SPV Projekt

Timo Briddigkeit Johannes Selymes

28. Mai 2015, Hagenberg

Inhaltsverzeichnis

1 Einleitung

Grundlage für die heute gänigen Quadrocopter sind Fortschritte in der Elektronik und Sensorik, die auf dem Markt ab etwa 2000 verfügbar waren und ab 2004 in Serienmodellen erschienen: Leistungsfähige Mikrocontroller werten die Daten von Gyroskopen aus und können so Kippmomente – die höher und plötzlicher auftreten als bei Hubschraubern, da der Auftriebsschwerpunkt meist in der Rotorebene liegt – automatisch ausregeln.

Dabei kommen Gyroskopsensoren auf Piezo-Basis oder MEMS (microelectromechanical systems) zur Messung der Winkelgeschwindigkeit zum Einsatz, die von dem Prozessor benutzt werden, um durch Drehzahlregelung der Elektromotoren die Drehraten zu dämpfen, womit das Fluggerät steuerbar bleibt.

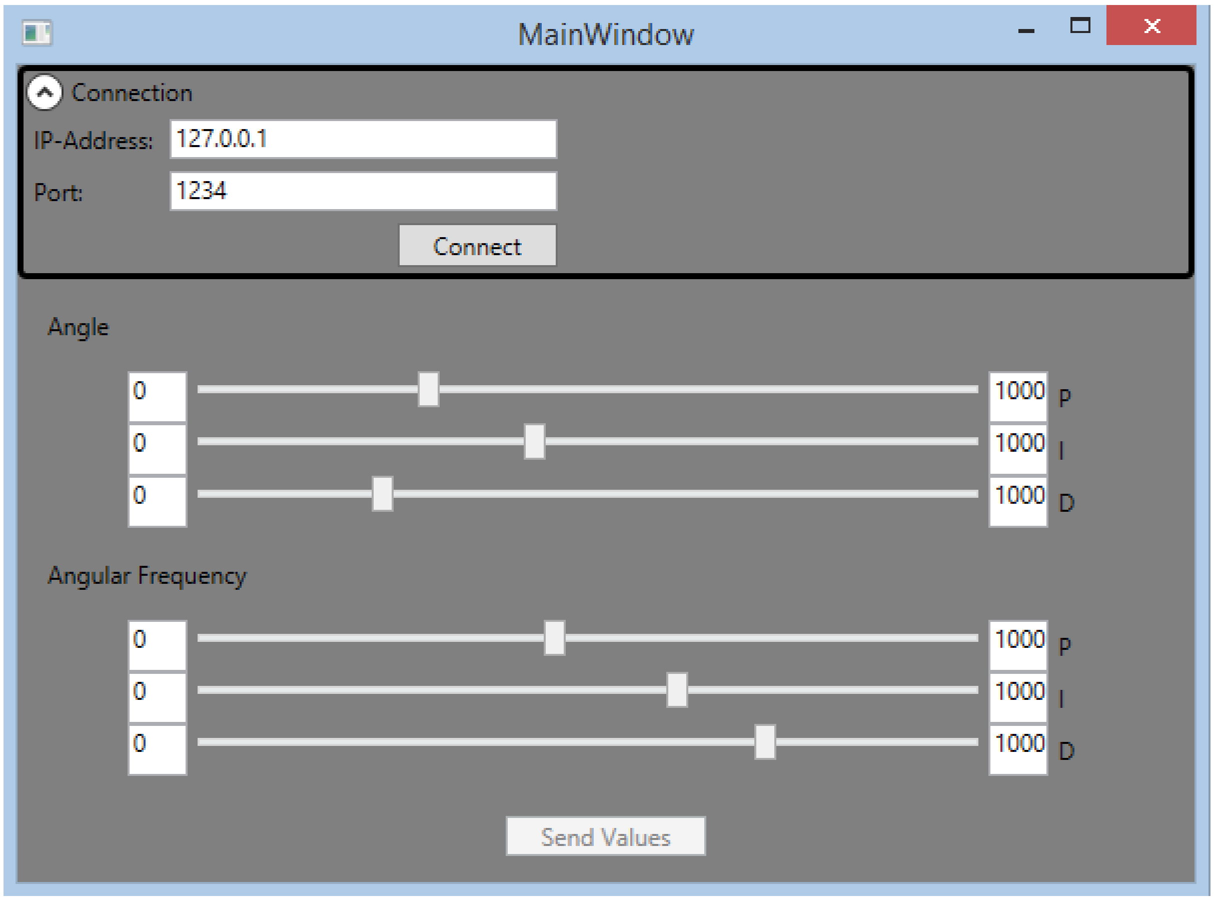
In diesem Projekt wird es ermöglicht, die Parameter der Regelung des Quadrocopters vom Computer aus einzustellen. Es wird dazu ein Server am Quadrocopter eingerichtet und eine WPF-Gui für das Windowssystem am Computer erstellt.

