

ICT Modul 121 – Steuerungsaufgaben bearbeiten

Steuerung eines Arduino Uno Mikrokontrollers via WebSocket

Samuel Hess

samuel.hess@tfbern.ch

Technische Fachschule Bern

12. Mai 2020

Inhaltsverzeichnis

1. Abstract	4
2. Einleitung	4
2.1. Fragestellung	4
2.2. Motivation	4
2.3. Literatur-Review	4
2.3.1. Arduino mit integriertem WLAN	4
2.3.2. WLAN Erweiterung	4
2.3.3. Serial Gateway	5
3. Experimenteller Teil	5
3.1. Informationsquellen	5
3.2. Prinzipskizze	5
3.3. Hardware	5
3.3.1. Anschluss der Sensoren	5
3.4. Software	5
3.4.1. Entwicklungsumgebung	5
3.4.2. Node Libraries	5
3.4.3. Arduino Libraries	6
3.4.4. Arduino Sketch	6
3.4.5. Serial Gateway	6
3.4.6. WebSocket Server	6
3.4.7. Web GUI	6
4. Resultate	6
5. Diskussion	7
6. Zusammenfassung	7
7. Danksagung	7
8. Interessenskonflikte	7
9. Quellenverzeichnis	7
A. Quellcode	10
A.1. Arduino Sketch	10
A.1.1. Funktionen zur Statusausgabe	11
A.1.2. Funktionen zur Steuerung	11
A.2. Serial Gateway	13
A.3. WebSocket Server	14
A.4. Web Client	15

B. Erstellung dieses Dokuments	19
B.1. LaTeX Umgebung	19
B.2. LaTeX Editor	19
B.3. LaTeX Pakete	19

1. Abstract

In der vorliegenden Arbeit wurde untersucht, wie ein Arduino Uno über eine WebSocket-Verbindung gesteuert werden kann, während dieser im Sekundentakt Statusmeldungen versendet.

Der verwendete Funduino Uno R3^[1] verfügt über keine WLAN-Schnittstelle. Daher erfolgt die WebSocket-Kommunikation extern auf einem Gateway. Als Gateway wird ein Windows-PC verwendet. Arduino und Gateway sind per USB verbunden und kommunizieren über eine virtuelle serielle Schnittstelle.

Das Ergebnis zeigt, dass eine Steuerung eines Arduinos per WebSocket problemlos möglich ist. Die eingehenden Nachrichten müssen auf dem Arduino geparkt werden. Dieses Parding ist einfacher, wenn das verwendete Austauschformat schlank gehalten wird. Deshalb wurde anstelle von JSON das URL Format verwendet.

2. Einleitung

In diesem Kapitel wird die Motivation erläutert und genaue Fragesellung definiert. Dann folgt eine kleine Übersichtsarbeit mit dazugehöriger Literaturrecherche.

2.1. Fragestellung

Welche Möglichkeiten gibt es, einen Arduino Uno via Websocket zu steuern?

Während einer explorativen Online-Suche wurden einzelne Lösungen gefunden. Eine systematische Zusammenstellung der Möglichkeiten fehlt jedoch.

2.2. Motivation

Die Motivation für die vorliegende Arbeit ist die Beantwortung der nachfolgenden Fragestellung. Weiter soll der Artikel interessierten Lesern als Einstiegslektüre dienen.

2.3. Literatur-Review

Zum Thema existiert diverse Fachliteratur unter anderem von Erik Bartmann^{[2][3][4]}.

2.3.1. Arduino mit integriertem WLAN

Der Arduino Uno hat keine eingebaute WLAN Schnittstelle. Es gibt jedoch andere Arduino Modelle mit integriertem WLAN, wie z.B. der Arduino MKR1000.

2.3.2. WLAN Erweiterung

Mehrere Autoren berichten^{[5][6]}, wie der Arduino mit dem dem WLAN Modul ESP8266 erweitert werden kann.

2.3.3. Serial Gateway

Eine weitere Möglichkeit, ist behelfsweise einen PC als Serial Gateway einzusetzen. Mangels kurzfristig verfügbarer Hardware wollen wir diese Option verfolgen.

3. Experimenteller Teil

3.1. Informationsquellen

Als Informationsquellen sind die Datenblätter zur jeweiligen Hardware sowie die Manuals zu den eingesetzten Softwarekomponenten zu nennen.

