Student Research Project Summer Term 2011

Analyse und Implementierung einer Höhenregelung für den Quadrocopter

Hochschule Esslingen
University of Applied Sciences

Martin Ehrle / Markus Schüler

Tutor: Prof. Dr. Jörg Friedrich

Index

1 Introduction....3
1.1 Problem....3
1.2 Solution....4
2 Ultrasound....5
2.1 Uses of ultrasound....5
2.2 Creation and Registering....6
2.3 Ultrasound Distance Measurement....6
3 Sensor Analysis....8
3.1 Aspects....8
3.2 Sensor Types and Sensors....9
4 SRF10 General....11
4.1 General information about SRF10....11
4.2 Data sheet information....12
4.3 Connections....12
4.4 Register overview....13
4.5 Commands....13
4.6 Possible Addresses....14
4.7 Calibration of SRF10 14
5 SRF10 Library....17
5.1 Library overview....17
5.2 Implementation of Library....18
6 Mounting SRF10 Sensor....20
6.1 Pin connections of Evalboard....20
6.2 Pin Connections of Quadrocopter....22
6.3 Firmware download to Quadrocopter Board....23
7 I²C-Bus....24
7.1 General information about I²C....24
7.2 Clock and cable length....24
7.3 Addresses in use....24
7.4 Data transmission....25
8 Simulation....26
8.1 Requirements....26
8.2 Model Overview....27
8.3 Height Controller....29
8.4 Velocity Calculation....33
9 Software for HIL- Tests and Quadrocopter....35
9.1 Changes....35
10 Summary....40
11 Future Prospects....40
12 Addendum....41
12.1 Bibliography....41

1 Introduction

This is the Quadrocopter.

