ICT Modul 121 – Steuerungsaufgaben bearbeiten

Steuerung eines Arduino Uno Mikrokontrollers via WebSocket

Samuel Hess
samuel.hess@tfbern.ch
Technische Fachschule Bern

10. Mai 2020

Inhaltsverzeichnis

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 geparst werden. Dieses Parding ist einfacher, wenn das verwendete Austauschformt 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 diesen.

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 jweiligen Hardware sowie die Manuals zu den einsesetzten 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 Temparatursensor 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]
- AVR Libc[17]

3.4.4. Arduino Sketch

Der Quellcode befindet sich im Kapitel A.1.

3.4.5. Serail Gateway

Der Quellcode befindet sich im Kapitel A.2.

3.4.6. WebSocket Server

Der Quellcode befindet sich im Kapitel A.3.

3.4.7. Web GUI

Der Quellcode befindet sich im Kapitel 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 Clound 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 Beruffachschulunterrichts durchgeführt und erhielt keine externde Finanzierung. Demnach bestehen keien Interessenkonflikte.

9. Quellen

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.

Online

[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] *LaTeX: Inhaltsverzeichnis anlegen - so geht's*. URL: https://www.heise.de/tipps-tricks/LaTeX-Inhaltsverzeichnis-anlegen-so-geht-s-4401672.html.

[21] *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] *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 command line interface*. 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.

[18] *TeX Live*. TeX User Group. URL: https://www.tug.org/texlive/.

[19] James Yu. *LaTeX Workshop*. URL: https://github.com/James-Yu/LaTeX-Workshop.

A. Quellcode

A.1. Arduino Sketch

```
/*
  This sketch periodically reads all analog inputs and logs the values in JSON format to the serial line.
  In addition it will listen to the serial line and process any incoming messages.
  To compile and upload this sketch from the arduino folder execute 'arduino-cli compile genericReadWrite -u'
  The options -b and -p will be taken automatically from sketch.json
  Upload is not possible while server.js is running because it uses the serial line also.

  The digital pins are named from 0 to 13. DIO 13 and the built-in LED are connected.
  Digital pins 0 and 1 should be avoided as they are internally conneted to the serial line.
  The analog pins A0 to A5 habe the IDs 14 to 19.
  Hence, in order to query all IO states (digital and analog) we can make a loop from 0 to 19
*/
#include <Arduino.h>
#include "status.h"
#include "control.h"

// global variables
unsigned long previousMillis = 0;    // count loops
const long interval = 1000;          // interval at which to send IO status updates (milliseconds)

// initialisation
void setup(){
  // init serial line with 9600 baud
  Serial.begin(9600);
  // init all digital pins
  for (int i = 0; i <= 13; i++) {
    pinMode(i, OUTPUT);
    digitalWrite(i, LOW);
  }
}

// main
void loop() {
  // ckeck if we have incoming message, i.e. check if something is in the serial buffer
  if (Serial.available() > 0) {
    // receive messages on serial line, parse and handle the messages
    processSerialInput();
    // send updated IO states to host
    Serial.println(readPins());
```

```
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

```
// serial gateway
const url = 'ws://websocksrv.herokuapp.com'
const WebSocket = require('ws');
const ws = new WebSocket(url);
const SerialPort = require('serialport')
const Readline = require('@serialport/parser-readline')
const serial = new SerialPort('COM3', { baudRate: 9600 })
const parser = new Readline()
// forwarding messages from serial to websocket
serial.pipe(parser)
parser.on('data', function (line) {
  try {
    var obj = JSON.parse(line)
    ws.send(line);
  } catch(err) {
    // console.log(err)
    console.log('gateway parse error: skipping malformed data')
  }
})
// forwarding messages from websocket to serial
ws.on('open', function() {
  console.log('Connected to ' + url)
})
ws.on('message', function(message) {
  console.log('gateway forwarding command to Arduino: ' + message)
  serial.write(message)
})
```

Listing 4: ../gateway/gateway.js

A.3. WebSocket Server

```
// WebSocket server
const port = process.env.PORT || 81
const WebSocket = require('ws');
const wss = new WebSocket.Server({ port: port });
wss.on('connection', function connection(ws) {
  ws.on('message', function incoming(data) {
    wss.clients.forEach(function each(client) {
      if (client !== ws && client.readyState === WebSocket.OPEN) {
        client.send(data);
      }
    });
  });
});
```

Listing 5: ../websocketserver/websocketserver.js

A.4. Web Client