3.2. Prinzipskizze

3.3. Hardware

Verwendet wurde das Lernset Nr. 8 von Funduino^[1]. Darin enthalten ist ein Funduino Uno. Weiter benötigen wir den Temperatursensor TMP36 und den Fotowiderstand.

3.3.1. Anschluss der Sensoren

3.4. Software

3.4.1. Entwicklungsumgebung

Zur Entwicklung wurde folgende Software eingesetzt.

- Visual Studio Code^[7] mit der Erweiterung C/C++ IntelliSense^[8]
- Arduino CLI^[9]
- Git for Windows^[10] und TortoiseGit^[11]

Nicht verwendet wurde die Arduino IDE. Windows verwendet den Standardtreiber *usbser.sys* für den virtuellen COM Port.

3.4.2. Node Libraries

Weiter wurde folgende NPM Packages eingesetzt:

- WebSockets^[12]
- Express^[13]
- Chart.js^[14]
- SerialPort^[15]

3.4.3. Arduino Libraries

Weiter wurde folgende Arduino Libraries eingesetzt:

- Arduino Library (Arduino.h)^{[16][17][18]}
- AVR Libc^[19]

3.4.4. Arduino Sketch

Zunächst müssen wir klären, in welcher Programmiersprache die Arduino Sketches geschrieben werden. Nachdem man sich die Build-Umgebung genauer unter die Lupe genommen hat, wird klar, dass keine eigene Arduino-Sprache existiert^[20]. Im Hintergrund wird aus dem Sketch eine C++ Datei erstellt und mit *avr-g++* kompiliert.

Die Problematik der Heap-Fragmentierung wird von mehreren Autoren aufgeworfen und diskutiert^{[21][22]}. Matt ist der Meinung, dass man deshalb auf die String Klasse in der Arduino Library gänzlich verzichten soll^[23]. In der Konsequenz müsste man die Stringfunktion aus der Standard C Library^[19] verwenden und in C programmieren. Ich sehe dies nicht ganz so eng und setze die Arduino String Klasse trotzdem, jedoch mit Zurückhaltung ein. Ich befolge Matt's Rat, die Variablen by Reference zu übergeben^[23].

Der Quellcode befindet sich im Anhang A.1.

3.4.5. Serial Gateway

Der Quellcode befindet sich im Anhang A.2.

3.4.6. WebSocket Server

Der Quellcode befindet sich im Anhang A.3.

3.4.7. Web GUI

Der Quellcode befindet sich im Anhang A.4.

4. Resultate

Es hat sich gezeigt, dass ein Seriell-zu-Websocket-Gatway unter Node.js einfach zu implementieren ist. Über diesen Umweg kann der Arduino Uno ans Internet angebunden werden.

5. Diskussion

6. Zusammenfassung

Statt des Arduino Uno könnte ein Arduino MKR1000 verwendet werden. Dieser könnte kann auch an die Arduino Cloud angebunden werden. Ein weitere Option ist die Beschaffung einer WLAN Erweiterung wie das Modul ESP8266.

7. Danksagung

Ich danke den Lernenden der Klasse BINF2017A für die Zusammenarbeit.

8. Interessenskonflikte

Das Projekt wurde im Rahmen des Berufsfachschulunterrichts durchgeführt und erhielt keine externe Finanzierung. Demnach bestehen keien Interessenkonflikte.

9. Quellenverzeichnis

Bücher

- [2] Erik Bartmann. *Mit Arduino die elektronische Welt entdecken*. 3. Aufl. Bombini-Verlag, 2017. ISBN: 978-3-946496-00-7.
- [3] Erik Bartmann. *Das ESP8266-Praxisbuch: Mit NodeMCU und ESPlorer*. Elektor Verlag, 2016. ISBN: 978-3-89576-321-2.
- [4] Erik Bartmann. *Das ESP32-Praxisbuch: Programmieren mit der Arduino-IDE*. Elektor Verlag, 2018. ISBN: 978-3-89576-333-5.

Web

- [1] *Lernset Nr.8 mit UNO Controller - Kit für Arduino*. Funduino. URL: https://www.funduinoshop.com/epages/78096195.sf/de_DE/?ObjectPath=/Shops/78096195/Products/01-U8.
- [5] Igor Khanenko Andrew Shvayka Igor Kulikov. *Temperature Dashboard Using Arduino UNO, ESP8266 And MQTT*. URL: <https://www.hackster.io/thingsboard/temperature-dashboard-using-arduino-uno-esp8266-and-mqtt-5e26eb>.
- [6] *WebSocket communication with an ESP8266 or Arduino in Python. Test with the ws4py library on Raspberry Pi*. URL: <https://diyprojects.io/websocket-communication-esp8266-arduino-python-test-ws4py-library-raspberry-pi/#.Xq6bKagzaUk>.