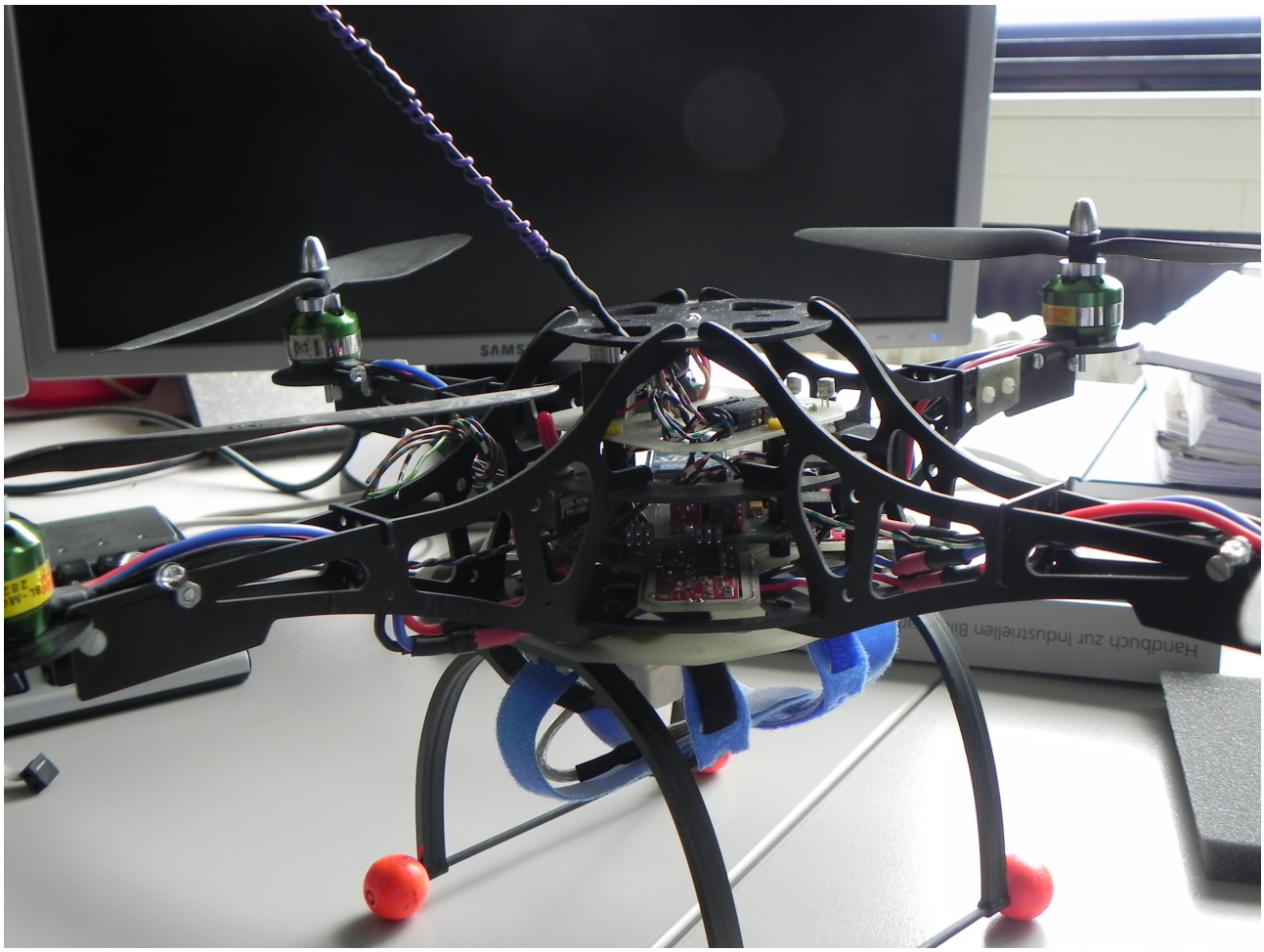


Illustration 1: Quadrocopter

A Quadrocopter is a remote controlled model aircraft with four propellers, arranged in a single plane.

It can tilt, rotate and change it's height by adjusting the rotation speed of the propellers.

To hover, the rotation speed has to be adjusted that the acceleration upwards is the same as the gravitational acceleration.

1.1 Problem

By tilting, the Quadrocopter accelerates horizontally, but loses some height, because the acceleration needed to hover is now split in two components: one vertically, opposing the gravitational acceleration, and one horizontally.

To oppose this effect, the Quadrocopter will be fitted with a sensor to measure the actual flight height. The rotation speed of the propellers has to be automatically controlled to avoid losing height, this means typically increasing the rotation speed.

1.2 Solution

It is necessary to measure the actual flight height of the Quadrocopter. So a sensor is needed. This sensor needs to measure the distance between Quadrocopter and floor.

There are several methods of distance measuring. An analysis is needed to determine possible methods of distance measuring and possible kinds of sensors.

Among these sensors one has to be selected. This sensor has to be mounted on the Quadrocopter. To be mounted, the sensor has to be small and light weighted enough. It will be supplied from the Quadrocopter battery, so the current draw has to be as low as possible.

An height controller has to be designed, tested and implemented into the Quadrocopter controlling software.

2 Ultrasound

[2.2]

Sound with frequencies greater than 20kHz is normally called ultrasound. While frequencies higher than 1GHz can be called hypersound, they are still ultrasound.

Ultrasonic frequencies are greater than the upper limit of human hearing.

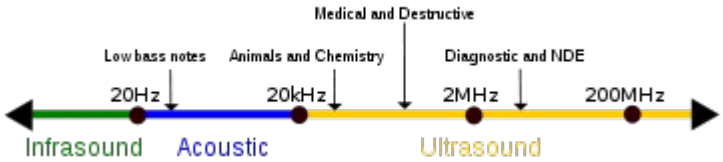


Illustration 2: Sound Frequencies [2.2]

2.1 Uses of ultrasound

[2.1.1]

Diagnostic sonography

Ultrasound is used to visualize muscles, tendons and internal organs via reflection of sound.

Non-invasive sensor

Ultrasound is used for detection of fluids, if inline sensors may contaminate the fluids. Sound travels at different speeds in different materials, this can be used to differentiate the "dry" signal from the signal in fluids.

Industrial ultrasound

Ultrasonic testing is used as a nondestructive testing method to find flaws in materials or measure the thickness of objects.

Ultrasonic cleaning

Ultrasonic cleaners are used at frequencies from 20 to 40 kHz for jewelery, lenses and other optical parts, watches, dental instruments, surgical instruments, diving regulators and industrial parts. An ultrasonic cleaner works mostly by energy released from the collapse of millions of microscopic cavitations near the dirty surface. The bubbles made by cavitation collapse forming tiny jets directed at the surface.

Ultrasonic humidifier

An ultrasonic humidifier works by vibrating a metal plate at ultrasonic frequencies to nebulize water. Because the water is not heated this produces a cold mist.

