Praxissemester bei isys vision GmbH & Co. KG

Bericht und Erfahrungen

Lorenz Bung lorenz.bung@htwg-konstanz.de
HTWG Konstanz
Angewandte Informatik, 4. Semester

25. Mai 2018

Zusammenfassung

In diesem Bericht geht es um mein Praktikum bei isys vision GmbH & Co. KG, welches ich im Rahmen meines Studiums im Fach Angewandte Informatik im 4. Studiensemester absolviert habe.

Ich werde beschreiben und erklären, welche Aufgaben mir weshalb gegeben wurden, wie ich sie gelöst habe und welche Werkzeuge und Vorgehensweisen ich dabei verwendet habe. Weiterhin werde ich die firmeninternen Abläufe, Ziele und Methoden nennen.

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

0. Einleitung

Zu Beginn möchte ich kurz die Hintergründe meines Praktikums erläutern, z.B. einige Informationen zu isys vision, die Wahl der Firma und den Bewerbungsprozess.

isys vision[1] ist ein Softwarehersteller, der sich mit grafischen Lösungen auseinandersetzt. Die Bildverarbeitung steht hierbei im Vordergrund und wird durch die von isys produzierte Software durchgeführt. Die im selben Gebäude gelegene Tochterfirma Ensenso[2] stellt Hardwarekomponenten her, welche dann in Kombination mit der Software von isys vision als Gesamtpaket an Endkunden, aber auch Großabnehmer verkauft wird. Dies umfasst hauptsächlich Komponenten wie Kameras und Beleuchtungsbauteile sowie Verbindungskabel usw.

Schon im Grundstudium war mir klar, dass ich für mein praktisches Studiensemester Firmen im Bereich der Bildverarbeitung, -generierung oder aber im Webumfeld suchen werde. Mit einem Praktikum in genannten Gebieten wollte ich einerseits meine Entscheidung für die Vertiefungsrichtung der Medieninformatik im Hauptstudium bekräftigen, andererseits war dies schon immer ein Bereich der Informatik, welcher mich besonders gereizt hat. Aus diesen Gründen habe ich mich unter Anderem bei isys vision beworben und schlussendlich ein halbes Jahr hier verbracht.

[1] http://www.isys-vision.de
[2] http://www.ensenso.com

1. Web-Oberfläche für das IVS

Das erste größere Projekt war die Entwicklung einer Web-Oberfläche für ein bereits bestehendes, jedoch noch nicht vollständiges Bildverarbeitungssystem. Sowohl Webinterface als auch Bildverarbeitung waren bereits vorhanden, jedoch nicht für den aktuellen Anwendungszweck geeignet und somit war eine Überarbeitung unabdingbar.

Die Aufgabe des betreffenden Systems ist die Ausrichtung einzelner Keramikseiten, welche dann in weiteren Schritten verarbeitet werden. Dabei geht es um die Produktion von Lambdasonden, wie sie in der Abgasfilterung von Autos zum Einsatz kommen.

Diese Keramik-"sheets" werden dabei mit hochpräziser Genauigkeit ausgestanzt, weswegen sie vor dem Stanzen entsprechend ausgerichtet werden müssen. Eine physikalische Ausrichtung ist dabei so gut wie unmöglich, da die gewünschte Genauigkeit dabei nicht erreicht werden kann.

Das von isys entwickelte System erkennt nun mithilfe zweier Kameras die Lage zweier Marken auf dem Sheet und somit die Lage auf dem verschieb- und rotierbaren Tisch. Mithilfe der Bildverarbeitungssoftware wird nun eine Motorbewegung errechnet, welche eine Ausrichtung des Werkstücks bewirkt. Anschließend kann das Teil durch weitere Maschinen in seiner nun festgelegten Position weiterverarbeitet werden.

Die Software besteht dabei nun aus drei verschiedenen Komponenten:

- Dem *ScbDrv*, welcher für die Ansteuerung der Motoren und Hardware verantwortlich ist.
- Dem *IVS* (Isys Vision Server), welcher für die Auswertung der Kameradaten sowie die Bildverarbeitung und Errechnung der Endposition verantwortlich ist.
- Dem *Webinterface*, welches dem Endkunden eine Möglichkeit zur Einsicht des aktuellen Status sowie zur Kontrolle über die Maschine gibt.

Während der ScbDrv nur geringfügig an die neue Maschinenkonfiguration angepasst werden musste, war mein Betreuer mit der Anpassung des IVS an die neuen Marken (in diesem Fall die Ecken der Keramiksheets) beschäftigt. Meine Aufgabe bestand aus dem visuellen Redesign der Weboberfläche, dem Anpassen des Inhaltes (Entfernen überflüssiger Elemente, Hinzufügen von neuen, wichtigen Dingen) sowie der Anpassung der Funktionalität mithilfe von HTML, CSS und JavaScript.

1.1 Einlesen in vorhandenen Code und Redesign

Da das Interface auf der alten, bereits vorhandenen Version aufbauen sollte, war das Einarbeiten in bereits vorhandenen Code notwendig. Um Änderungen vorzunehmen war dabei ein großes Verständnis für diesen Code nötig, da es sich um komplexe und vor allem abstrakte Abläufe handelte, die nicht leicht nachzuvollziehen waren.

1.1.1 Flat-Design mithilfe von CSS

Da meine bisherigen Erfahrungen mit Javascript und jQuery mittelmäßig bis wenig vorhanden waren, machte ich mir zunächst das grafische Redesign der Webseite zur Aufgabe. Mit CSS hatte ich bereits gearbeitet, sodass ich mich verhältnismäßig schnell in das vorhandene Stylesheet einarbeiten konnte und erste Änderungen vornahm.

Dazu gehörten beispielsweise das Entfernen von Schatten (`box-shadow`), was Buttons und Menüs deutlich besser lesbar und sauberer macht, oder die Anpassung von Farben (hellgrau wird ersetzt durch weiß etc). Auch wurde ein Hintergrundbild von mir entfernt und durch eine durchgängige Farbe ersetzt, was zum gewünschten "cleanen" Erscheinungsbild beitrug.

Durch bereits wenige Änderungen im Stylesheet konnte ich hierbei recht große Erfolge erzielen, was die Modernität der Seite angeht. Mit diesen Grundlagen mussten nun Kleinigkeiten angepasst werden, wie beispielsweise zu große Listeneinträge in der Menüleiste, fehlender Umrandung von Buttons oder Ähnlichem. Diese Anpassungen waren minimal, jedoch war der Zeitaufwand dafür immens, da Regeln im CSS immer wieder unterdrückt oder überschrieben wurden und erst das richtige Vorgehen bzw. die dafür verantwortliche Regel ermittelt werden musste.

Schlussendlich waren die Anpassungen jedoch vorgenommen und ich konnte mich auf Basis eines sauberen, nutzbaren und leserlichen Designs dem Verständnis und der Modifikation des Javascript-Frameworks widmen.

1.1.2 Javascript-Framework

In der bereits vorhandenen Version der Webseite wurde sämtliche Funktionalität durch ein Javascript-Framework geregelt, welches unabhängig von Inhalt und Design steht. Im Folgenden ging es nun darum, den hier geschriebenen Code durchzugehen, nachzuvollziehen, zu verinnerlichen und zu verstehen. In etwa 1500 Zeilen waren Initialisierungsfunktionen, Kommunikationsmodule und Eventhandler für Buttons etc. undokumentiert aufzufinden.