- [20] *C++ vs. The Arduino Language?* URL: <https://arduino.stackexchange.com/questions/816/c-vs-the-arduino-language>.
- [21] *What is Heap Fragmentation?* 6. Nov. 2018. URL: <https://cpp4arduino.com/2018/11/06/what-is-heap-fragmentation.html>.
- [22] Colin Walls. *Dynamic Memory Allocation and Fragmentation in C and C++*. 6. Nov. 2018. URL: <https://www.design-reuse.com/articles/25090/dynamic-memory-allocation-fragmentation-c.html>.
- [23] *The Evils of Arduino Strings*. 4. Feb. 2016. URL: <https://majenko.co.uk/blog/evils-arduino-strings>.
- [26] *LaTeX: Inhaltsverzeichnis anlegen - so geht's*. URL: <https://www.heise.de/tipps-tricks/LaTeX-Inhaltsverzeichnis-anlegen-so-geht-s-4401672.html>.
- [27] *LaTeX: Literaturverzeichnis erstellen - so klappt's*. URL: https://www.heise.de/tipps-tricks/LaTeX-Literaturverzeichnis-erstellen-so-klappt-s-4401420.html#%C3%9Cberschrift_1.

Manuals

- [16] *Sprach-Referenz*. URL: <https://www.arduino.cc/reference/de/>.
- [17] *Arduino Befehlsübersicht*. URL: <https://www.arduinoforum.de/code-referenz>.
- [18] *Arduino Programming Cheat Sheet*. URL: <https://github.com/liffiton/Arduino-Cheat-Sheet>.
- [19] *AVR Libc*. URL: <https://www.nongnu.org/avr-libc/>.

Software

- [7] *Visual Studio Code*. Microsoft. URL: <https://code.visualstudio.com/>.
- [8] *C/C++ IntelliSense, debugging, and code browsing*. Microsoft. URL: <https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools>.
- [9] *Arduino CLI*. Arduino. URL: <https://github.com/arduino/arduino-cli>.
- [10] *Git for Windows*. URL: <https://gitforwindows.org>.
- [11] *TortoiseGit*. URL: <https://tortoisegit.org>.
- [12] *Simple to use, blazing fast and thoroughly tested WebSocket client and server for Node.js*. URL: <https://github.com/websockets/ws>.
- [13] *Express Schnelles, offenes, unkompliziertes Web-Framework für Node.js*. URL: <https://expressjs.com>.
- [14] *Simple yet flexible JavaScript charting for designers & developers*. URL: <https://www.chartjs.org>.

- [15] *Node SerialPort*. URL: <https://serialport.io>.
- [24] *TeX Live*. TeX User Group. URL: <https://www.tug.org/texlive/>.
- [25] James Yu. *LaTeX Workshop*. URL: <https://github.com/James-Yu/LaTeX-Workshop>.

A. Quellcode

A.1. Arduino Sketch

```
1  /*
2   This sketch periodically reads all analog inputs and logs the values in
3   JSON format to the serial line.
4   In addition it will listen to the serial line and process any incoming
5   messages.
6   To compile and upload this sketch from the arduino folder execute '
7   arduino-cli compile genericReadWrite -u'
8   The options -b and -p will be taken automatically from sketch.json
9   Upload is not possible while server.js is running because it uses the
10  serial line also.
11
12  The digital pins are named from 0 to 13. DIO 13 and the built-in LED are
13  connected.
14  Digital pins 0 and 1 should be avoided as they are internally conneted
15  to the serial line.
16  The analog pins A0 to A5 habe the IDs 14 to 19.
17  Hence, in order to query all IO states (digital and analog) we can make
18  a loop from 0 to 19
19  */
20
21 #include <Arduino.h>
22 #include "status.h"
23 #include "control.h"
24
25 // global variables
26 unsigned long previousMillis = 0;    // count loops
27 const long interval = 1000;         // interval at which to send IO status
28                                     // updates (milliseconds)
29
30 // initialisation
31 void setup() {
32     // init serial line with 9600 baud
33     Serial.begin(9600);
34     // init all digital pins
35     for (int i = 0; i <= 13; i++) {
36         pinMode(i, OUTPUT);
37         digitalWrite(i, LOW);
38     }
39 }
40
41 // main
42 void loop() {
43     // ckeck if we have incoming message, i.e. check if something is in the
44     // serial buffer
45     if (Serial.available() > 0) {
46         // receive messages on serial line, parse and handle the messages
47         processSerialInput();
48         // send updated IO states to host
49         Serial.println(readPins());
50     }
51 }
```

```

40 } else {
41     // get the milliseconds since start
42     unsigned long currentMillis = millis();
43     // do only something if interval time has passed
44     if (currentMillis - previousMillis >= interval) {
45         // save the last time we sent the IO states
46         previousMillis = currentMillis;
47         // send IO states to host
48         Serial.println(readPins());
49     }
50 }
51 }

