatsächliche Ballbewegung gewertet. Eine Bewegung in die entgegengesetzte Richtung im nächsten Zeitschritt, bedeutet Stillstand. So waren die auf der Stelle zitternden Bälle eliminiert. Alle tatsächlichen Ballbewegungen habe ich dann noch gefiltert um die Ballbahn gleichmäßiger zu machen.

Zustandslosigkeit Das große Problem, das sich mit der Zeit herausstellte, war die Zustandslosigkeit des Intelligenz-Moduls. Ich hatte es so aufgebaut, dass jeweils nur anhand des letzten Zeitschrittes (wenn überhaupt) entschieden wird, wie als nächstes agiert werden soll. In der Realität stellte sich aber heraus, dass dies nicht genug war. Es war nötig, Roboter in Zustände zu versetzen und diese über mehrere Zyklen beibehalten zu können. So benötigt es mehrere Zyklen um einen Ball, der hinter einem Roboterarm festgeklemmt ist, wieder in's Spiel zu bringen. Der Roboter muss dazu seinen Arm heben, über den Ball zurück fahren und ihn von hinten anstoßen. Während dieses Vorgangs sollte er nicht unterbrochen werden. Änderungen an der Grundstruktur der Intelligenz waren an diesem Punkt, vor der ersten Messe, zeitlich nicht mehr möglich; so setzte ich auf Variablen, um Zustände zu speichern. Die Lösung war nicht unbedingt schön, aber zu diesem Zeitpunkt zweckmäßig.

Wo ist der Ball? Die punktuelle Sichtweise der Dinge machten uns auch im Bezug auf die Position des Balles zu schaffen. Das Problem war, dass wir, wenn kein Ball vom Vision-Modul gefunden wurde, nicht sagen konnten, ob der Ball einfach nicht erkannt

wurde, er sich gar nicht mehr im Bild befand, oder er unter einem Roboterarm für die Kamera unsichtbar war. Ein Mensch nimmt dieses Problem zuerst gar nicht wahr. Für uns war es aber zentral, denn je nach tatsächlicher Position mussten verschiedene Aktionen ausgeführt werden: Wurde der Ball nur nicht erkannt, dann sollten die Arme abwarten, in der Hoffnung, dass der Ball im nächsten Zyklus wieder erkannt würde. War der Ball vom Tisch, dann sollten die Roboter in die Wartestellung fahren und so verweilen. War der Ball aber unter einem Arm eingeklemmt, dann sollte dieser eine "Befreiungsbewegung" ausführen um den Ball wieder in's Spiel zu bringen. Würde er, im letzten Fall, dies nicht tun, dann wäre das Spiel in einer Sackgasse angelangt, denn beim Abwarten, oder Bewegen in die Wartestellung würde der Ball weiterhin unter dem Arm bleiben — unsichtbar.

Eine der wichtigsten Anforderungen an das Programm war jedoch, dass es ohne menschliches Einwirken quasi endlos laufen sollte. Aus Sackgassensituationen wie dieser mussten also auch Wege hinaus führen.

Dieses Problem löste sich ziemlich gut, indem wir eine wahrscheinliche Ballposition (ausgehend von der bisherigen Bewegung) annahmen, falls er nicht gefunden wurde. Wurde er jedoch über mehrere Zyklen nicht gefunden, sollte zuerst der Roboter der dem Ball wahrscheinlich am nächsten war, eine Befreiungsbewegung ausführen. War der Ball dann immer noch verschwunden, sollten alle Roboter die Bewegung ausführen. Brachte dies den Ball noch immer nicht zum Vorschein, dann war er vermutlich außerhalb des Spielfeldes und die Roboter sollten in ihre Ruhestellung fahren und warten ... bis der Ball wieder auftauchen würde.

Eine neue Struktur Nach der ersten Messe machte ich mich daran, die Struktur des Intelligenz-Moduls im Bezug auf dessen Zustandslosigkeit zu ändern. Ich führte ein Task-Konzept ein, bei dem einem Roboter eine bestimmte Aufgabe zugewiesen werden kann. Diese erledigt er komplett, bevor er für weitere Aufgaben frei ist. Aufgaben höherer Priorität brechen allerdings jederzeit unwichtigere Jobs ab und treten an deren Stelle.