Aufgrund der trotz weniger Codezeilen doch sehr hohen Komplexität und vor allem den fehlenden Kommentaren gestaltete es sich als sehr schwierig, sich einzuarbeiten. Ein weiterer Nachteil war meine mangelnde Erfahrung mit Javascript; durch viel Recherche und weiterführende Beispiele konnte ich mir jedoch einiges herleiten und somit verstehen.

Meine erste Änderung war hierbei das Ersetzen von javascript-basierten Zoom-Buttons durch eine Leiste (in HTML ein `<input range>`-Element). Dies sah zum einen deutlich besser aus und war einfacher zu bedienen, außerdem war somit der Inhalt (in diesem Fall die Zoom-Leiste im HTML) besser von der Funktion getrennt.

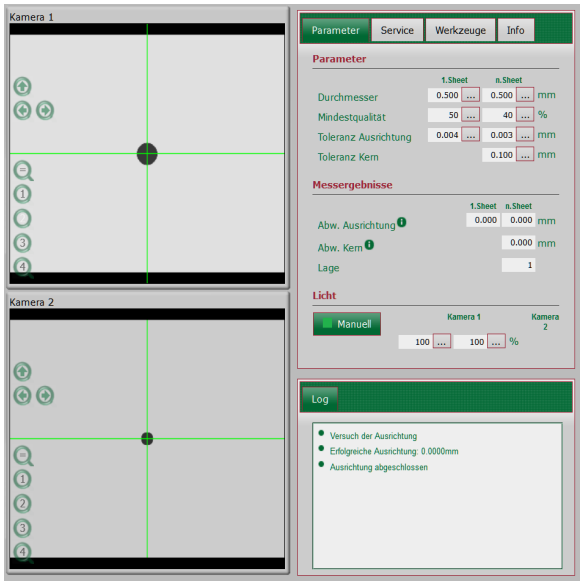

Abbildung 1.1: Interface vor dem Redesign

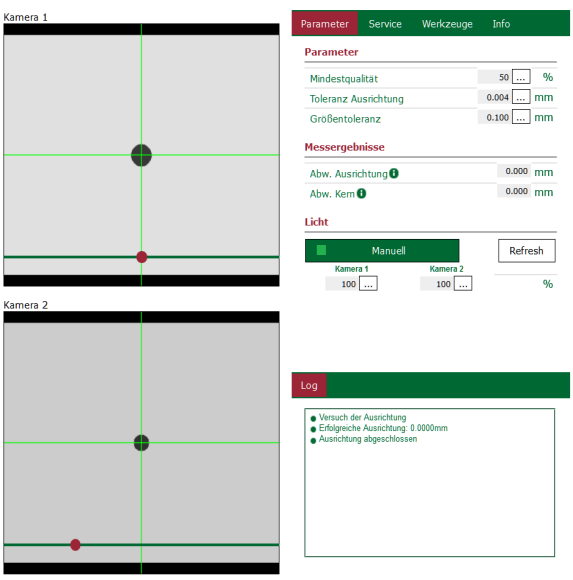

Abbildung 1.2: Interface nach dem Redesign

Weitere Modifikationen am Framework umfassten vor allem die Vereinfachung und Lesbarkeit des Codes. Die Hauptaufgabe in dieser Phase war das Verständnis, da dies die Grundlage für weitere Arbeit am Javascript ist.

1.2 Einschub: Filterfunktion für SVN-Repositories

Da mein Betreuer, welcher mit mir gemeinsam für die Entwicklung des neuen IVS-Systems verantwortlich war, krankheitsbedingt eineinhalb Wochen fehlte, war die weitere Arbeit an der Weboberfläche für diese Zeit leider nicht möglich. In der Zwischenzeit wurde mir dafür die Verbesserung der Oberfläche des Intranets aufgetragen.

Das Intranet enthält eine Liste von SVN-Repositories, die in Form eines Baumes dargestellt werden. Die einzelnen Einträge sind dabei in verschiedenen Ebenen enthalten. Im folgenden Codebeispiel (HTML) wird dies deutlich:

```
<ul class="mktree" id="mktree1">
  <li id="1">Firma 1</li>
    <ul>
      <li id="1-1">Repository 1-1</li>
      <li id="1-2">Repository 1-2</li>
    </ul>
  </li>
  <li id="2">Repository 2</li>
  <li id="3">Firma 3
    <ul>
      <li id="3-1">Repository 3-1</li>
      <li id="3-2">Projekt 3-2
        <ul>
          <li id="3-2-1">Repository 3-2-1</li>
```

```
15          <li id="3-2-2">Repository 3-2-2</li>
16        </ul>
17      </li>
18    </ul>
19  </li>
20 </ul>
```

Listing 1.1: Beispielcode zu den SVN-Repositories im Intranet

Zur Darstellung des Baumes in aus- und einklappbarer Form wurde das schon vorhandene Script `mktree` von Matt Kruse[3] verwendet.

Meine Aufgabe bestand nun darin, eine Suchfunktion für Einträge in dieser Liste zu implementieren. Dazu habe ich mithilfe von `JavaScript` und `jQuery` gearbeitet und einige Funktionen geschrieben. Diese Funktionen basieren zudem auf dem oben genannten Script (`mktree`) und bauen auf dessen Funktionalität auf.

```
1  function filter() {
2    var input = document.getElementById('searchInput');
3    var filter = input.value.toLowerCase();
4    var url = window.location.href.split("?")[0];
5    document.getElementById("searchURL").href = url + "?filter=" + filter;
6    //Search through all list items
7    var li = $('li');
8    for (var i = 0; i < li.length; i++) {
9      //Get all parents of this node which have the filter.
10     var parents = $(li[i])
11       .parents('li')
12       .filter(function() {
13       return this.textContent
14         .trim()
15         .split('␣␣␣␣')[0]
16         .toLowerCase()
17         .indexOf(filter) > -1;
18     });
19     //Select the correct String to search in.
20     var str = li[i]
21       .textContent
22       .split('␣(')[0] //Remove the ( browse / ...)
23       .toLowerCase()
24       .trim()
25       .split("␣␣␣␣")[0]; //Bug fix
26     if (str.indexOf(filter) > -1 || parents.length) {
27       //Item or Parent contains filter
28       li[i].style.display = ""; //Display normally
29       $(li[i]).parents().each(function() {
30         this.style.display = "";
31       });
32       if (li[i].id != "") {
33         expandToItem('mktree1', li[i].id); //Expand tree
34       }
35       window.scrollTo(0, 0); //Fixes weird behaviour
```

[3] http://www.mattkruse.com/javascript/mktree

```
36      } else {
37        li[i].style.display = "none"; //Hide item
38      }
39    }
40  }
```

Listing 1.2: Filterfunktion für das Intranet

Die oben geschriebene Funktion wird aufgerufen, sobald sich der Eingabewert des `<input>`-Felds (siehe Zeile 8) ändert. Eine Änderung der Eingabe hat somit zur Folge, dass die darunter stehende Liste von Repositories neu gefiltert wird und nicht passende Einträge entsprechend versteckt werden.

Weiterhin wird die Funktionalität von `mktree` genutzt, um den Baum zu den angezeigten Repositories automatisch aufzuklappen. Dies wird durch einen einfachen Aufruf der richtigen Funktion erreicht.