```
<!DOCTYPE html>
<html>
<head>
  <title>Arduino IoT Demo App</title>
  <link href="https://unpkg.com/material-components-web@v4.0.0/dist/material-components-web.min.css" rel="stylesheet">
  <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
  <script src="https://unpkg.com/material-components-web@v4.0.0/dist/material-components-web.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
</head>
<body onload="init()">
  <div class="mdc-layout-grid">
    <h2>Configure WebSocket Server</h2>
    <input id="wsurl" type="text" size="50" value="ws://websocksrv.herokuapp.com">
    <input type="button" value="configure" onclick="init()">
    <p id="help"></p>
    <h2>Switch Arduino Digital Outputs</h2>
    <select id="pin">
      <option value=all>all</option>
    </select> 
    <select id="state">
      <option value=0>low</option>
      <option value=1>high</option>
    </select> 
    <input type="button" value="send" onclick="sendCommand()">
    <div id="container" style="width: 80%;">
      <h2>Temperatur</h2>
      <canvas id="myChart1"></canvas>
    </div>
    <div id="container" style="width: 80%;">
      <br>
      <h2>Analogue Inputs</h2>
      <canvas id="myChart2"></canvas>
    </div>
    <div id="container" style="width: 80%;">
      <br>
      <h2>Digital Inputs</h2>
      <canvas id="myChart3"></canvas>
    </div>
  </div>
</body>

<script>
// const url = 'ws://localhost:81'
var url = null
var ws = null
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
       . This page displays the received status information with charts.</br
       >' +
64   'Also, we can send commands to Arduino through the socket, e.g. switch
       an output. </br>' +
65   'In Chrome press F12 for further details.'
66 }
67 function sendCommand(){
68   var pin = document.getElementById('pin').value
69   var state = document.getElementById('state').value
70   var command = '';
71   if (pin == 'all') {
72     for (i=0; i <= 13; i++) {
73       command+= i + "=" + state
74       command+= ( i == 13 ) ? '' : '&'
75     }
76   } else {
77     command = pin + "=" + state
78   }
79   console.log(command)
80   ws.send(command)
81 }
82
83 function calculateTemperature(adcValue) {
84   return adcValue/1023*5*100 - 50
85 }
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
            states
107     updateChart(chart2, keys.slice(14,20), values.slice(14,20)) // analog
            states
108   } catch(err) {
109       console.log('skipping invalid JSON package "' + event.data + '" from
            ' + event.origin )
110       // console.log(err); // error in the above string (in this case, yes
            )!
111   }
112 }
113
114 function createWebsocket(url) {
115   temperatures = []
116   timestamps = []
117   ws = new WebSocket(url);
118   ws.onopen = function() {
119       console.log('WebSocket connection to ' + url + ' established');
120       console.log('type "log=true" to log incoming JSON objects');
121       console.log('type "ws.send("13=1")" to switch the build-in LED on')
              ;
122   };
123   // handle messages from server
124   ws.onmessage = processIncomingMessages
125 }
126
127 function createChart(ctx, type, label)  {
128   var chart = new Chart(ctx, {
129       // The type of chart we want to create
130       type: type,
131       // The data for our dataset
132       data: {
133           labels: [],
134           datasets: [{
135               label: label,
136               backgroundColor: 'rgb(255, 99, 132)',
137               borderColor: 'rgb(255, 99, 132)',
138               data: []
139           }]
140       },
141   });
142   return chart
```

```
}
function updateChart(chart, xValues, yValues) {
  if (xValues && yValues) {
    chart.data.labels = xValues
    chart.data.datasets[0].data = yValues
    chart.update();
  }
}

var ctx1 = document.getElementById('myChart1').getContext('2d');
var ctx2 = document.getElementById('myChart2').getContext('2d');
var ctx3 = document.getElementById('myChart3').getContext('2d');

var chart1 = createChart(ctx1, 'line', 'Temperatursensor an A0')
chart1.options.scales.yAxes[0].ticks.suggestedMin = -50
chart1.options.scales.yAxes[0].ticks.suggestedMax = 150
chart1.data.datasets[0].fill = false

var chart2 = createChart(ctx2, 'bar', 'Analog Inputs')
chart2.options.scales.yAxes[0].ticks.min = 0
chart2.options.scales.yAxes[0].ticks.max = 1024
chart2.options.scales.yAxes[0].ticks.stepSize = 128

var chart3 = createChart(ctx3, 'bar', 'Digital Inputs')
chart3.options.scales.yAxes[0].ticks.min = 0
chart3.options.scales.yAxes[0].ticks.max = 1
chart3.options.scales.yAxes[0].ticks.stepSize = 1


</script>
</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` klk

B.1. LaTeX Umgebung

Dazu wurde TeX Live[18] 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[19] 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[20] und[21].

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.