2 QuadServer

Auf dem Quadrocopter ist ein BeagleBone, auf welchem ein gentoo Linux läuft, installiert. Auf diesem Linux wird eine Applikation erstellt, welche als Server für das QuadcopterGui verwendet wird. Um die Applikation später in der Regelung einbinden zu können, wird der Server in einen Thread verpackt, der zyklisch Daten vom Socket abruft. Wenn Daten erhalten werden, werden diese geparst und die Parameter werden in das globale Parameter Singleton geschrieben. Dann wird noch ein Flag gesetzt, dass die Parameter geändert wurden, was dann später in der Regelung abgefragt wird.

3 QuadcopterGui

QuadcopterGui ist ein Prototyp für eine WPF-Anwendung, in der ein PID-Regler (**P**roportional, **I**ntegral, **D**ifferential) mit Winkel und Winkelgeschwindigkeit angesteuert werden kann.
Die Anwendung baut eine Verbindung zum QuadServer auf und schickt durch Betätigung des Send-Buttons PID-Regelwerte an den Quadserver, welche sich über sog. Slider einstellen lassen. Eine Textbox links und rechts vom Slider definiert den Minimal- bzw. Maximalwert des entsprechenden Regelwertes.
Die Übergabe der Minimal- bzw. Maximalwerte erfolgt über sog. Databinding an den Slider.



IP-Address: 127.0.0.1
Port: 1234

Angle: P 0 – 1000, I 0 – 1000, D 0 – 1000
Angular Frequency: P 0 – 1000, I 0 – 1000, D 0 – 1000

Abbildung 1: Screenshot der QuadcopterGui Anwendung

4 Netzwerkkommunikation

Die beiden Anwendungen kommunizieren über eine TCP/IP Verbindung. Dabei versendet die QuadcopterGui ihre Daten in einem JSON-Format, ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen, welche vom QuadServer geparst werden. Das JSON-Format ist notwendig, um das Endian-Problem zu umgehen, welches sich zwischen unterschiedlichen Computersystem stellen kann. Während heutige PC-Systeme in der Regel Little-Endian sind, sind ARM-Prozessoren, welche häufig im Embedded-Bereich anzutreffen sind, Bi-Endian Systeme. Die eigentliche TCP/IP Kommunikation erfolgt wie nachfolgend dargestellt drahtlos.

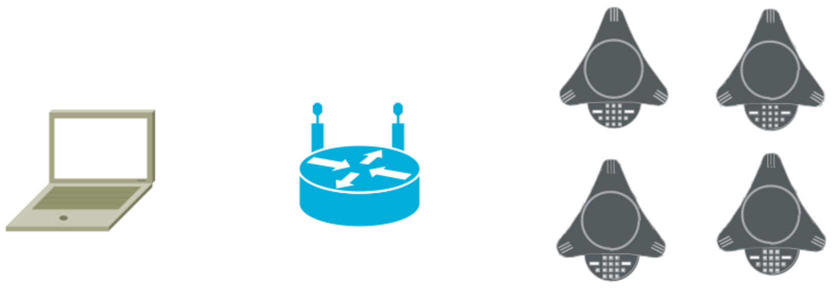
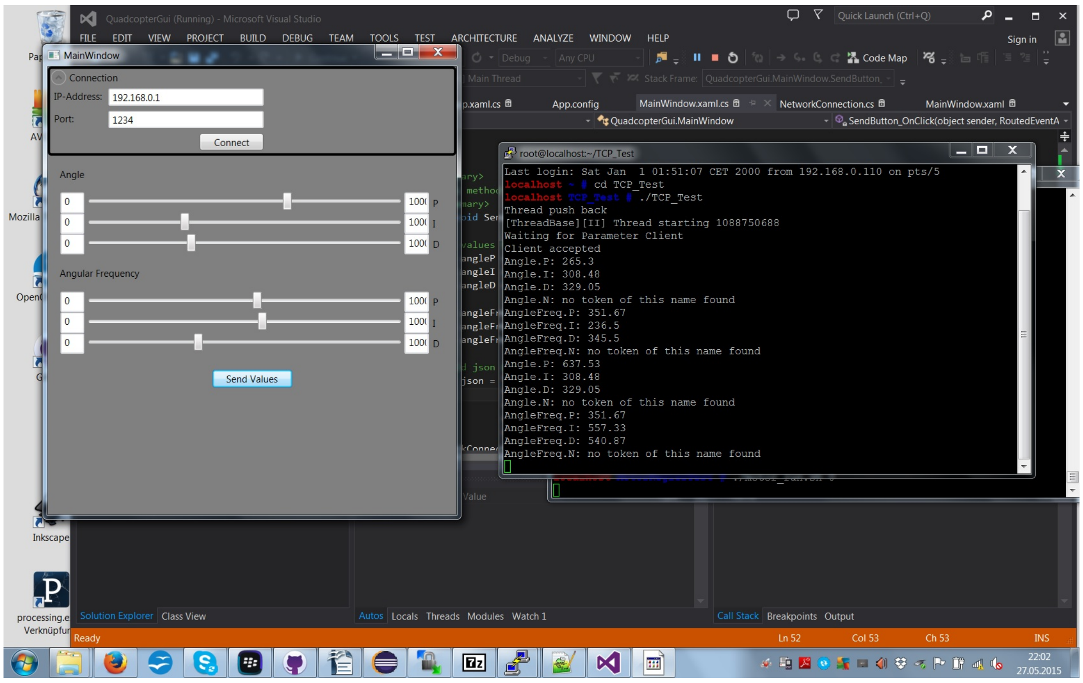