Auf diese Weise hatte ich die vorher "zusammengeschusterte" Funktionialität mit einem soliden Konzept nachgebaut. Die Handhabung ist jetzt deutlich einfacher und das Programm ohne Probleme erweiterbar. Da der neue Code nur sinngemäß dem alten entspricht (und nicht dessen Unregelmäßigkeiten beinhaltet), muss er erst noch optimiert werden um ein gleich gutes oder besseres Verhalten beim Fußballspiel zu erzielen. Leider war dies bisher noch nicht möglich, es sollte aber kein Problem darstellen.

Ergebnis Es existiert nun ein ordentliches Intelligenz-Modul, in etwa 500 Zeilen Quelltext, das vier Robotern zum Fußballspiel verhilft. Seine Struktur wurde mit Blick auf zukünftige Erweiterungen entworfen. Es bietet Ansatzpunkte für spieltaktische Raffinessen, auch wenn diese unsere (derzeitige) Hardware leider nicht ermöglicht. Das Modul ist in seinem jetzigen Umfang für seine Aufgabe gut geeignet. Es sollte allerdings noch Zeit in das Feintuning gesteckt werden. Bei Bedarf kann weitere Spiellogik sehr einfach auf die vorliegende Basis aufgesetzt werden.

4.4 Die Vision

Für dieses Modul war zwar hauptsächlich meine Kollegin zuständig, jedoch möchte ich der Vollständigkeit halber ein paar Informationen dazu hier anführen.

In jedem Zyklus holt das Vision-Modul zunächst ein Bild von der Kamera. Dieses wird auf etwa 200x150 Pixel verkleinert um Rechenzeit zu sparen und invertiert. Dann wird mit einem *Haar-like*-Filter aus der *OpenCV*-Bibliothek nach einem Ball gesucht. Von den gefundenen Bällen werden diejenigen aussortiert, die außerhalb der Spielfläche sind. Sind dann noch immer mehr als ein erkannter Ball übrig, wird derjenige Ball ausgewählt, der ausgehend von der letzten Position am wahrscheinlichsten ist. Als Resultat liefert das Modul die Koordinaten des Balles, oder zeigt, dass kein Ball gefunden wurde.

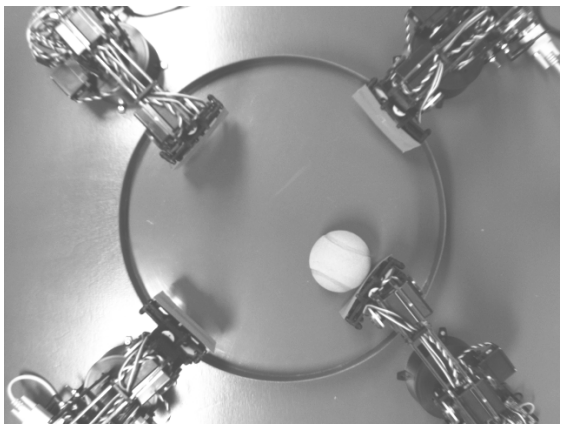

Abbildung 4.4: Von der Kamera aufgenommener Bereich

Heuristik Bilderkennung ist nicht deterministisch und so können bei mehreren Durchläufen mit gleichen Eingangsbild unterschiedliche Bälle gefunden werden. Dies führt dazu, dass bei uns manchmal kein Ball gefunden wird obwohl einer vorhanden ist, ebenso wie gefundene Bälle an Stellen, wo keine sind. Insbesondere die Roboterarme werden von Zeit zu Zeit als Ball erkannt.

Auch sonst lief bei der Bilderkennung nicht alles so, wie wir uns das dachten — es scheint fast, als gäbe es dafür ganz eigene Regeln. Unsere Trainingsbilder waren allesamt von einem ganz anderen Ball, als wir nacher verwendeten. Das beste Ergebnis lieferte dann ein früher Test mit nur etwa hundert Fotos; mit mehr Fotos wurde das Ergebnis nur schlechter. Die Größe des Balles war (abgesehen von der Helligkeit natürlich) der wichtigste Einflussfaktor auf den Erkennungserfolg. Das Oberflächenmaterial und die -bemalung wirkten sich kaum aus. Unser Ball hatte am Ende sogar große schwarze Punkte auf weißem Untergrund (in Anlehnung an den “Europass”, den Ball der EM 2008), ohne dass die Erkennungsrate merklich schlechter wurde.