Ultrasound Identification (USID)

Ultrasound Identification (USID) is a Real Time Locating System (RTLS) or Indoor Positioning System (IPS) technology used to automatically track and identify the location of objects in real time using simple, inexpensive nodes (badges/tags) attached to or embedded in objects and devices, which then transmit an ultrasound signal to communicate their location to microphone sensors. [2.1.2]

Ultrasonic welding

High frequencies (15kHz – 40kHz) are used to create heat by friction between the plastic materials to be joined.

2.2 Creation and Registering

Commonly used for creating ultrasound are speakers using an inverted piezo effect, plates of piezo ceramics are connected with a membrane. By applicating a voltage the membrane swings because of the inverted piezo effect of the plates. High frequencies with low amplitudes can be created.

Registering of ultrasonic waves uses commonly the same technology. A voltage is created by the piezo effect of the plates. [2.2.1]

2.3 Ultrasound Distance Measurement

Flight height measurement means measurement of the distance between Quadrocopter and floor.

Distance measurement means commonly a run time measurement of a signal.

In run time measurement the time a signal needs to travel to the target and back is measured. Using the propagation speed of the signal the distance can be calculated. [2.3.1]

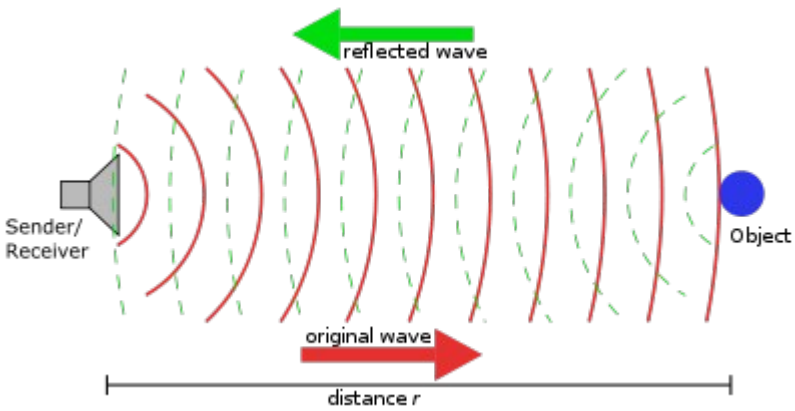


Illustration 3: Range Finding [2.3.2]

Ultrasound travels with the speed of sound. The speed of sound is noted at c = 343m/s in air at an air pressure of 1013hPa and a temperature of 20°C. [2.3.3]

This means:

Distance $d = 2 \cdot c \cdot t$ with speed of sound c and travel time t.

3 Sensor Analysis

To implement an height sensor, the sensor first has to be selected.

There are many distance sensors with different methods of measurement, and this chapter will explain the way the used sensor was selected.

3.1 Aspects

To find a fitting sensor, there are certain aspects to be considered. These aspects are now explained.

The aspects in consideration are price, size, weight, energy consumption, minimum range and maximum range of measurement.

Price

The price of the sensor and its components to mount it. Although being in consideration, the price is the aspect with lowest priority, functionality is more important.

Size

The measurements of the sensor. The Quadrocopter can not support big additional components, so the sensor should be as small as possible.

Energy Consumption

Because the Quadrocopter runs on rechargeable batteries and uses four motors draining this batteries, the sensor should consume as little energy as possible in order not to shorten flight time unnecessarily.

Maximum Range

The maximum range of the sensor is also the maximum range of the height controller. To ensure a smooth flight in most of the situations the Quadrocopter is flown, the maximum range of the sensor should exceed 5 meters.

Minimum Range

Most sensors measure the distance via run time measurement. This method commonly requires a minimum distance to the object to function. This is the minimum range of the sensor. To properly attach the sensor under the Quadrocopter this minimum range should not go below 0.05 meters.

Weight

The Quadrocopter as an aircraft has to lift it's own weight with all components into the air. So the capacity is very limited, which means, the sensor should be as light weighted as possible. This is the most important aspect in choosing a sensor.