Abbildung 2: Netzwerkkommunikation

5 Testausgaben

Hier sieht man auf der linken Seite das GUI und auf der rechten Seite das Linux des BeagleBones mit SSH verbunden. Wie zu sehen ist, fehlt noch der N Parameter, welcher aber dank JSON noch leicht hinzugefügt werden kann.



```
Last login: Sat Jan  1 01:51:07 CET 2000 from 192.168.0.110 on pts/5
localhost ~ # cd TCP_Test
localhost TCP_Test # ./TCP_Test
Thread push back
[ThreadBase][II] Thread starting 1088750688
Waiting for Parameter Client
Client accepted
Angle.P: 265.3
Angle.I: 308.48
Angle.D: 329.05
Angle.N: no token of this name found
AngleFreq.P: 351.67
AngleFreq.I: 236.5
AngleFreq.D: 345.5
AngleFreq.N: no token of this name found
Angle.P: 637.53
Angle.I: 308.48
Angle.D: 329.05
Angle.N: no token of this name found
AngleFreq.P: 351.67
AngleFreq.I: 557.33
AngleFreq.D: 540.87
AngleFreq.N: no token of this name found
```

Abbildung 3: Screenshot der Anwendung

6 Code des Servers

Einige Dateien werden der Übersichtlichkeit halber nicht ins PDF mitaufgenommen.

```
/*
 * Copyright (c) 2004-2013 Sergey Lyubka <valenok@gmail.com>
 * Copyright (c) 2013 Cesanta Software Limited
 * All rights reserved
 *
 * This library is dual-licensed: you can redistribute it and/or
   modify
 * it under the terms of the GNU General Public License version 2
   as
 * published by the Free Software Foundation. For the terms of
   this
 * license, see <http: *www.gnu.org/licenses/>.
 *
 * You are free to use this library under the terms of the GNU
   General
 * Public License, but WITHOUT ANY WARRANTY; without even the
   implied
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
   PURPOSE.
 * See the GNU General Public License for more details.
 *
 * Alternatively, you can license this library under a commercial
 * license, as set out in <http://cesanta.com/products.html>.
 */

#ifndef FROZEN_HEADER_INCLUDED
#define FROZEN_HEADER_INCLUDED

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#include <stdarg.h>

enum json_type {
  JSON_TYPE_EOF     = 0,      /* End of parsed tokens marker */
  JSON_TYPE_STRING  = 1,
  JSON_TYPE_NUMBER  = 2,
  JSON_TYPE_OBJECT  = 3,
  JSON_TYPE_TRUE    = 4,
  JSON_TYPE_FALSE   = 5,
  JSON_TYPE_NULL    = 6,
  JSON_TYPE_ARRAY   = 7
};

struct json_token {
  const char *ptr;      /* Points to the beginning of the token */
  int len;              /* Token length */
  int num_desc;         /* For arrays and object, total number of
      descendants */
  enum json_type type;  /* Type of the token, possible values
      above */
};

/* Error codes */
#define JSON_STRING_INVALID           -1
#define JSON_STRING_INCOMPLETE        -2
#define JSON_TOKEN_ARRAY_TOO_SMALL    -3

int parse_json(const char *json_string, int json_string_length,
               struct json_token *tokens_array, int
                   size_of_tokens_array);
struct json_token *parse_json2(const char *json_string, int
    string_length);
struct json_token *find_json_token(struct json_token *toks, const
    char *path);