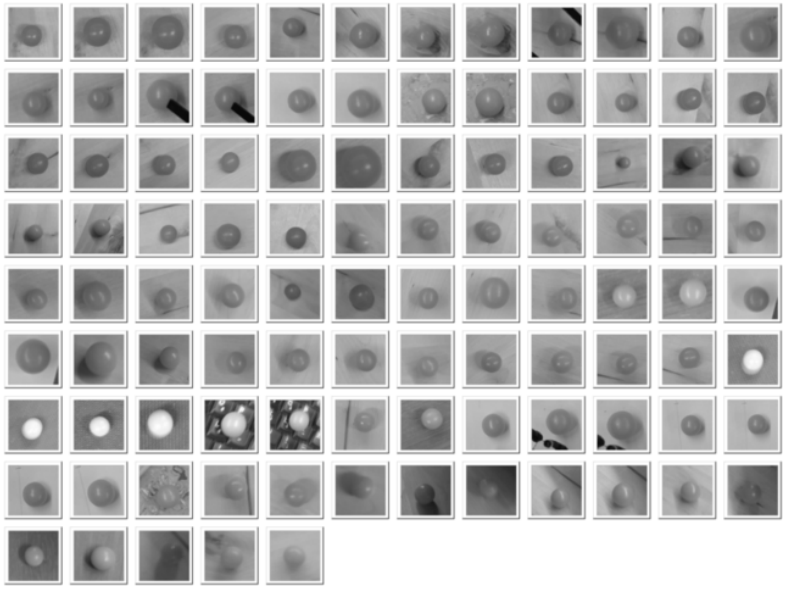

Abbildung 4.5: Unsere Trainingsbilder

Ergebnis Alles in allem können wir aber sehr zufrieden mit unserer Ballerkennung sein. Wir haben eine konstante Erfolgsrate von über 90% und schlechtes Licht oder ein teilweise verdeckter Ball wirken sich wenig aus. Gleichzeitig ist auch die Rate der Fehlerkennungen recht gering. Durch das invertierte Bild und hellem Ball auf dunklem Grund haben wir sowieso fast alle Fehlermöglichkeiten ausgeschlossen. Mit einem matten Untergrund war dann wirklich die letzte Irritationsgefahr gebannt und unser Vision-Modul arbeitet jetzt äußerst zuverlässig.

5 An der Öffentlichkeit

5.1 Automatica

Das erste Ziel des Projekts war die Messe *Automatica* die vom 10. bis 13. Juni in München statt fand. Dort stand unser Showcase am Stand von *Matrix Vision*. Unser Betreuer war vor Ort und stand den Besuchern für Fragen zur Verfügung. Wir durften auch einen Tag zur Messe fahren um unsern Showcase dort zu erleben. Nebenbei hatten wir Gelegenheit uns über den aktuellen Stand der Technik zu informieren und Anregungen aller Art aufzusammeln.

Der Showcase war ein Erfolg! Er war ein Eyecatcher — nicht nur am Stand von Matrix Vision. Er gehörte er zu den Dingen, bei denen die Besucher stehen blieben um sie sich genauer anzuschauen.

Das Besondere bei uns war, dass unsere Roboter keine definierten Bahnen abfuhren, wie es bei den meisten der anderen Aussteller (verständlicherweise) der Fall war. Unsere Roboter waren frisch, fröhlich, unberechenbar und hatten "ihren eigenen Kopf". Dass der Ball beim Spiel ab und zu vom Tisch fiel, war nicht schlimm, brachte es doch die Beobachter dazu, ihn wieder einzuwerfen und so mit den Robotern zu interagieren. Manch einer fragte sich, was wohl passieren möge, wenn die Kamera kein Bild mehr liefern würde ... und hielt sie dann einfach zu. Der Ball wurde festgehalten, mit der Hand abgeschirmt, es wurde ein weiterer Ball in's Spiel gebracht — alles auf ganz natürliche Weise, dem Spieltrieb folgend.

Unsere Roboter überstanden die Dauerbelastung an sich recht gut. Nur drei Servo-Getriebe waren nicht robust genug und gingen kaputt. Da wir dies eingeplant hatten, hatten wir Ersatzteile dabei.

5.2 Unternehmensinterne Vorführungen

Unsere Roboter riefen auch bei den Mitarbeitern in unserer Umgebung großes Interesse hervor. Selbstverständlich folgten wir dem Wunsch, bei ihrem Abteilungsmeeting eine kurze Einführung in unser Projekt mit anschließender Demonstration des Fußballspiels zu geben.