```

Listing 1: ../arduino/genericReadWrite/genericReadWrite.ino

A.1.1. Funktionen zur Statusausgabe

```

1 #include <Arduino.h>
2 #include "status.h"
3
4 // get status of all pin ans make JSON object , e.g. {"D0":1, ... , "A0
   ":1023}
5 String readPins() {
6     String ioStates = "{";
7     for (int i = 0; i <= 19; i++) {
8         ioStates += (i <= 13) ? "\"D\"" : "\"A\"";
9         ioStates += (i <= 13) ? i : i-14;
10        ioStates += "\"\":";
11        ioStates += (i <= 13) ? digitalRead(i) : analogRead(i);
12        ioStates += (i == 19) ? "\"" : ",";
13    }
14    ioStates += "}";
15    return ioStates;
16 }

```

Listing 2: ../arduino/genericReadWrite/status.cpp

A.1.2. Funktionen zur Steuerung

```

1 #include <Arduino.h>
2 #include "control.h"
3
4 // process incoming messages
5 void processSerialInput() {
6     String incomingMessage;           // e.g. 0=1&2=0&3=1
7     String keyValuePair;              // splitted at &
8     incomingMessage = Serial.readString(); // read the whole String in the
        serial buffer
9     while (true) {
10        // get the next key-value pair to process

```

```

11     int pos = incomingMessage.indexOf('&');
12     if (pos != -1) {
13         // '&' is present, we have multiple key-value pairs
14         keyValuePair = incomingMessage.substring(0,pos); // get the first
            key-value pair
15         processCommand(keyValuePair); // process it
16         incomingMessage = incomingMessage.substring(pos+1); // and remove it
            from incomingMessage
17     } else {
18         // just one command or last command
19         processCommand(incomingMessage); // process it
20         break;
21     }
22 }
23 }
24
25 // process key-value pair
26 void processCommand(String &command) { // &command makes receiving the
    String object by reference, hence avoid copies and save resources
27     int pos = command.indexOf('=');
28     if (command.length() > 2 && pos > 0 ) { // min 3 characters with '='
        present at minimum second position
29         int pin = command.substring(0,pos).toInt(); // characters before '='
30         int state = command.substring(pos+1).toInt(); // characters after '='
31         if (pin >= 0 && pin <= 19 && ( state == 0 || state == 1 )) {
32             // Serial.println("Setting input " + key + " to " + value);
33             digitalWrite(pin, state);
34         }
35     }
36 }

```

Listing 3: ../arduino/genericReadWrite/control.cpp

A.2. Serial Gateway

```
1 // serial gateway
2 const url = 'ws://websocksrv.herokuapp.com'
3 const WebSocket = require('ws');
4 const ws = new WebSocket(url);
5 const SerialPort = require('serialport')
6 const Readline = require('@serialport/parser-readline')
7 const serial = new SerialPort('COM3', { baudRate: 9600 })
8 const parser = new Readline()
9 // forwarding messages from serial to websocket
10 serial.pipe(parser)
11 parser.on('data', function (line) {
12   try {
13     var obj = JSON.parse(line)
14     ws.send(line);
15   } catch(err) {
16     // console.log(err)
17     console.log('gateway_parse_error: skipping malformed data')
18   }
19 })
20 // forwarding messages from websocket to serial
21 ws.on('open', function() {
22   console.log('Connected to ' + url)
23 })
24 ws.on('message', function(message) {
25   console.log('gateway_forwarding_command_to_Arduino: ' + message)
26   serial.write(message)
27 })
```