int json_emit_long(char *buf, int buf_len, long value);
int json_emit_double(char *buf, int buf_len, double value);
int json_emit_quoted_str(char *buf, int buf_len, const char *str,
    int len);
int json_emit_unquoted_str(char *buf, int buf_len, const char
    *str, int len);
int json_emit(char *buf, int buf_len, const char *fmt, ...);
int json_emit_va(char *buf, int buf_len, const char *fmt,
    va_list);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* FROZEN_HEADER_INCLUDED */
```

```
#ifndef VALUES_H_PWZ5324D
#define VALUES_H_PWZ5324D

#include <cstdint>
#include <thread>
#include <mutex>

template<typename T>
struct Triplet {
  T A;
  T B;
  T C;
};

typedef Triplet<double> DoubleTriplet;

struct PIDParam {
  double KP;
  double KI;
  double KD;
};

class Parameters {
public:

  static Parameters & getInstance(void) {
    static Parameters instance;
    return instance;
  }

  void SetParamsHaveChanged(bool val){
    mMutexParamsHaveChanged.lock();
    ParamsHaveChanged = val;
    mMutexParamsHaveChanged.unlock();
  }

  bool GetParamsHaveChanged(){
    bool val = false;
    mMutexParamsHaveChanged.lock();
    val = ParamsHaveChanged;
    mMutexParamsHaveChanged.unlock();
    return val;
  }

  // System Parameter
  int UMax;
  double DistancePropellor;
  double ThrustConst;
  double AirResistance;

  std::uint8_t MotorMin;
  std::uint8_t MotorMax;

  // Regel Parameter
  Triplet<PIDParam> PIDAngular;
  DoubleTriplet PIDAngularN;
  Triplet<PIDParam> PIDAngularRate;
  DoubleTriplet PIDAngularRateN;
  DoubleTriplet TauMax;

  // Steuerung
  DoubleTriplet SetPointAngular;
  double Thrust;
  double ThrustMin;
  double ThrustMax;
  double ThrustStep;

  // Sensoren
  DoubleTriplet GyroOffset;
  DoubleTriplet AccOffset;

private:
  Parameters() :
      ParamsHaveChanged(true), UMax(7000), DistancePropellor(0.22), ThrustConst(
          1.6155e-7), AirResistance(0.0000001), MotorMin(10), MotorMax(
          255), Thrust(1.0), // Starting Thrust, can be changed at runtime
      ThrustMin(0.1), ThrustMax(20), ThrustStep(0.02) {
    GyroOffset = {0.0, 0.0, 0.0};
    AccOffset = {0.0, 0.0, 0.0};
    TauMax = {20, 20, 5};
    SetPointAngular = {0.0, 0.0, 0.0};
    PIDAngular = { {0.0, 0.0, 0.0}, {0.0, 0.0, 0.0}, {0.0, 0.0,
       0.0}};
    PIDAngularRate =
    {
      {
        100,
        0,
        20
      },
      {
        100,
        0,
        20
      },
      {
        0,
        0,
        0
      }
    };

    PIDAngularN = {1.0, 1.0, 1.0};
    PIDAngularRateN = {
      1,
      1,
      1
    };
  }
  Parameters(Parameters &);
  Parameters(Parameters const &);
  Parameters(Parameters &&);
  Parameters(Parameters const &&);
  Parameters & operator= (Parameters &);
  Parameters & operator= (Parameters const &);

  // variable that shows that Parameters where changed
  bool ParamsHaveChanged;
  std::mutex mMutexParamsHaveChanged;
};

#endif /* end of include guard: VALUES_H_PWZ5324D */
```

```
/*
 * Parameters.cpp
 *
 *  Created on: 22.01.2015
 *      Author: Johannes Selymes
 */