Nach der Implementierung dieser Funktionalität habe ich noch eine weitere Verbesserung an der Seite durchgeführt, wodurch sich der Cursor nicht im Textfeld befinden muss, um zu suchen. Dies hat den Vorteil, dass direkt nach dem Laden der Webseite oder auch nach dem manuellen Ein- bzw. Ausklappen des Baumes ohne weitere Mausbetätigung gesucht werden kann.

```
1  document.onkeypress = function(event) {
2    //Variables for search field and pressed key
3    var input = document.getElementById('searchInput');
4    var key = event.key;
5    if (document.activeElement !== input && key.length === 1) {
6      //Not focused => Put the pressed key in the search bar.
7      input.value += key;
8    }
9    //Jump to the search field.
10   input.focus();
11 }
```

Listing 1.3: Funktion für Suche auf Tastendruck

Um dies zu erreichen, habe ich eine Funktion geschrieben, welche beim Tastendruck aufgerufen wird. Die gedrückte Taste wird gespeichert und das Textfeld fokussiert. Handelt es sich bei der Taste um einen Buchstaben (nicht etwa `backspace`, daher `key.length === 1`), so wird der Wert der Taste zusätzlich ins Textfeld geschrieben, um einen doppelten Tastendruck zu vermeiden.

Eine letzte Optimierung der Seite erfolgte durch die Möglichkeit, eine Suchanfrage in Form der URL zu speichern (ein sogenannter Permalink) und somit weiterleitbar zu machen. Meine Vorgehensweise dafür war, den Filter mit dem entsprechenden Tag `?filter=` an die URL anzuhängen. Beim Laden der Seite wird die URL dann nach dem entsprechenden Tag durchsucht, und falls ein Tag gefunden wird, wird der Inhalt des Suchfeldes gesetzt und die Filterfunktion aufgerufen.

Weiterhin wird auf der Webseite ein Hyperlink dargestellt, welcher immer zur aktuellen Suchanfrage verlinkt. Durch Kopieren dieses Links kann die entsprechende Suche also weitergeleitet werden. Erreicht wird dies durch das Aktualisieren der dabei hinterlegten URL nach dem Aufruf von `filter()`.

```
1  document.ready = function() {
2    //First, let's do some ugly bug fixing for mktree (-_-)
3    setDefault("treeClass","mktree");
4    setDefault("nodeClosedClass","liClosed");
5    setDefault("nodeOpenClass","liOpen");
6    setDefault("nodeBulletClass","liBullet");
7    setDefault("nodeLinkClass","bullet");
8    setDefault("preProcessTrees",true);
9
10   var href = window.location.href;
11   //Filter the URL to find the filter.
12   var filter = undefined;
13   try {
14     filter =  href.split('?')[1] //after first "?"
15               .split('filter=')[1] //after "filter="
16               .split('&')[0]; //stop at next "&"
17   } catch (e) {}
18   if (filter != undefined) {
19     //Filter found => search for it
20     document.getElementById('searchInput').value = filter;
21     document.getElementById('searchInput').onkeyup();
22   }
23   document.getElementById("searchURL").href = window.location.href;
24 }
```

Listing 1.4: Filterung beim Laden der Seite

Beim Laden der Webseite wird nun also der Filter in der URL ausgelesen und ins Textfeld eingefügt.

1.3 Dokumentation des Frameworks

Eine weitere wichtige Aufgabe war das Dokumentieren des bereits vorhandenen Codes. Da der ursprüngliche Autor leider nicht mehr bei isys vision angestellt war und ich mich nun schon länger damit auseinander gesetzt hatte, sollte nun eine umfassende Dokumentation des Aufbaus erstellt werden.

Dies erleichtert einerseits die Arbeit am Framework selbst (Änderungen und Updates beispielsweise) und bietet denjenigen, die es nur verwenden (z.B. für eine neue Webseite mit demselben Framework) ein einfaches, gut verständliches Interface. Weiterhin können komplexere Vorgänge wie die Datenübertragung anschaulich erklärt und mithilfe von Diagrammen verdeutlicht werden.

1.3.1 Codekommentierung mit JSDoc

Im ersten Schritt arbeitete ich dabei nur am Quelltext selbst, den ich zuvor schon in lesbare Form gebracht hatte. Durch zusätzliche Kommentare an wichtigen und irreführenden Stellen konnte ich dabei eine erste Grundlage schaffen. Die größte Änderung war die Einführung einer Codedokumentation im JSDoc-Format[4], ähnlich zu den bei Java eingesetzten JavaDoc-Kommentaren.

[4] https://en.wikipedia.org/wiki/JSDoc

Die Vorteile davon liegen auf der Hand: Jede Methode bzw. Funktion hat somit eine eigene Beschreibung, in der Funktionsweise, Parameter, Rückgabewert usw. beschrieben werden.

Ein weiterer Vorteil von JSDoc ist die Möglichkeit, die Dokumentation automatisch zu generieren. Dies ist mit dem JSDoc-Tool von `Node.js` möglich, was nach der Installation einfach über den Befehl `jsdoc file.js` aufgerufen kann. Somit wird ein gut strukturiertes Dokument (inklusive Referenzen) automatisch erstellt.

1.3.2 Beschreibung der API

Nachdem der Quellcode selbst dokumentiert war, konnte ich mich nun mit den internen Abläufen des Frameworks beschäftigen und diese anhand von Grafiken anschaulich machen. In diesem Fall ging es vor allem um die Daten- und Befehlsübertragung vom IVS zum Webinterface und umgekehrt.

Die Übertragung der Daten erfolgt hierbei über eine `XMLHttpRequest`, bzw. (bei älteren Browsern) über ein `ActiveXObject`. Die Befehlscodes sind dabei im HTML-Code der betreffenden Buttons gespeichert:

```
1 <input type="text" size="6" maxlength="6" name="lms/SheetF.MinQuality">
2 <button name="lms/Cmd" value="10">Ausrichtung</button>
3 <button name="lms/Cmd" value="20">
4   <ul>
5     <li>LmsScb/Mot.RefOk</li>
6     <li>Referenzfahrt</li>
7   </ul>
8 </button>
```

Listing 1.5: Ins HTML eingebettete Befehle für das IVS

In Zeile 1 wird beispielsweise das Register `lms/SheetF.MinQuality` geschrieben und auf einen sechsstelligen Wert gesetzt, der vom Nutzer eingegeben wird.
In Zeile 2 befindet sich ein einfacher Befehlsbutton, der den Wert `10` ins Register `lms/Cmd` schreibt (beim IVS ist dies der Code für eine Ausrichtung).
In den Zeilen 3 - 8 findet sich nun ein Knopf mit Status, der

- den Wert `20` ins Register `lms/Cmd` schreibt und somit eine Referenzfahrt auslöst
- innerhalb der unsortierten Liste (Zeile 5) den Wert `LmsScb/Mot.RefOk` liest (Status der Referenzfahrt).

Die Liste (`<ul>`) dient dabei ausschließlich der Darstellung und hat keine Auswirkung auf die Funktion; die Listenelemente werden innerhalb des Buttons entsprechend formatiert. Weiterhin wird je nachdem, was im gelesenen Register `LmsScb/Mot.RefOk` steht, ein entsprechendes Label gesetzt (grün falls Referenzfahrt OK, rot falls nicht).

Im Javascript-Framework wird nun ein Event-Listener für alle Buttons und Eingabefelder erstellt, der dann die entsprechenden Befehle erkennt. Diese werden in Form einer URL in einer Liste gespeichert, welche dann mithilfe der bereits genannten `XMLHttpRequest` bzw. des `ActiveXObjects` ans IVS gesendet werden. Die Antwort des IVS wird dann entsprechend weiterverarbeitet und beispielsweise in Form eines Kamerabildes direkt ausgegeben.