Listing 4: ../gateway/gateway.js

A.3. WebSocket Server

```
1 // WebSocket server
2 const port = process.env.PORT || 81
3 const WebSocket = require('ws');
4 const wss = new WebSocket.Server({ port: port });
5 wss.on('connection', function connection(ws) {
6   ws.on('message', function incoming(data) {
7     wss.clients.forEach(function each(client) {
8       if (client !== ws && client.readyState === WebSocket.OPEN) {
9         client.send(data);
10      }
11    });
12  });
13 });
```

Listing 5: ../websocketserver/websocketserver.js

A.4. Web Client

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Arduino IoT Demo App</title>
5   <link href="https://unpkg.com/material-components-web@v4.0.0/dist/
6     material-components-web.min.css" rel="stylesheet">
7   <script src="https://unpkg.com/material-components-web@v4.0.0/dist/
8     material-components-web.min.js"></script>
9   <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
10 </head>
11 <body onload="init()">
12   <div class="mdc-layout-grid">
13     <h2>Configure WebSocket Server</h2>
14     <input id="wsurl" type="text" size="50" value="ws://websocksrv.
15       herokuapp.com">
16     <input type="button" value="configure" onclick="init()">
17     <p id="help"></p>
18     <h2>Switch Arduino Digital Outputs</h2>
19     <select id="pin">
20       <option value=all>all</option>
21     </select>&nbsp;
22     <select id="state">
23       <option value=0>low</option>
24       <option value=1>high</option>
25     </select>&nbsp;
26     <input type="button" value="send" onclick="sendCommand()">
27     <div id="container" style="width:_80%;">
28       <h2>Temperatur</h2>
29       <canvas id="myChart1"></canvas>
30     </div>
31     <div id="container" style="width:_80%;">
32       <br>
33       <h2>Analogue Inputs</h2>
34       <canvas id="myChart2"></canvas>
35     </div>
36     <div id="container" style="width:_80%;">
37       <br>
38       <h2>Digital Inputs</h2>
39       <canvas id="myChart3"></canvas>
40     </div>
41   </div>
42 </body>
43 <script>
44   // const url = 'ws://localhost:81'
45   var url = null
46   var ws = null
47   var temperatures = []
```

```

47 var timestamps = []
48 var log = false
49
50 function init(){
51     wsurl = document.getElementById("wsurl").value
52     createWebsocket(wsurl)
53
54     var x = document.getElementById("pin");
55     for (i=0; i<=13; i++) {
56         var option = document.createElement("option");
57         option.text = "Digital_Output_" + i;
58         option.value = i;
59         x.add(option);
60     }
61     document.getElementById("help").innerHTML =
62     'This_page_will_open_a_WebSocket_connection_to_' + wsurl + '._</br>' +
63     'Through_the_socket_we_will_receive_JSON_objects_with_status_information'
64     '._This_page_displays_the_received_status_information_with_charts.</br>'
65     'Also,_we_can_send_commands_to_Arduino_through_the_socket,_e.g._switch'
66     'an_output._</br>' +
67     'In_Chrome_press_F12_for_further_details.'
68 }
69 function sendCommand(){
70     var pin = document.getElementById('pin').value
71     var state = document.getElementById('state').value
72     var command = '';
73     if (pin == 'all') {
74         for (i=0; i <= 13; i++) {
75             command+= i + "==" + state
76             command+= ( i == 13 ) ? '' : '&'
77         }
78     } else {
79         command = pin + "==" + state
80     }
81     console.log(command)
82     ws.send(command)
83 }
84 function calculateTemperature(adcValue) {
85     return adcValue/1023*5*100 - 50
86 }
87 function createTimestamp() {
88     return new Date().toISOString().slice(0,19).replace("T", "_")
89 }
90
91 function processIncomingMessages(event) {
92     try {
93         var ioStates = JSON.parse(event.data)
94         if (log) {
95             console.log(ioStates);

```