```

```
#include "Parameters.h"
```

```

/**
 * @author Johannes Selymes
 * @date   2013-11-14
 */

#ifndef SOCEKTCOMMUNICATION_H_AP3HYVIW
#define SOCEKTCOMMUNICATION_H_AP3HYVIW

#include <cstdint>
#include <memory>
#include "TCP_TestConnection.h"

class ClientSocket;
class ServerSocket;

typedef std::unique_ptr<ClientSocket> UnqClientSocket;

class SocketBase
{
public:
    virtual ~SocketBase();

    // TODO documentation
    void close(void);
    bool is_blocking(void) const;
    bool set_blocking(bool blocking);
    bool is_valid(void) const; // socket fd != -1

protected:
    SocketBase();
    SocketBase(SocketBase &);
    SocketBase(SocketBase const &);
    SocketBase(SocketBase &&);
    SocketBase(SocketBase const &&);
    SocketBase & operator= (SocketBase &);
    SocketBase & operator= (SocketBase const &);

    int mSockfd;
    UnqLogger SOCK_LOG;
};

class ClientSocket : public SocketBase
{
public:
    TReadStatus write(std::uint8_t const * const data,
       std::uint32_t len) const;
    TReadStatus read(std::uint8_t * const data, std::uint32_t
       len) const;

private:
    ClientSocket(int fd);
    friend class ServerSocket;

    ClientSocket();
    ClientSocket(ClientSocket &);
    ClientSocket(ClientSocket const &);
    ClientSocket(ClientSocket &&);
    ClientSocket(ClientSocket const &&);
    ClientSocket & operator= (ClientSocket &);
    ClientSocket & operator= (ClientSocket const &);
};


class ServerSocket : public SocketBase
{
public:
    ServerSocket();

    /**
     * @brief bind to a specific port
     */
    bool bind(std::int16_t port) const;

    /**
     * @brief Listen for a number of clients
     *
     * @param  backlog  Indicates the maximum number of incoming
         connections
     *
     * @return True if the various operation were successful
     */
    bool listen(std::int32_t backlog) const;

    /**
     * @brief Accept new client connect and return the client
         socket
     *
     * @return ClientSocket or NULL on EWOULDBLOCK or failure
     */
    ClientSocket * accept() const;

    /**
     * @brief Tries to connect to a different Server
     *
     * @return ClientSocket
     */
    ClientSocket * connect(std::string address, std::int16_t
       port) const;

private:
    ServerSocket(ServerSocket &);
    ServerSocket(ServerSocket const &);
    ServerSocket(ServerSocket &&);
    ServerSocket(ServerSocket const &&);
    ServerSocket & operator= (ServerSocket &);
    ServerSocket & operator= (ServerSocket const &);
};

#endif /* end of include guard: SOCEKTCOMMUNICATION_H_AP3HYVIW */
```

```
#ifndef THREADBASE_H_QWZX93J7
#define THREADBASE_H_QWZX93J7

#include <pthread.h>
#include <vector>
#include <iostream>

class ThreadBase
{
public:
    ThreadBase ();
    virtual ~ThreadBase ();
    void SetPriority(int prio);

    bool Start(void);
//     void *Join(void);
    void Stop(void);

    bool IsRunning(void);
    virtual void CleanUp(void) = 0;
    virtual void Run(void) = 0;


protected:

private:
    static void * ThreadFunc(void *param);
    pthread_t *mThreadHdl;
};

extern std::vector<ThreadBase*> THREADS;

#endif /* end of include guard: THREADBASE_H_QWZX93J7 */
```

```

#include "TCP_TestConnection.h"
#include "ThreadBase.h"

#include <sched.h>
#include <signal.h>
#include <cassert>
#include <iostream>

static UnqLogger T_LOG {LOG.createLogger("ThreadBase")};

std::vector<ThreadBase*> THREADS;

void * ThreadBase::ThreadFunc(void *param)
{
    assert(param != 0);

    ThreadBase *pObj = static_cast<ThreadBase*>(param);

    T_LOG->info("Thread starting " +
       std::to_string(pthread_self()));

    // Base priority change later
    pObj->SetPriority(5);
    pObj->Run();
    T_LOG->info("Thread finished " +
       std::to_string(pthread_self()));
    pObj->CleanUp();

    return NULL;
}

ThreadBase::ThreadBase() :
    mThreadHdl(NULL)
{
}

ThreadBase::~ThreadBase(void)
{
}

void ThreadBase::SetPriority(int priority)
{
    struct sched_param sparam;
    sparam.sched_priority = priority;

    if (pthread_setschedparam(pthread_self(), SCHED_FIFO,
       &sparam) == -1) {
        T_LOG->error("Can't change pthread priority");
    }
}

bool ThreadBase::Start(void)
{
    if (mThreadHdl)
        return false;
    mThreadHdl = new pthread_t;
    if (mThreadHdl == 0){
      std::cout << "mThreadHdl == 0" << std::endl;
    }
    int ret = pthread_create(mThreadHdl,
                             NULL,
                             ThreadBase::ThreadFunc,
                             static_cast<void*>(this));

    if (ret) {
      T_LOG->error("can't create thread");
        delete mThreadHdl;
        mThreadHdl = NULL;
        return false;
    }
    else {
        std::cout << "Thread push back" << std::endl;
      THREADS.push_back(this);
        return true;
    }
    return true;
}

void ThreadBase::Stop(void)
{
    if (mThreadHdl != NULL) {
        pthread_cancel(*mThreadHdl);
        T_LOG->info("Thread stopped " +
           std::to_string(*mThreadHdl));
        delete mThreadHdl;
        mThreadHdl = NULL;
    }
    CleanUp();
}

bool ThreadBase::IsRunning(void)
{
    return mThreadHdl != NULL;
}
```

```
//#include "QuadcopterDaemon.h" comment in in real
#include "TCP_TestConnection.h"
```

```
#include "ThreadBase.h"
#include "SocketCommunication.h"
#include "JSONParser.h"
#include <string>