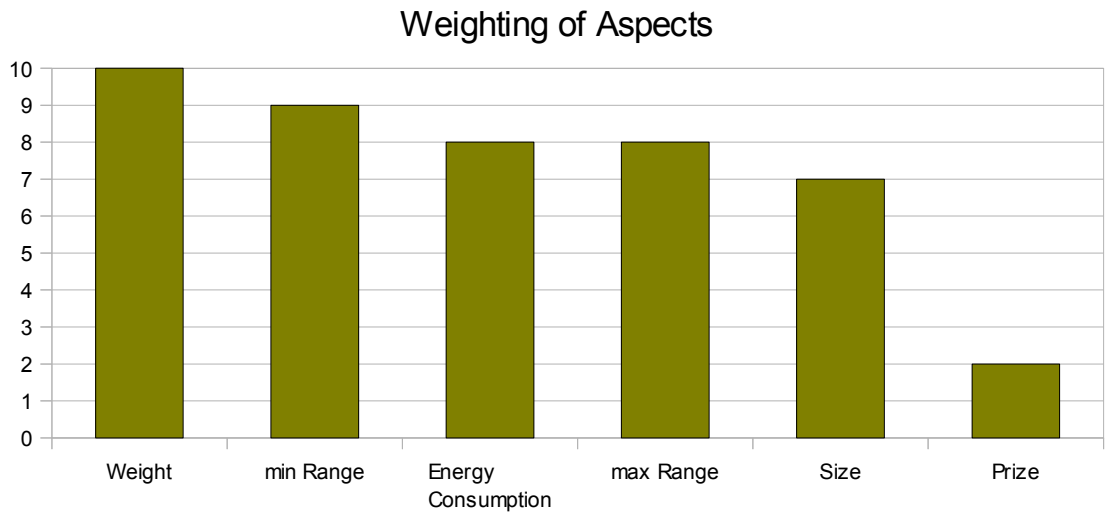


Illustration 4: Weighting of Aspects

This illustration shows the weighting of the explained aspects. As you can see, the method of measurement is not in the aspects. But the requirements will reduce the possible methods to just one.

3.2 Sensor Types and Sensors

Distance measuring of the kind needed is commonly possible with three methods: via infrared light, laser or ultrasound.

For every method there are different sensors, a few examples and the main characteristics of the sensors are explained below.

3.2.1 Infrared

Infrared sensors measure distances with light invisible to the human eye. They are typically small, but the maximum range is very low. There is trouble with shining surfaces.

Example: GP2Y0A02YK [3.2.1.1]

This is an infrared sensor for long distance measuring (as stated in the data sheet). The maximum range of this sensor is about 1.50m, this does not meet the requirements of a maximum range of 5m.

This is an infrared sensor with a very high maximum range. Commonly infrared sensors maximum ranges are under 1m. So infrared sensors can not be used for the Quadrocopter height control.

3.2.2 Laser

In this sensors a laser is used for measurement. This sensors typically have resolutions of 1 millimeter or even less and high maximum ranges, but they are big, heavy and the minimum range is too high. There is trouble with shining surfaces, also are lasers powerful enough to damage a human eye permanently.

Example: VDM28-8-L [3.2.2.1]

This is a small laser sensor. The dimensions are 54mm length, 88mm width and 24mm height. So it is too big to mount under the Quadrocopter. The minimum range is about 0.2m, this also does not meet the requirements. The power drain is about 70mA, a lot higher than the later selected sensor and 90g weight is very heavy for an model aircraft. The maximum range is about 10m and the resolution is 25mm, this is far better than the SRF10.

This sensor reflects the main characteristics of laser sensors. The resolution and maximum range are very good, but they have a high current draw and are too big and heavy for a small model aircraft like the Quadrocopter.

Also this sensor is about ten times more expensive than the SRF10.

3.2.3 Ultrasound

This sensors use sound for measurement. Ultrasound can not be detected by human ears. The sensors are typically small and light, ranges vary widely and energy consumption is low. Because the speed of sound is much lower than the speed of light, sample time is higher than with laser or infrared sensors.

Example: SRF04 [3.2.3.1]

The SRF04 is a small, lightweight sensor, it's specifications meet all requirements.

Example: SRF08 [3.2.3.2]

The SRF08 is a small, lightweight sensor, it's specifications meet all requirements.

4 SRF10 General

This chapter introduces the selected sensor, the SRF10.

The sensor was selected over the SRF04 or SRF08 because it is smaller and the weight is lower.

4.1 General information about SRF10

The sensor is small enough with a length of 32mm, width of 15mm and height of 10mm to fit under the Quadrocopter.