```

96     }
97
98     // update temperature chart
99     temperatures.push(calculateTemperature(ioStates.A0))
100    timestamps.push(createTimestamp())
101    updateChart(chart1, timestamps, temperatures)
102
103    // update IO charts
104    var keys = Object.keys(ioStates)
105    var values = Object.values(ioStates)
106    updateChart(chart3, keys.slice(0,14), values.slice(0,14)) // digital
107    updateChart(chart2, keys.slice(14,20), values.slice(14,20)) // analog
108    states
109    } catch(err) {
110        console.log('skipping_invalid_JSON_package_' + event.data + '"_from
111        _' + event.origin )
112        // console.log(err); // error in the above string (in this case, yes
113        )!
114    }
115    }
116
117    function createWebsocket(url) {
118        temperatures = []
119        timestamps = []
120        ws = new WebSocket(url);
121        ws.onopen = function() {
122            console.log('WebSocket_connection_to_' + url + '_established');
123            console.log('type="log=true"_to_log_incoming_JSON_objects');
124            console.log('type="ws.send("13=1")"_to_switch_the_build-in_LED_on')
125            ;
126        };
127        // handle messages from server
128        ws.onmessage = processIncomingMessages
129    }
130
131    function createChart(ctx, type, label) {
132        var chart = new Chart(ctx, {
133            // The type of chart we want to create
134            type: type,
135            // The data for our dataset
136            data: {
137                labels: [],
138                datasets: [{
139                    label: label,
140                    backgroundColor: 'rgb(255,99,132)',
141                    borderColor: 'rgb(255,99,132)',
142                    data: []
143                }]
144            },
145        });
146        return chart

```

```

143 }
144 function updateChart(chart, xValues, yValues) {
145     if (xValues && yValues) {
146         chart.data.labels = xValues
147         chart.data.datasets[0].data = yValues
148         chart.update();
149     }
150 }
151
152 var ctx1 = document.getElementById('myChart1').getContext('2d');
153 var ctx2 = document.getElementById('myChart2').getContext('2d');
154 var ctx3 = document.getElementById('myChart3').getContext('2d');
155
156 var chart1 = createChart(ctx1, 'line', 'Temperatursensor_Lan_A0')
157 chart1.options.scales.yAxes[0].ticks.suggestedMin = -50
158 chart1.options.scales.yAxes[0].ticks.suggestedMax = 150
159 chart1.data.datasets[0].fill = false
160
161 var chart2 = createChart(ctx2, 'bar', 'Analog_Inputs')
162 chart2.options.scales.yAxes[0].ticks.min = 0
163 chart2.options.scales.yAxes[0].ticks.max = 1024
164 chart2.options.scales.yAxes[0].ticks.stepSize = 128
165
166 var chart3 = createChart(ctx3, 'bar', 'Digital_Inputs')
167 chart3.options.scales.yAxes[0].ticks.min = 0
168 chart3.options.scales.yAxes[0].ticks.max = 1
169 chart3.options.scales.yAxes[0].ticks.stepSize = 1
170
171
172 </script>
173 </html>

```

Listing 6: ../client/index.html

B. Erstellung dieses Dokuments

Dieses Dokument wurde mit LaTeX erstellt. Der Quellcode befindet sich in der Datei `_doc/paper.tex`. Die Quellensammlung befindet sich in der Datei `_doc/quellen.tex` kkk

B.1. LaTeX Umgebung

Dazu wurde TeX Live [24] installiert. Dieses Paket stellt zahlreiche Tools zur Verfügung, darunter `pdflatex`

B.2. LaTeX Editor

Als Editor wurde Visual Studio Code [7] mit der Erweiterung LaTeX Workshop [25] verwendet.

Das automatische Speichern kann in Visual Studio Code unter `File -> Auto save` aktiviert werden. Mit `Ctrl+Alt+V` wird die Vorschau des generierten PDFs aktiviert. Mit `Alt+Z` kann der automatische Zeilenumbruch eingeschalten werden.

Vorteile gegenüber Microsoft Word sind die automatische Vervollständigung beim Schreiben, sowie die einfache Einbindung von Quellcode Listings wie in Anhang A.

Weitere Informationen zum Erstellen des Inhaltsverzeichnis und des Literaturverzeichnis finden sich unter [26] und [27].

B.3. LaTeX Pakete

Wie in `_doc/setup.tex` zu sehen, werden folgende Pakete verwendet.

- `babel` – Multilingual support for Plain TeX or LaTeX
- `xcolor` – Driver-independent color extensions for LaTeX and pdfLaTeX
- `listings` – Typeset source code listings using LaTeX
- `biblatex` – Sophisticated Bibliographies in LaTeX
- `csquotes` – Context sensitive quotation facilities

Diese Pakete sind im Lieferumfang von TeX Live dabei und müssen nicht speziell installiert werden.