Ein zweiter Auftritt war beim *Take your children to work day*, den *IBM* regelmäßig veranstaltet. Wir organisierten eine Programm für eine zwanzig-köpfige Gruppe von Jugendlichen, die sehr interessiert und aktiv teilnahmen.

Auch beim Besuch einer Schulklasse wurden wir für einen Programmpunkt eingesetzt.

Wir waren bei diesen Aktionen immer gerne dabei, konnten wir doch mit unserem Projekt Andere begeistern. Das Präsentieren, vor immer wieder anderen Gruppen, war für uns auch eine gute Übung, da man solche Gelegenheiten nicht allzuoft hat. Zudem sind Events etwas Besonderes und schon alleine deshalb faszinierend.

5.3 Games Convention Developers Conference

Für das Ende meines Praxissemsters stand dann das zweite Highlight an: die *Games Convention Developers Converence* (kurz GCDC). Sie findet im Rahmen der *Games Convention*, der weltgrößten Spielemesse, in Leipzig statt. Bevor die etwa 200.000 Privatpersonen die Hauptmesse stürmen, treffen sich die Spiele-Entwickler und -Firmen in kleinerer Runde zur *Developers Conference* um sich auszutauschen und um gemeinsam in die Zukunft zu blicken.

Wir als *IBM* waren dabei, um die Branche über unsere Dienste und Hardware (insbesondere den Cell-Prozessor) zu informieren.

Unser Showcase war in erster Line der Eyecatcher des Standes ... und wohl auch **die interessanteste Demo aller Stände**. Die Roboter sollten aber nicht nur Aufmerksamkeit erregen, auch die Technik dahinter passte zum Themenfeld: der Cell-Prozessor. Er ist mit seiner hohen Rechenperformance optimal für die derzeitigen und kommenden Anforderungen der Spieleindustrie geeignet. Sei das in Spielekonsolen wie der *Playstation 3*, oder in Server-Backends für Online-Spiele.

Unsere Fußball-Roboter waren also ein schöner Einstieg für Gespräche: Vom sichtbaren Aufbau, zur Technologie dahinter, zu den Einsatzgebieten der Cell-Technologie für Spielefirmen.

Das Feedback von Seiten der Besucher, als auch Firmen-intern, war sehr positiv — eine Bestätigung unserer Leistung.

5.4 Medienwirkung

Der Showcase rief im Allgemeinen ein großes Interesse bei der Presse hervor. Auf den Messen kamen mehrere Fehrnsehteams (darunter das ZDF und der MDR) und eine Menge Fotografen vorbei um die Roboter abzulichten. Auch Interviews mit Reportern gehörten dazu.

Unser Projekt wurden in einem Bericht der *Sindelfinger Zeitung* beschrieben. ("Suche nach kreativen Köpfen", 2008-07-25)

Zudem erschien auf *heise online* eine News-Meldung, die ein Foto der Roboter enthält. (Meldungsnummer: 114500, "IBM will KI- und Physikberechnungen auf Online-Server verlagern", 2008-08-20)

Weitere Erwähnungen dürften folgen.