class ParameterUpdateThread : public ThreadBase
{
public:
    ParameterUpdateThread(int port);

    void Run(void);
    void CleanUp(void);
private:
    double GetDoubleToken(json_token* tokens, std::string name);
    int mPort;
    UnqLogger mLogger;
    ServerSocket mServer;
};
```

```
#include <unistd.h>
#include <iostream>

#include "Parameters.h"
#include "SocketCommunication.h"
#include "ParameterUpdateThread.h"

using namespace std;

ParameterUpdateThread::ParameterUpdateThread(int port) :
   ThreadBase(), mPort(port),
    mLogger(LOG.createLogger("ParameterUpdate"))
{
  mServer.set_blocking(true);
  mServer.bind(port);
  mServer.listen(1);
}

void PrintToken(json_token* token) {
  if (token != 0) {
    cout << atof(token->ptr) << endl;
  } else {
    cout << "no token of this name found" << endl;
  }
}

void PrintToken(json_token* tokens, string name) {
  cout << name << ": ";
  PrintToken(find_json_token(tokens, name.c_str()));
}
```

```

double ParameterUpdateThread::GetDoubleToken(json_token* tokens,
    string name) {
  json_token* token = find_json_token(tokens, name.c_str());
  if (token != 0 && token->ptr != 0) {
    return atof(token->ptr);
  } else {
    mLogger->info("can't parse double json");
    return 0.0;
  }
}

void ParameterUpdateThread::Run(void) {
  Parameters & params = Parameters::getInstance();
  while (1) {
    cout << "Waiting for Parameter Client" << endl;
    UnqClientSocket client { mServer.accept() };
    if (client == NULL) {
      mLogger->error("Could not accept client");
      continue;
    }
    cout << "Client accepted" << endl;

    uint32_t bufferSize = 200;
    unsigned char readBuffer[bufferSize];
    uint32_t numOfTokens = 50;
    struct json_token tokens[numOfTokens];

    while (1) {
      TReadStatus ret = client->read(readBuffer, bufferSize);
      if (ret == eSuccess) {
        int success = parse_json((char*) readBuffer, bufferSize,
            tokens,
             numOfTokens);
        if (success < 0) {
          mLogger->error("Cannot parse json string");
        } else {
          PrintToken(tokens, "Angle.P");
          PrintToken(tokens, "Angle.I");
          PrintToken(tokens, "Angle.D");
          PrintToken(tokens, "Angle.N");
          PrintToken(tokens, "AngleFreq.P");
          PrintToken(tokens, "AngleFreq.I");
          PrintToken(tokens, "AngleFreq.D");
          PrintToken(tokens, "AngleFreq.N");

          // update them in parameters, axis A and B the same
              values // TODO: maybe not thread-safe
          // Angle
```

```
            params.PIDAngular.A.KP = GetDoubleToken(tokens,
               "Angle.P");
            params.PIDAngular.A.KI = GetDoubleToken(tokens,
               "Angle.I");
            params.PIDAngular.A.KD = GetDoubleToken(tokens,
               "Angle.D");
            params.PIDAngularN.A = GetDoubleToken(tokens,
               "Angle.N");

            params.PIDAngular.B.KP = GetDoubleToken(tokens,
               "Angle.P");
            params.PIDAngular.B.KI = GetDoubleToken(tokens,
               "Angle.I");
            params.PIDAngular.B.KD = GetDoubleToken(tokens,
               "Angle.D");
            params.PIDAngularN.B = GetDoubleToken(tokens,
               "Angle.N");

            // Angular Rate
            params.PIDAngularRate.A.KP = GetDoubleToken(tokens,
               "AngleFreq.P");
            params.PIDAngularRate.A.KI = GetDoubleToken(tokens,
               "AngleFreq.I");
            params.PIDAngularRate.A.KD = GetDoubleToken(tokens,
               "AngleFreq.D");
            params.PIDAngularRateN.A = GetDoubleToken(tokens,
               "AngleFreq.N");

            params.PIDAngularRate.B.KP = GetDoubleToken(tokens,
               "AngleFreq.P");
            params.PIDAngularRate.B.KI = GetDoubleToken(tokens,
               "AngleFreq.I");
            params.PIDAngularRate.B.KD = GetDoubleToken(tokens,
               "AngleFreq.D");
            params.PIDAngularRateN.B = GetDoubleToken(tokens,
               "AngleFreq.N");

            // Parameters have changed, MainControl has to update
               parameters
            mLogger->info("Parameters have been updated in
               Parameter.h");
            params.SetParamsHaveChanged(true);
          }
        } else if (ret == eFailed) {
          mLogger->error("can't read data, closing connection");
          client->close();
          break;
        }
        // sleep a bit to not interrupt other tasks too much
```

```
      usleep(200000);
    }
    mLogger->info("Connection closed for ParameterUpdateThread");
  }
}

void ParameterUpdateThread::CleanUp(void) {
  mServer.close();
}
```

```
/*
 * TCP_TestConnection.h
 *
 *  Created on: 17.05.2015
 *      Author: Johannes Selymes
 */

#ifndef SRC_TCP_TESTCONNECTION_H_
#define SRC_TCP_TESTCONNECTION_H_

#include "Logging.h"

extern Logging LOG;

enum TReadStatus {
    eWouldBlock = -1,
    eFailed     =  0,
    eSuccess    =  1
};

#endif /* SRC_TCP_TESTCONNECTION_H_ */
```

```
/*
 * TCP_TestConnection.cpp
 *
 *  Created on: 17.05.2015
 *      Author: Johannes Selymes
 */

#include "ParameterUpdateThread.h"

using namespace std;

Logging LOG("TCP_TestConnection.log");

int main(int argc, char* argv[]) {
  int port = 1234;
  ParameterUpdateThread param_thread(port);
  param_thread.Start();

  while(true){}
}
```