2. Integration von Jira und Confluence

Das nächste größere Projekt war die Integration der Atlassian-Produkte *Jira*[5] und *Confluence*[6] in den Firmenalltag. Diese beiden Programme sind für das Production Management gedacht und haben das Ziel, einen durchgängigen, geplanten und leicht nachvollziehbaren Arbeitsfluss zu erreichen.

Confluence ist dabei ein einfaches System, dessen Ziel die einfache Weitergabe von Informationen ist. Es dient der Erstellung von Dokumenten, welche externe Daten wie beispielsweise Microsoft-Office-Dokumente oder YouTube-Videos sowie Bilder usw. enthalten können und vereinfacht die Zusammenarbeit mehrerer Personen an einem Dokument.

Jira arbeitet im Zusammenspiel mit Confluence und ist ursprünglich für die Softwareentwicklung gedacht. Entwicklungsvorgehensweisen wie *Scrum* o.Ä. können hier ohne Umwege umgesetzt werden und Jira bietet Funktionalitäten wie Sprints oder Burndown-Charts an, um den Arbeitsfluss zu sichern.

Die Verwendung von Jira und Confluence sollte das davor eingesetzte (und veraltete) Tool *easyPM* ersetzen, welches für die Planung von Aufgaben, Abläufen und Verwaltung von Zeitspalten eingesetzt wurde. Informationen wie E-Mails, Anrufprotokolle oder sonstige Daten waren hier jedoch dezentral, bzw. konnten nicht direkt in easyPM gespeichert werden. Durch das Zusammenspiel der beiden neuen Systeme sollte dieses Problem behoben und die Speicherung der Daten konsistent werden.

2.1 Einrichtung der Datenbank

Jira und Confluence sind beides Tools, die über einen Server laufen und dann über den Webbrowser der Endgeräte aufgerufen werden können. Dies sichert zum einen die Zentralität der Datenspeicherung und -verarbeitung und entlastet andererseits die Nutzergeräte stark. Weiterhin ist die Nutzung eines firmeninternen Servers (welcher außerdem für andere Zwecke wie das Intranet schon vorhanden war) sinnvoll, da somit die Sicherheit der gespeicherten Daten gewährleistet wird.

Für den Aufbau des Webservers wird dabei sowohl von Jira als auch von Confluence eine Datenbank verwendet. Das kann sowohl eine *MySQL*-, *Oracle*- oder *Microsoft-SQL*-Datenbank sein, oder aber auch die Open-Source-Alternative *PostgreSQL*. Wegen der eher schlechten Unterstützung von MySQL habe ich schlussendlich eine PostgreSQL-Datenbank eingesetzt.

PostgreSQL ist ein freies und quelloffenes Datenbankmanagementsystem. Es wird von Betriebs-

[5] https://de.wikipedia.org/wiki/Jira_(Software)
[6] https://de.wikipedia.org/wiki/Confluence_(Atlassian)

systemen wie Windows, Linux und anderen UNIX-ähnlichen Systemen unterstützt und wird bei den meisten Linux-Distributionen bereits mitgeliefert.

Um den Linux-Server von isys zu simulieren und die Datenbank entsprechend einzurichten, habe ich eine virtuelle Maschine mit einer Ubuntu-Installation verwendet. Nach der Einrichtung des Systems konnte ich mit der Konfiguration des Datenbankservers beginnen. Nach einiger Verwirrung und etwas Recherche stellte sich heraus, dass für das Erstellen einer Tabelle in PostgreSQL[7] ein Linux-Nutzername angegeben werden muss. Dies war praktisch, da die Jira- und Confluence-Systemdienste sowieso einen eigenen Account erstellten, den ich somit direkt mitverwenden konnte. Ich hatte nun also drei Zugänge:

- `ubuntu`, Super-User-Account und Hauptaccount des Rechners,
- `confluence`, für die Datenbank und den Systemdienst des Confluence-Servers,
- `jira`, für die Datenbank und den Systemdienst des Jira-Servers.

Mit einigen einfachen SQL-Befehlen waren die notwendigen Datenbanken dann auch schnell erstellt:

```
1 CREATE DATABASE confluencedb WITH ENCODING 'UNICODE';
2 GRANT ALL PRIVILEGES ON DATABASE confluencedb TO confluence;
3
4 CREATE DATABASE jiradb WITH ENCODING 'UNICODE';
5 GRANT ALL PRIVILEGES ON DATABASE jiradb TO jira;
```

Listing 2.1: SQL-Befehle zur Erstellung der Datenbanken

Zur Anbindung an Jira bzw. Confluence musste nun nur noch bei der Einrichtung jeweils die entsprechenden Datenbankdaten angegeben werden. Da hier der Jira- bzw. Confluence-Server und die Datenbank auf dem selben Rechner laufen, ist der Host beispielsweise `127.0.0.1` bzw. `localhost`.

Die Installation von Jira und Confluence erfolgte dann einfach über die Ausführung eines Shell-Skriptes und die Eingabe einer Lizenz. Somit war die Einrichtung der Systeme vollständig und ich konnte einige Einstellungen vornehmen, z.B. Anpassungen im Design der Seiten, sodass die Oberfläche den Farben der Firma entsprach (Grün [#006833] und Rot [#9B2536]).

2.2 Einrichtung des Intranetservers

Nun, da Jira und Confluence online und konfiguriert waren, setzte ich mich mit der Einrichtung eines Servers auseinander, welcher im Intranet betrieben werden und die beiden Webtools hosten sollte.

Im ersten Schritt machte ich den dafür zur Verfügung gestellten Rechner nutzbar, indem ich eine frische Linux-Installation vornahm. Auf dieser Basis konnte ich nun mit `VirtualBox`[8] eine virtuelle Maschine einrichten; auch darauf lief ein Linux-Betriebssystem. Die Nutzung einer virtuellen Maschine hatte hierbei mehrere Vorteile:

[7] `https://de.wikipedia.org/wiki/PostgreSQL`
[8] `https://wiki.ubuntuusers.de/VirtualBox/Installation/#VirtualBox-OSE-Open-Source-Edition`

1. **Flexibilität**: Durch das Speichern in einem **Disk Image**[9] ist der Ort des Servers sehr leicht austauschbar. Durch Verschieben des Images sind sämtliche Daten des Servers übernommen, was die Wartung stark vereinfacht.

2. **Sicherheit**: Da die Daten in einem einzigen Image liegen, welches auf der Festplatte des Wirtrechners gespeichert wird, ist die Datensicherung bzw. Erstellung von Backups besonders komfortabel. Mithilfe eines RAID-Systems wird in diesem Fall die Datensicherheit gewährleistet und Backups erstellt.

3. **Vielseitigkeit**: Der Server läuft in einer virtuellen Maschine, welche auf die Ressourcen des Wirtrechners zugreift. Durch das Einrichten einer weiteren virtuellen Maschine kann dessen Rechenleistung und Speicherkapazität optimal ausgenutzt werden und die Maschine somit auch für andere Zwecke verwendet werden.

Nach erneuter Einrichtung der in Kapitel 2.1 beschriebenen Datenbank und der Anbindung an Jira/Confluence musste ich mich nun noch mit der Netzwerkverbindung auseinandersetzen. Die zugeordnete IP-Adresse sollte dabei `10.0.0.14` sein; Jira und Confluence jeweils unter den Standardports `8080` bzw. `8090` erreichbar sein.

Als Erstes musste die virtuelle Maschine über eine Netzwerkbrücke auf das lokale Netzwerk zugreifen können. Dies war schnell eingestellt; eine einfache Änderung im VirtualBox-Interface genügte dafür. Mithilfe der Netzwerkbrücke wird hier das lokale Netzwerk (isys vision Intranet) mit dem von der virtuellen Maschine erstellten Netzwerk verbunden, sodass diese die gehosteten Services auf die Webports anlegen kann.