The current draw is about 3mA in standby and 15mA while measuring. A battery with 2000mAh (standard rechargeable battery for RC- models) can sustain the measuring sensor for over 100h. A normal flight of the Quadrocopter is under 5min before the batteries are drained, so the assumption can be made that the sensor will not influence the duration of flight significantly.

The ranges are within requirements with a maximum range of 6m and a minimum range of 0.04m, the resolution is 1cm. The ranges meet the requirements quite well and the error in measurement with the given resolution is neglectable in comparison with the required flight characteristics of the Quadrocopter.

It's weight is 4g, the Quadrocopter weighs about 1kg, so the sensors weight is about 0.4% of the Quadrocopters weight. The Quadrocopter is easily able to carry this sensor and it's wiring.

The sensor connects as an I²C- slave, this bus is present on the Quadrocopter hardware and already in use by the motor controllers. So the sensor can be connected by four simple cables (see I²C- Bus). There are no additional actions or components needed to connect the sensor to the hardware. [4.1.1]

Illustration 5: Sensor: SRF10 [4.1.1]

4.2 Data sheet information

Supply Voltage:	5V
current consumption:	~3mA Standby, ~15mA while measurement
frequency:	40kHz
max Range:	6 meters
min Range:	4 centimeters
weight:	4 gram
measurement value logging	intern, there is no need for an external Controller for time measurement
measurement precision	is in [cm]
interface:	I²C
output format:	µs, cm or inch
feature:	analog gain 40-700 tunable (in 16 steps)
size:	32mm x 15mm x 10mm
manufacturer:	Devantech Ltd

4.3 Connections

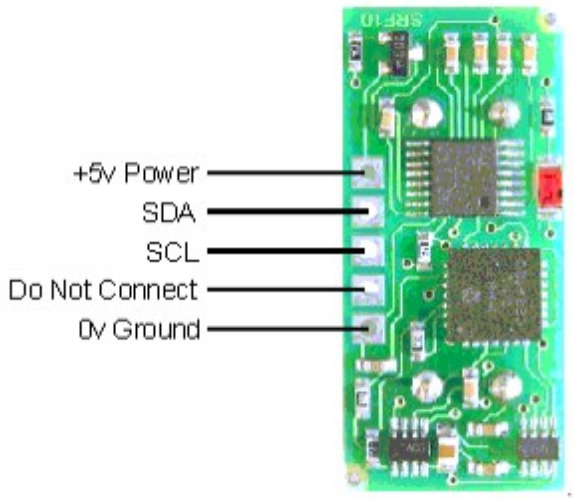


Illustration 7: SRF Pin Connection

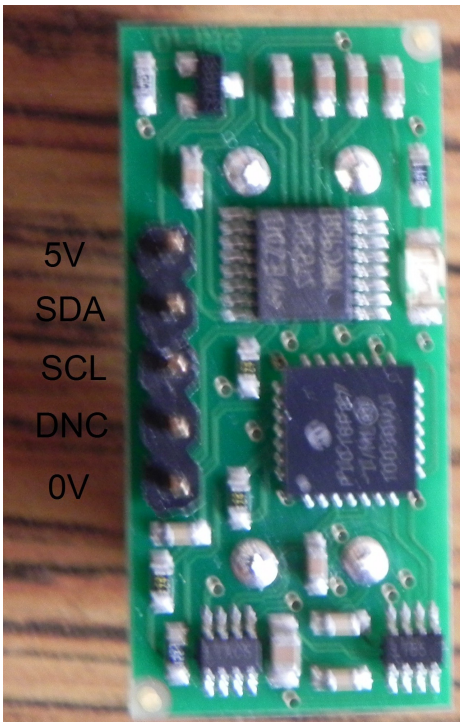


Illustration 6: SRF10 Pin Connection

SDA: serial data line

SCL: serial clock line

4.4 Register overview

This table shows the registers of the SRF10 sensor.

Register #	Read	Write
0	Software Revision	Command Register
1	Unused (reads 0x80)	Max Gain Register (default 16)
2	Range High Byte	Range Register (default 255)
3	Range Low Byte	N/A

4.5 Commands

The SRF10 allows you to measure in inches, centimeters and in micro-seconds.

Besides there is an option to change the address of the sensor.