7 Code der GUI

```
ï»¿using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Windows;

namespace QuadcopterGui
{
  /// <summary>
  ///   Class for the network communication. This class is able
     to connect to the server and send the values.
  /// </summary>
  internal class NetworkConnection
  {
    private readonly int _port;
    private readonly IPAddress _serverIpAddress;
    private TcpClient _client;
    private NetworkStream _stream;
    private bool _isConnected;
    private StreamWriter _streamWriter;

    /// <summary>
    ///   Constructor to initialise this class
    /// </summary>
    /// <param name="serverIpAddress">Serveraddress</param>
    /// <param name="port">TCP Port of the server</param>
    public NetworkConnection(IPAddress serverIpAddress, int port)
    {
      _serverIpAddress = serverIpAddress;
      _port = port;
    }

    /// <summary>
    ///   Connect to the server
    /// </summary>
    /// <returns>True if successfully connected</returns>
    public bool Connect()
    {
      _client = new TcpClient();
      if (_serverIpAddress != null)
      {
        try
        {
          _client.Connect(_serverIpAddress, _port);
          _stream = _client.GetStream();
          _streamWriter = new StreamWriter(_stream);
        }
        catch (Exception exception)
        {
          return false;
        }
        _isConnected = true;
        return true;
      }
      return false;
    }

    /// <summary>
    ///   Disconnect from server
    /// </summary>
    public void Disconnect()
    {
      _streamWriter.Close();
      _client.Close();
      _stream = null;
      _isConnected = false;
    }

    /// <summary>
    ///   Sends a string to the server
    /// </summary>
    /// <param name="data">JSON string to send</param>
    public void Send(string data)
    {
      if (_isConnected) // only send if we are connected
      {
        if (_streamWriter != null)
        {
          _streamWriter.Write(data);
          _streamWriter.Flush();
        }
      }
    }
  }
}
```

```
ï»¿using System;
using System.Globalization;
using System.Net;
using System.Threading;
using System.Windows;

namespace QuadcopterGui
{
  /// <summary>
  /// Interaktionslogik fÃ¼r MainWindow.xaml
  /// </summary>
  public partial class MainWindow : Window
  {
    private NetworkConnection _networkConnection;

    public MainWindow()
    {
        Thread.CurrentThread.CurrentCulture = new
           CultureInfo("en-US");
      InitializeComponent();
    }

    /// <summary>
    /// This is method is called if the connect button is pressed
    /// </summary>
    private void ConnectButton_OnClick(object sender,
       RoutedEventArgs e)
    {
      IPAddress ip = IPAddress.Parse(IpTextBox.Text);
      int port = Convert.ToInt32(PortTextBox.Text);
      _networkConnection = new NetworkConnection(ip, port);
      if (_networkConnection.Connect())
      {
        SendButton.IsEnabled = true;
      }
    }

    /// <summary>
    ///  This method is called if the send button is pressed
    /// </summary>
    private void SendButton_OnClick(object sender,
       RoutedEventArgs e)
    {
      // get values from sliders
      double angleP = Double.Parse(SliderAngleP.Value.ToString());
      double angleI = Double.Parse(SliderAngleI.Value.ToString());
      double angleD = Double.Parse(SliderAngleD.Value.ToString());

      double angleFreqP =
         Double.Parse(SliderAngleFreqP.Value.ToString());
      double angleFreqI =
         Double.Parse(SliderAngleFreqI.Value.ToString());
```

```
            double angleFreqD =
                Double.Parse(SliderAngleFreqD.Value.ToString());

            // build json string and round slider values to 2 of
                fractional digits
            string json = "{ \"Angle\" : { " +
                          "\"P\": " + Math.Round(angleP,2) + ", \"I\":
                              " + Math.Round(angleI, 2) + ", \"D\": " +
                              Math.Round(angleD, 2) + ", } " +
                          "\"AngleFreq\" : { " +
                          "\"P\": " + Math.Round(angleFreqP, 2) + ",
                              \"I\": " + Math.Round(angleFreqI, 2) + ",
                              \"D\": " + Math.Round(angleFreqD, 2) + ",
                              } " +
                          "}";
            _networkConnection.Send(json); // send to server
        }
    }
}
```

```
ï»¿<Window x:Class="QuadcopterGui.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525"
            Background="Gray">
    <StackPanel>
        <Expander BorderBrush="Black" BorderThickness="3"
            Header="Connection">
             <Grid>
                 <Grid.RowDefinitions>