Im nächsten Schritt braucht die virtuelle Maschine eine feste (statische) IP-Adresse. In diesem Fall sollte dies die `10.0.0.14` sein. Um diese zuzuweisen, kann unter Linux die Datei `/etc/network/interfaces` bearbeitet werden:

```
1 auto enp0s3 #Name der Netzwerkbrücke
2 iface enp0s3 inet static #IP auf Statisch ändern
3 #Setzen von Verbindungsinformationen
4 netmask 255.255.0.0
5 gateway 10.0.0.10
6 network 10.0.0.0
7 broadcast 10.0.0.255
8 dns-nameservers 10.0.0.10
```

Listing 2.2: Änderungen in `/etc/network/interfaces`

Mithilfe dieser Änderungen wird dem Rechner eine statische Adresse zugewiesen. Dies ist hier notwendig, da der Rechner sonst bei jedem Start vom Router eine neue, variable Adresse zugewiesen bekommt. Damit der Server also immer mit der selben Adresse erreichbar ist, muss die IP fest sein. Später lässt sich auch eine DNS zuweisen, beispielsweise `intranet.isys-vision.de/jira`.

2.3 Ausfallsicherheit über ein RAID-System

Um die Stabilität des Servers im Intranets zu gewährleisten, sollte das System durch ein RAID-5[10] abgesichert werden. Ein RAID-5 besteht aus mindestens 3 unabhängigen Festplatten, welche

[9] `https://de.wikipedia.org/wiki/Speicherabbild`
[10] `https://de.wikipedia.org/wiki/RAID#RAID_5`

dieselben Daten redundant abspeichern. Beim Ausfall einer der Platten wird somit sichergestellt, dass die Daten weiterhin verfügbar (und auch nutzbar) bleiben.

Da Hardware-RAIDs oft sehr teuer sind, habe ich auf ein Software-RAID zurückgegriffen. Hierbei wird die Redundanz der Daten durch Software erreicht, welche das Plattenmanagement übernimmt. Das Betriebssystem verblieb dabei auf der vorherigen Platte, sodass nur die virtuelle Maschine auf dem RAID gespeichert wird.

Unter Linux lässt sich ein RAID mit dem Tool `mdadm` sehr gut realisieren. Man kann dabei auswählen, welches RAID-Level gewählt (in diesem Fall ein RAID-5-System) und welche und wie viele Festplatten genutzt und werden sollen. Dazu reicht ein einziger Befehl aus:

```
1 sudo mdadm --create --verbose /dev/md0 --level=5 --raid-devices=3
    /dev/sdb /dev/sdc /dev/sdd
```

`/dev/sdb`, `/dev/sdc` und `/dev/sdd` sind die Festplatten, die genutzt werden sollen. Unter `/dev/md0` wird nach Abschluss des Vorgangs das neue "Gerät" liegen, es lässt sich wie eine einzelne Festplatte behandeln. Das System wird nun eingerichtet, und mithilfe von

```
1 cat /proc/mdstat
```

kann man den aktuellen Status ansehen. Das Einrichten dauert je nach Festplattengeschwindigkeit und -größe sehr lange.

Nach Abschluss dieser Einrichtung muss ein Dateisystem für die Festplatten festgelegt werden. Unter Linux wird standardmäßig das Ext4-System verwendet. Es lässt sich mit

```
1 sudo mkfs.ext4 -F /dev/md0
```

installieren. Weiterhin muss ein Einhängepunkt für die Festplatte erstellt werden (also ein Verzeichnis im Dateisystem, von dem aus darauf zugegriffen werden kann). Dies geschieht mit

```
1 sudo mkdir /media/md0
```

Durch einen entsprechenden Eintrag in der Datei `/etc/fstab` kann die Festplatte nun beim Systemstart automatisch eingehängt werden. Die neue Zeile lautet

```
1 /dev/md0 /media/md0 ext4 defaults 0 2
```

In der ersten Spalte wird der Ort der Festplatte angegeben, danach der Einhängepunkt. Es folgen das Dateisystem und weitere Optionen (defaults reicht hier aus).

Das RAID-System ist nun vollständig konfiguriert. Nach einem Neustart des Computers kann das Speicherabbild der virtuellen Maschine nun darauf gespeichert und die Daten somit ausfallsicher gemacht werden.

3. Web-Tools für das Mikado-Projekt

Nach meiner Arbeit am IVS, Jira und Confluence wurde ich nun Teil eines anderen Projekts namens Mikado[11]. Mikado ist ein Robotikprojekt, bei dem es hauptsächlich um das sogenannte "Bin Picking" geht. Dabei wird ein Roboter dazu eingesetzt, zufällig in einer Kiste liegende Teile mithilfe einer 3D-Kamera zu erkennen, zu greifen und in eine andere Position zu bringen. Die Herausforderung liegt hierbei beim Leeren der Kiste, d.h. alle sich darin befindlichen Teile gleichgültig ihrer Position greifen zu können.

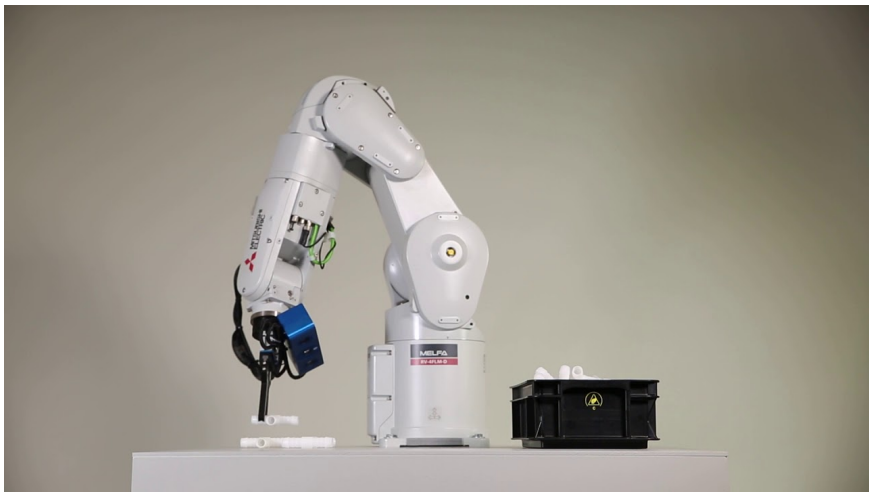

Abbildung 3.1: Mikado-Roboter von Mitsubishi

3.1 Demoablauf für Messen

Zum Kennenlernen der Programmschnittstelle und des Roboters sollte ich zunächst mit dem von isys geschriebenen Programm einen Demoablauf aufbauen, welcher für Messen oder ähnliche Anwendungsfälle verwendet werden sollte. Im Programm lassen sich dafür Abläufe graphisch regeln, beispielsweise eine Bewegung, das Greifen eines Teils oder die Bildaufnahme mit der 3D-Kamera. Diese noch nicht veröffentlichte Software konnte somit außerdem von mir getestet und Verbesserungsvorschläge geäußert werden.