Abbildung 5.1: Automatica: Unser Showcase am Stand von Matrix Vision

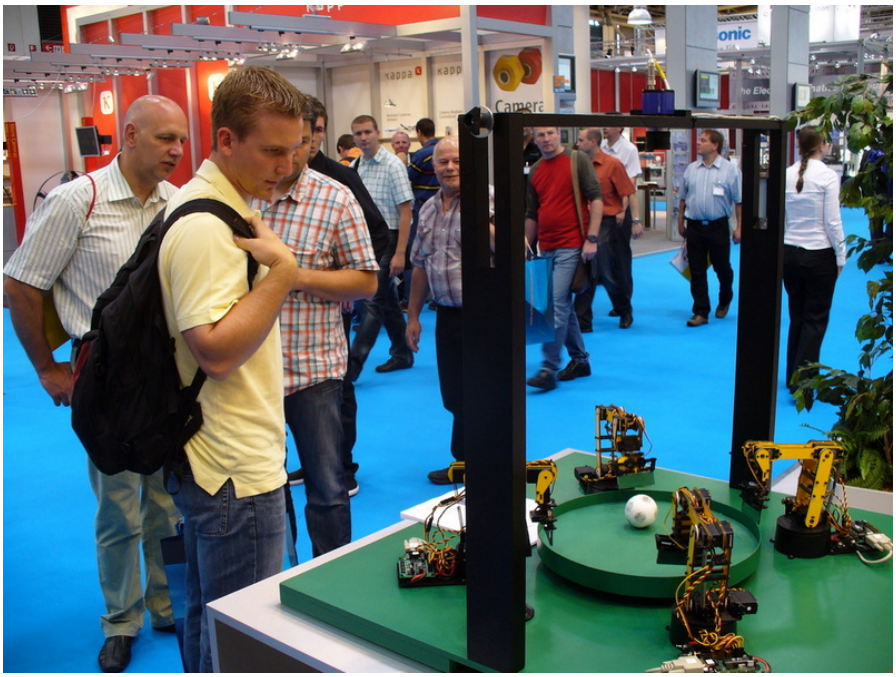

Abbildung 5.2: Automatica: Interessierte Besucher

Abbildung 5.3: GCDC: Gespräch mit einem Kunden

Abbildung 5.4: GCDC: Die Roboter machen Spaß!

6 Fazit

Ich habe mich bei der *IBM* vor allem deshalb beworben, weil ich mit C unter Unix ein interessantes Projekt anpacken wollte. Diesbezüglich bin ich voll auf meine Kosten gekommen. Nicht nur die reine Programmierung, sondern auch die Administration unseres Systems und der verwendeten Programme und Bibliotheken, erweiterten mein fachliches Wissen stark.

Der zweite entscheidende Grund war, dass ich in einem großen bedeutenden Computer-Unternehmen arbeiten wollte um diese Welt kennenzulernen. Auch hier habe ich wieder sehr viele Erfahrungen gesammelt. Was ich in meinem ersten Praxissemster in der kleinen IT-Abteilung des Industriebetriebs aufgenommen habe, wurde nun ergänzt mit Wissen aus dem Alltag der wohl größten Computer-Firma der Welt.

Wovon ich zum Bewerbungszeitpunkt noch nichts wusste, was ich jetzt aber sehr schätze, ist das Team mit dem ich arbeiten durfte und dessen Struktur. Wir waren drei Studenten und hatten viel Freiraum um uns selbst zu organisieren. Wir konnten unseren Weg zum Erreichen der vorgegebenen Ziele selbst festlegen. Und wir hatten die Möglichkeit mit kreativen Vorschlägen die Zukunft des Projekts mitzubestimmen. Wir waren stets ein entscheidender Teil "unseres" Projektes; jeder Einzelne steuerte sein Stück bei und trug damit auch seinen Anteil am Erfolg. Dieses Projekt bot einen direkten Zusammenhang zwischen Input und Output: Hängte ich mich rein, so sah ich das Ergebnis meiner Bemühungen schon wenige Tage später. Dies motivierte mich natürlich sehr.

Möglich wurde all das durch die Art wie mein Betreuer das Team führte — er ermöglichte was wir nutzten. Er, wie auch alle anderen Personen die ich bei der *IBM* kennenlernen durfte, waren stehts freundlich und hilfsbereit, egal weswegen ich sie aufsuchte. Es war wirklich eine tolle Erfahrung hier gearbeitet zu haben!

Dieses Praxissemster hat mich in jeder Hinsicht vorwärts gebracht.

Dafür danke ich dem Unternehmen, der Abteilung und besonders meinem Betreuer, Matthias Fritsch, herzlich!

Markus Schnalke
Breitingen, den 4. September 2008

Abbildungsverzeichnis

Abbildungen ohne Herkunftshinweis stammen aus dem Projekt.

Referenzen

Informationsquellen

Meine Informationen habe ich aus folgenden Quellen:

- IBM intern
- Software Development Kit for Multicore Acceleration (Version 3.0): “Programmer’s Guide” und “Programming Tutorial”
- http://ibm.com/de
- http://en.wikipedia.org/wiki/IBM
  http://de.wikipedia.org/wiki/IBM
- http://en.wikipedia.org/wiki/Cell_(microprocessor)
  http://de.wikipedia.org/wiki/Cell_(Prozessor)
- http://lynxmotion.com

Verwendete Software

Zum Erstellen dieses Berichts habe ich diese Software eingesetzt:

- Debian GNU/Linux & Red Hat Enterprise Linux
- pdflatex aus der teTeX-Distribution
- Vim
- ImageMagick, GIMP, gThumb und OpenOffice.org
- Mercurial zur Versionierung