                     <RowDefinition Height="Auto" />
                     <RowDefinition Height="Auto" />
                     <RowDefinition Height="*" />
                     <RowDefinition Height="28" />
                 </Grid.RowDefinitions>
                 <Grid.ColumnDefinitions>
                     <ColumnDefinition Width="Auto" />
                     <ColumnDefinition Width="200" />
                 </Grid.ColumnDefinitions>
                 <Label Grid.Row="0" Grid.Column="0"
                     Content="IP-Address:"/>
                 <Label Grid.Row="1" Grid.Column="0"
                     Content="Port:"/>
                 <TextBox Grid.Column="1" Grid.Row="0" Margin="3"
                     Name="IpTextBox">127.0.0.1</TextBox>
                 <TextBox Grid.Column="1" Grid.Row="1" Margin="3"
                     Name="PortTextBox">1234</TextBox>
                 <Button Grid.Column="1" Grid.Row="3"
                     HorizontalAlignment="Right"
            MinWidth="80" Margin="3" Content="Connect" Name="ConnectButton" Click="ConnectButton_OnClick" />
            </Grid>
        </Expander>
        <Label Margin="10">Angle</Label>
        <Grid HorizontalAlignment="Center">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <TextBox Grid.Row="0" Grid.Column="0" Width="30" Name="AnglePMinValueBox">0</TextBox>
            <Slider Name="SliderAngleP"  Grid.Row="0" Grid.Column="1" Width="400" Minimum="{Binding ElementName=AnglePMinValueBox, Path=Text}" Maximum="{Binding ElementName=AnglePMaxValueBox, Path=Text}" />
            <TextBox Grid.Row="0" Grid.Column="2"  Width="30" Name="AnglePMaxValueBox">1000</TextBox>
            <Label Grid.Row="0" Grid.Column="3" Content="P"/>
            <TextBox Grid.Row="1" Grid.Column="0"  Width="30" Name="AngleIMinValueBox">0</TextBox>
            <Slider Name="SliderAngleI" Grid.Row="1" Grid.Column="1" Width="400" Minimum="{Binding ElementName=AngleIMinValueBox, Path=Text}" Maximum="{Binding ElementName=AngleIMaxValueBox, Path=Text}"/>
            <TextBox Grid.Row="1" Grid.Column="2" Width="30" Name="AngleIMaxValueBox">1000</TextBox>
            <Label Grid.Row="1" Grid.Column="3" Content="I"/>
            <TextBox Grid.Row="2" Grid.Column="0" Width="30" Name="AngleDMinValueBox">0</TextBox>
            <Slider Name="SliderAngleD" Grid.Row="2" Grid.Column="1" Width="400" Minimum="{Binding ElementName=AngleDMinValueBox, Path=Text}" Maximum="{Binding ElementName=AngleDMaxValueBox, Path=Text}"/>
            <TextBox Grid.Row="2" Grid.Column="2" Width="30" Name="AngleDMaxValueBox">1000</TextBox>
            <Label Grid.Row="2" Grid.Column="3" Content="D"/>
        </Grid>
        <Label Margin="10">Angular Frequency</Label>
        <Grid HorizontalAlignment="Center">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <TextBox Grid.Row="0" Grid.Column="0"  Width="30"
               Name="AngleFreqPMinValueBox">0</TextBox>
            <Slider Name="SliderAngleFreqD" Grid.Row="0"
               Grid.Column="1" Width="400" Minimum="{Binding
               ElementName=AngleFreqPMinValueBox, Path=Text}"
               Maximum="{Binding
               ElementName=AngleFreqPMaxValueBox, Path=Text}" />
            <TextBox Grid.Row="0" Grid.Column="2"  Width="30"
               Name="AngleFreqPMaxValueBox">1000</TextBox>
            <Label Grid.Row="0" Grid.Column="3" Content="P"/>
            <TextBox Grid.Row="1" Grid.Column="0"  Width="30"
               Name="AngleFreqIMinValueBox">0</TextBox>
            <Slider Name="SliderAngleFreqI" Grid.Row="1"
               Grid.Column="1" Width="400" Minimum="{Binding
               ElementName=AngleFreqIMinValueBox, Path=Text}"
               Maximum="{Binding
               ElementName=AngleFreqIMaxValueBox, Path=Text}"/>
            <TextBox Grid.Row="1" Grid.Column="2" Width="30"
               Name="AngleFreqIMaxValueBox">1000</TextBox>
            <Label Grid.Row="1" Grid.Column="3" Content="I"/>
            <TextBox Grid.Row="2" Grid.Column="0" Width="30"
               Name="AngleFreqDMinValueBox">0</TextBox>
            <Slider Name="SliderAngleFreqP" Grid.Row="2"
               Grid.Column="1" Width="400" Minimum="{Binding
               ElementName=AngleFreqDMinValueBox, Path=Text}"
               Maximum="{Binding
               ElementName=AngleFreqDMaxValueBox, Path=Text}"/>
            <TextBox Grid.Row="2" Grid.Column="2" Width="30"
               Name="AngleFreqDMaxValueBox">1000</TextBox>
            <Label Grid.Row="2" Grid.Column="3" Content="D"/>
        </Grid>
        <Button Margin="20" Name="SendButton"
           Click="SendButton_OnClick" IsEnabled="False"
           Width="100">Send Values</Button>
    </StackPanel>
```

```
</Window>
```

Abbildungsverzeichnis