Zur Erkennung der Teile wird mithilfe der Kameras ein 3D-Bild aufgenommen, welches anschließend von der Recheneinheit analysiert und zu einer Punktwolke transformiert wird. Diese Punktwolke wird nun mit dem eintrainierten CAD-Modell abgeglichen und somit mögliche Kandidaten anhand ihrer Übereinstimmung identifiziert. Die dabei gewählte Mindestübereinstimmung lässt sich dabei einstellen, was je nach Wert Vor- bzw. Nachteile mit sich bringt:

[11] `http://www.mikado-robotics.de`

Höhere Mindestqualität	Niedrigere Mindestqualität
Erkannte Teile können mit höherer Sicherheit richtig und sicher gegriffen werden	Erkannte Teile können sich in nahezu ungreifbarem Winkel befinden und daher nicht sicher gegriffen werden
Weniger Teile werden erkannt, daher höhere Chance, die Kiste nicht leer zu bekommen	Viele Teile werden erkannt, die Wahrscheinlichkeit ist daher größer, alle verbleibenden Teile zu identifizieren

Tabelle 3.1: Vor- bzw. Nachteile einer höheren vs. niedrigeren Mindestqualität

Die Bewegungen des Robotors lassen sich entweder über die Winkel der einzelnen Gelenke (beim von isys verwendeten Roboter von Mitsubishi Electrics sind es 6 Gelenke) oder das Format `xyz rpy` (`xyz` ist hierbei die Position im Raum (3 Koordinaten), `rpy` (roll, pitch, yaw) die Drehung des Greifers) angeben. Sämtliche Bewegungen und Rotationen sind dabei kollisionsgeprüft, d.h. sobald das System eine mögliche Kollision erkennt, wird die Bewegung nicht ausgeführt. In diesem Fall kann eine alternative Aktion durch einen Sprung im Programmablauf durchgeführt werden.

Über einzelne Sektionen und Sprünge lässt sich nun ein strukturiertes Programm erstellen, was sowohl einfache Funktionen (wie in diesem Fall eine Demonstration der Funktionalität) oder auch komplexe Abläufe wie den Zusammenbau eines Schneckengewindes übernehmen kann. Der Programmablauf kann dabei mit den Funktionen `start`, `pause`, `step` und `reset` gesteuert werden, was in folgenden Aufgaben noch wichtig wird.

Mein Demonstrationsprogramm war hierbei die Abwandlung einer älteren Demo. Der Roboter nimmt dabei Zahnräder aus einer Kiste, testet mithilfe einer Lichtschranke, ob er das Teil tatsächlich gegriffen hat, und legt dieses schlussendlich so ab, dass der Schriftzug "isys" entsteht. Ist dies geschehen, nimmt er nacheinander die abgelegten Teile wieder auf und legt sie zurück in die Kiste.

Der Zweck der Demonstration ist einerseits das Heranführen an den Roboter an sich (z.B. die Arbeitsweise der Maschine oder die graphische Programmierung) sowie die Fähigkeit, wirklich sämtliche Teile aus der Kiste nehmen zu können, zu demonstrieren. Erreichen lässt sich dies einfach dadurch, dass zu Beginn nur so viele Zahnräder in die Kiste gelegt werden, wie am Ende auch für den Schriftzug benötigt werden.

3.2 Web-Remote für die Mikado-Software

Für die Demonstration bei Messen oder bei ähnlichen Veranstaltungen sollte nun von mir ein Webinterface erstellt werden, was als Fernzugriff für die Mikadosoftware verwendet werden kann. Der Sinn dahinter ist, dass der Programmablauf nicht mehr ausschließlich mithilfe von Bildschirm und Tastatur gesteuert, sondern beispielsweise über ein Smartphone modern und komfortabel bedient werden soll. Dies ist insbesondere nützlich, da aus Sicherheitsgründen bei Vorführungen eine Glaswand vor dem Roboteraufbau ist, was Eingriffe in den Ablauf stark erschwert.

3.2.1 Design der neuen Webseite

Nachdem ich im 1. Projekt bereits mit Javascript, CSS und HTML in Berührung gekommen war, konnte ich nun eine völlig neue Webseite basierend auf meinen erlangten Kenntnissen selbstständig und von Grund auf aufbauen. Dazu kümmerte ich mich zuerst um Inhalt und Design der neuen Seite und fügte anschließend die Funktionalität durch Javascript und jQuery hinzu.

Struktur mit HTML

Zunächst gliederte ich die neue Webseite mithilfe der zugehörigen HTML-Tags in sinnvolle Abschnitte. Mit `<nav>`, `<main>` und `<footer>` sowie geschachtelten `<div>`-Tags konnte ich eine sinnvolle Struktur in den Inhalt der Seite bringen.

Im `<nav>`-Teil der Seite wird die Navigationsleiste definiert; in meinem Fall umfasste sie die Listenpunkte **Logs**, **Kontrolle** und **Über** sowie ein **Hamburger-Menü-Icon**[12], welches zum Ein- und Ausklappen des Menüs notwendig war. Im `<main>`-Teil des HTML-Dokuments wird sämtlicher Inhalt der Seite definiert, also in meinem Fall das Logfenster, die Kontroll- und die Informationsseite. Diese einzelnen Abschnitte sind jeweils wieder in einzelne `<div>`s geteilt, sodass Struktur und Design gewährleistet werden können. Schlussendlich befinden sich im `<footer>`-Element einige Informationen über das Produkt, wie z.B. Copyright, Webseite oder Kontakt.

```
<div id="control" class="content">
  <div class="control-window">
    <div class="control-element">
      <button onclick="start()">
        <img src="media/start.svg" alt="Start">
      </button>
    </div>
    <div class="control-element">
      <button onclick="step()">
        <img src="media/step.svg" alt="Step">
      </button>
    </div>
    <div class="control-element">
      <button onclick="pause()">
        <img src="media/pause.svg" alt="Pause">
      </button>
    </div>
    <div class="control-element">
      <button onclick="reset()">
        <img src="media/reset.svg" alt="Reset">
      </button>
    </div>
  </div>
</div>
```

Listing 3.1: Struktur der Kontrollseite

Wie man sieht, bestehen die Kontrollbuttons aus einem Funktionsaufruf (den ich erst später

[12] `https://de.wikipedia.org/wiki/Hamburger-Men%C3%BC-Icon`

implementiert habe) und einer SVG-Grafik[13].

Erstellen von SVG-Grafiken

SVG-Grafiken haben den großen Vorteil, dass sie vom Browser gerendert werden. Dadurch ist einerseits keine große Bilddatei notwendig, was (insbesondere auf mobilen Endgeräten) zu sehr schnellen Ladezeiten führt, und andererseits auch nicht das Problem zu geringer Auflösungen mit sich bringt, wodurch die Grafik immer gestochen scharf ist. Da jedoch keine entsprechenden Grafiken vorhanden waren, musste ich diese selbst erstellen. Auch für das Navigationsmenü habe ich SVG-Dateien entworfen.

Der Inhalt einer SVG-Grafik ist der Syntax von HTML bzw. XML sehr ähnlich. Zunächst wird mit dem Tag <svg> gekennzeichnet, dass es sich um eine SVG-Grafik handelt. Innerhalb dieses Tags können nun Bildelemente hinzugefügt werden:

```
1  /<svg width="54" height="54" viewBox="0␣0␣54␣54"
       xmlns="http://www.w3.org/2000/svg" xmlns:xlink=
       "http://www.w3.org/1999/xlink">
2    <circle
3      cx="27" cy="27" r="25"
4      stroke="black" stroke-width="4" fill="none"
5    />
6    <line
7      x1="18" y1="13"
8      x2="18" y2="41"
9      stroke="black" stroke-width="8" stroke-linecap="butt"
10   />
11   <line
12     x1="36" y1="13"
13     x2="36" y2="41"
14     stroke="black" stroke-width="8" stroke-linecap="butt"
15   />
16 </svg>
```

Listing 3.2: Bildelemente in einer SVG-Grafik

Hier wird beispielsweise zunächst ein umschließender Kreis definiert (ll. 2 - 5), welcher den Mittelpunkt $P(27|27)$ und den Radius $r = 25$ hat. Die Linienfarbe soll Schwarz sein, die Linienbreite 4 Pixel und der Kreis soll nicht gefüllt sein. Anschließend werden zwei Linien gezeichnet, jeweils mit der Farbe Schwarz, der Breite 8px und einem geraden Linienende. Linie 1 geht in diesem Fall von $P(18|13)$ nach $Q(18|41)$.

Stilregeln durch CSS

Im nächsten Schritt ging es um das Design mithilfe von CSS. Die vorhergehend definierten Inhalte konnte ich nun schöner Darstellen, indem ich Stilregeln für bestimmte Elemente und Klassen vornahm. Dazu gehörten unter Anderem die Schriftart, Farbschema für Menüleiste und Links oder auch Positionierung von Abschnitten und Festlegen von Abständen.

[13] https://de.wikipedia.org/wiki/Scalable_Vector_Graphics

Sehr wichtig war hierbei die Orientierung an relativen Werten wie der Bildschirmweite (`viewport width`). Da es sich hierbei um ein für Mobilgeräte optimiertes Webinterface handelt, spielt Skalierbarkeit auf unterschiedliche Größen eine große Rolle. Umsetzen lässt sich dies beispielsweise durch ein sogenanntes **Responsive Design**[14], bei dem die Webseite der Bildschirmgröße entsprechend reagiert. Dies sichert auch eine gute und intuitive Darstellung beim Wechsel vom Portrait- in den Landschaftsmodus (z.B. bei Tablet PCs).

In der Umsetzung erfolgt dies durch die Vermeidung von Pixelwerten (z.B. `width: 12pt;`) und die Ersetzung ebenjener durch relative Werte (wie beispielsweise `width: 1em;`). Somit wird die Größe von Elementen immer von der Bildschirm- oder Schriftgröße abhängig gemacht und passt sich dementsprechend an.

Weiterhin werden mithilfe eines `@media`-Tags im CSS für besonders kleine oder besonders große Geräte andere Regeln angegeben, welche zumeist die davor definierten Regeln überschreiben. Dies verhindert eine unproportionale Darstellung, wie z.B. extrem verkleinerte Bilder, unleserliche Schrift oder aber im Gegenteil riesige Buttons oder Überschriften.

Mit CSS konnte ich weiterhin einige Animationen erreichen, zum Beispiel ein Popup beim Klick auf die Kontrollbuttons. Dies lässt sich über das CSS-Attribut `transition` erreichen. Mit `transition: width .1s linear;` bekommt man beispielsweise einen 100 ms lang andauernden Übergang betreffend der Breite. In meinem Fall hatte ich für `button:hover` einen anderen Stil festgelegt als für einen normalen Button, und zwar ein größeres Icon und weniger Durchsichtigkeit. Beim Übergang zu `button:active` wird das Icon jedoch wieder klein, was einen "Klick"-Effekt zur Folge hat.

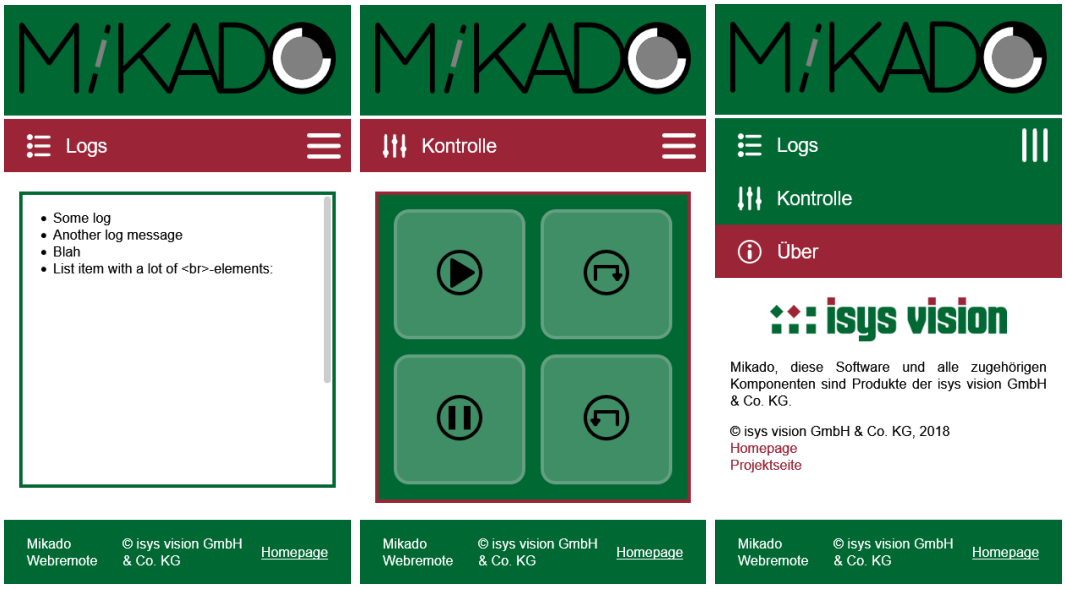

Abbildung 3.2: Logfenster Abbildung 3.3: Kontrolle Abbildung 3.4: Info / Menü

[14] `https://de.wikipedia.org/wiki/Responsive_Webdesign`

3.2.2 Befehls- und Logübertragung mithilfe von RosJS

Auf dem Hostrechner des Roboters läuft zur Kontrolle von diesem eine Python-Bibliothek namens ROS (Roboter Operating System). Sie bietet viele Funktionalitäten, beispielsweise zum Steuern der einzelnen Gelenkwinkel durch Python.

Weiterhin werden Services zur Ausführung von einzelnen Aufgaben bereitgestellt. Der Service `/reset` setzt beispielsweise den aktuellen Programmablauf zurück und kann mit `rosservice call /reset` aufgerufen werden. Es sind außerdem die Services `/run` und `/pause` vorhanden, welche die für mich notwendige Funktionalität beinhalten. Die beiden letzteren werden mit Parametern aufgerufen: `rosservice call /run "step:false blocking:false"`, bzw. `rosservice call /pause "blocking:false"`.

Der Parameter `blocking` ist dabei in meinem Fall irrelevant; umso interessanter ist jedoch das Argument `step`: Ist es auf `false` gesetzt, läuft das Programm beim Aufruf normal durch. Sobald der Service jedoch mit `step:true` aufgerufen wird, wird nur ein Schritt im Programmablauf durchgeführt.

Zur Anbindung an das von mir entwickelte Webinterface habe ich nun die Bibliothek RoslibJS verwendet, welche als Kommunikationsmodul für ROS fungiert. Hiermit können entsprechende Aufrufe im Javascript festgelegt werden, um diese anschließend an den Hostrechner zu senden und somit den zugehörigen Service zu starten.

Meinen `<button>`-Elementen hatte ich jeweils die zugehörige Funktion im Argument `onclick` zugeordnet (Beispiel: `<button onclick="pause()">Pause</button>`). Im zugehörigen Javascript sind nun einmal die generelle Einrichtung von RoslibJS (wie z.B. Verbindungsinformationen etc.) und für jeden Button die zugehörige Funktion vorhanden.

```
1  var ros = new ROSLIB.Ros();
2  ros.on('error', function(error) {/* Log weggelassen */});
3  ros.on('connection', function() {});
4  ros.on('close', function(){});
5  ros.connect('ws://10.0.4.136:8080');
6  /* Irrelevante Funktionen weggelassen */
7  function start() {
8    var rosStart = new ROSLIB.Service({
9      ros: ros,
10     name: '/run',
11     messageType: 'mikado_msgs/Run',
12   });
13   var rosStartRequest = new ROSLIB.ServiceRequest({
14     step: false,
15     blocking: false,
16   });
17   rosStart.callService(rosStartRequest);
18 }
19 /* Funktionsrumpf weggelassen */
20 function step() {};
21 function pause() {};
22 function reset() {};
23 /* Funktion zum Anhängen von Logs an das Logfenster. */
24 function initLog() {
```

```
25    var rosLog = new ROSLIB.Topic({
26      ros: ros,
27      name: '/rosout',
28      messageType: 'rosgraph_msgs/Log',
29    });
30    rosLog.subscribe(function(message) {
31      var log = '<li␣class="';
32      switch (message.level) {
33        case 1:
34          log += "debug";
35          break;
36        case 2:
37          log += "info";
38          break;
39        case 4:
40          log += "warn";
41          break;
42        case 3:
43          log += "error";
44          break;
45      }
46      log += '">' + message.msg + '</li>';
47      addLog(log);
48    });
49  }
```

Listing 3.3: Serviceaufruf durch RoslibJS

Zunächst wird in den ersten 5 Zeilen die Verbindung zum ROS hergestellt. In den anschließend folgenden Funktionen wird zunächst ein neuer Service für die aktuelle Routine erstellt (Zeilen 8 - 12), welcher dann mit einer Anfrage (Zeilen 13 - 16) gestartet wird (Zeile 17). Der Service selbst enthält hierbei Informationen über den aufgerufenen Befehl (in diesem Fall `/run`), während die Anfrage Informationen über die Parameter enthält (hier die Argumente `step` und `blocking`).

In der Funktion `initLog()` in Zeile 24 wird eine Verbindung zum ROS hergestellt, um neue Lognachrichten ans Logfenster anzuhängen. Zunächst wird dafür ein sogenanntes "Topic" erstellt, welches dann mithilfe von `subscribe()` laufend aktualisiert werden kann. Den hier ankommenden Nachrichten wird nun mithilfe des Wichtigkeitslevels eine entsprechende CSS-Klasse zugewiesen (z.B. haben Meldungen vom Typ `debug` die Farbe grün, `error` ist rot usw). Anschließend werden die eigentlichen Logmeldungen an die bereits existierende Liste angehängt und erscheinen somit im Logfenster.

3.2.3 Webhosting und Netzwerk

Nachdem das Interface nun eigentlich die gewünschte Funktionalität besitzt, muss es noch erreichbar gemacht werden. Dazu wird ein HTTP-Server verwendet, welcher die Webseite hostet. Mit der Python-Bibliothek `flask` ist dies recht einfach zu realisieren:

```
1 import rospy
2 from flask import Flask, render_template
3 app = Flask(__name__)
4
```

```
@app.route('/')
def render_static():
    return render_template('index.html')
if __name__ == '__main__':
    app.run()
```

Listing 3.4: Python-Skript zum Hosten des Webservers

Nun lässt sich unter der IP-Adresse des Rechners (lokal `127.0.0.1`) die gehostete Webseite aufrufen. Über ein WLAN-Netzwerk lässt sich nun eine Verbindung zu mobilen Endgeräten aufbauen, welche dann auf das Webinterface zugreifen und somit den Roboter kontrollieren können.

3.3 Web-Preview zur Darstellung von Punktwolken

Zur Objekterkennung beim Mikado-Roboter wird von diesem mithilfe der 3D-Kamera eine sogenannte Punktewolke aufgenommen - eine Ansammlung von Punkten, welche jeweils mit ihren Koordinaten abgespeichert werden. Somit wird eine Art "3D-Gitter" erzeugt, welches dann mit dem vogegebenen Modell verglichen werden kann.

Zur Veranschaulichung sollten diese Punktewolken nun mithilfe von Javascript in einer einfachen Seite visualisiert werden. Die Nutzung eines Javascript-Frameworks ist hierbei sehr sinnvoll und komfortabel, da

1. nur eine einzige Datei erstellt werden muss, welche dann z.B. per E-Mail versendet werden kann
2. durch die Darstellung im Browser nahezu jedes Endgerät die Möglichkeit hat, die generierte Datei darzustellen.

3.3.1 Rendering mit `THREE.js`

Zur Darstellung der Punktewolke habe ich `THREE.js`[15] verwendet. Dabei handelt es sich um ein Framework, welches auf OpenGL bzw. WebGL basiert und Funktionalitäten zum Rendering von Objekten bereitstellt.

Zunächst muss dafür ein HTML-Grundgerüst aufgebaut werden, welches THREE einbindet und als Darstellungsfläche verwendet wird.

```
<html>
  <head>
    <title>Mikado WebPreview</title>
    <script src=''three.min.js''></script>
  </head>
  <body>
    <!--Titel, Leiste, ....-->
  </body>
</html>
```

Listing 3.5: HTML-Gerüst für die WebPreview

[15] `https://threejs.org/`

Nachdem dies geschehen ist, kann der Renderer nun initialisiert werden. Das Framework enthält dabei viele Methoden und Objekte, welche nun entsprechend deklariert und genutzt werden können. Da es sich um viele kleine Konfigurationsschritte handelt, ist folgendes Listing gekürzt.

```
1  function load() {
2    window.addEventListener('keypress', keypress);
3    scene.background = new THREE.Color(0x444444);
4    camera.position.x = 0.44;
5    renderer.setSize(window.innerWidth * 0.75, window.innerHeight * 0.75);
6    loader.load('cloud.pcd', function(mesh) {
7        mesh.material.color.set(0x9b2436); //isys color
8        scene.add(mesh);
9        controls.update();
10   });
11 }
12 function animate() {
13   requestAnimationFrame(animate);
14   renderer.render(scene, camera);
15   controls.update();
16 }
17 /* Handles keyboard input */
18 function keypress(ev) {
19   var pointCloud = scene.getObjectByName('cloud.pcd');
20   switch (ev.key || String.fromCharCode(ev.keyCode || ev.charCode)) {
21     case '+':
22       pointCloud.material.size *= 1.2;
23       pointCloud.material.needsUpdate = true;
24       break;
25     case '-':
26       pointCloud.material.size /= 1.2;
27       pointCloud.material.needsUpdate = true;
28       break;
29   }
30 }
```

Listing 3.6: Initialisierung von `THREE.js`

Die Funktion `load()` wird nun beim Laden der Seite aufgerufen und erstellt einen Renderer, eine Szene (in der das Bild dargestellt wird) und lädt die Punktwolke (Zeile 6). Weiterhin wird beispielsweise die Farbe des Materials festgelegt und das Lesen der Tastatur gestartet (Zeile 9).

Die Funktion `animate()` ist nun eine einfache Update-Funktion, welche das angezeigte Bild aktualisiert und mit jedem Frame aufgerufen wird. Auch hier wird die Tastatureingabe gelesen bzw. auf einen relevanten Tastendruck gewartet.

In der Funktion `keypress()` wird nun die Tastatureingabe ausgewertet: Beim Druck der Taste '+' wird die Größe der Punkte vergrößert, beim Druck der Taste '-' dementsprechend verkleinert.

Abbildung 3.5: Rendering einer Beispielpunktwolke