Command		Action
Decimal	Hex	
80	0x50	ranging in inches
81	0x51	ranging in centimeters
82	0x52	ranging in micro-seconds
160	0xA0	1st in sequence to change I2C address
165	0xA5	3rd in sequence to change I2C address
170	0xAA	2nd in sequence to change I2C address

There is also a possibility to check if a measurement is finished.

Read register 0. If the value is 255 then it's still ranging. Otherwise the firmware version will be returned.

Caution: We tried to use this function to improve the number of measurements. We advice against using this function. If you read too often without any delay in between then there can be some bus errors and the Quadrocopter- Software can occur in a dead-lock.

The most common error is that you miss an acknowledge from the slave (SRF10).

4.6 Possible Addresses

Default address is 0xE0.
The address can be changed in one of the following 16 addresses.

E0	E2	E4	E6	E8	EA	EC	EE	F0	F2	F4	F6	F8	FA	FC	FE

4.7 Calibration of SRF10

SRF10 allows you to change the maximum range and analog gain.

4.7.1 Change Range

The range can be limited to a maximum.

This means echos from objects that are farther away will be ignored.

$$Max\ Range = (43\ mm * Register2) + 43\ mm$$

[meters]	Register 2
11	255
6,02	139
4	92
3	68
2	45

Caution: Range won't be saved. It has to be set after each Power On Reset!

4.7.2 Change Analog Gain

Gain describes the sensitivity of the sensor. The higher the gain, the more an incoming signal will be amplified.

Gain Register		Maximum Analogue Gain
Decimal	Hex	
0	0x00	40
1	0x01	40
2	0x02	50
3	0x03	60
4	0x04	70
5	0x05	80
6	0x06	100
7	0x07	120
8	0x08	140
9	0x09	200
10	0x0A	250
11	0x0B	300
12	0x0C	350
13	0x0D	400
14	0x0E	500
15	0x0F	600
16	0x10	700

"Note that the relationship between the Gain Register setting and the actual gain is not a linear one. Also there is no magic formula to say "use this gain setting with that range setting". It depends on the size, shape and material of the object and what else is around in the room. Try playing with different settings until you get the result you want. If you appear to get false readings, it may be echo's from previous "pings", try going back to firing the SRF10 every 65mS or longer (slower).
If you are in any doubt about the Range and Gain Registers, remember they are automatically set by the SRF10 to their default values when it is powered-up. You can ignore and forget about them and the SRF10 will work fine, detecting objects up to 6 meters away every 65mS or slower." [4.7.2.1]

To measure objects in an angle of +/- 30° you need to set the gain < 8...10 and the range less than 4 meters. This means you need to write in Register 1=0x08 and in Register 2=92

Caution: Gain/Range won't be saved. They have to be set after each Power On Reset!

4.7.3 Change sensor address

- Connect only one sensor to the bus (SDA, SCL)
- use function of SRF10 Library: srf10_changeAddress(oldAddress, newAddress)
- forgot sensor address?

On each power up the LED of SRF10 flashes like shown in the table.

So in case you forgot the sensor address you can "see" it on each power up.

Address		Long Flash	Short Flash
Decimal	Hex		
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6
238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13
252	FC	1	14
254	FE	1	15

5 SRF10 Library

5.1 Library overview

```
void initI2CBus22(void);
uint8 i2cReadOneByte(uint8 slaveAdress, uint8 regName)
void i2cReadTwoByte(uint8 slaveAddress, uint8 regName, uint8 * highbyte, uint8 * lowbyte)
void i2c_SendByte(uint8 slaveAddress, uint8 regName, uint8 data)
```

```
void srf10_starte_messung(uint8 slaveAddress, uint8 Mode)
void srf10_lese_messwert(uint8 slaveAdress, uint8 * highbyte, uint8 * lowbyte)
void srf10_changeAddress(uint8 oldAddr, uint8 newAddr)
void srf10_changeRange(uint8 slaveAdress, uint8 newRange)
void srf10_changeGain(uint8 slaveAdress, uint8 newGain)
```

void InitI2CBus22	Set possible bus speed: 400 kHz, 300 kHz, 200 kHz, 100 kHz, 50 kHz, 40 kHz, 37 kHz enable IIC module, disable interrupts, clear status flags
uint8 i2cReadOneByte	Read one Byte, read-data will be returned
void I2cReadTwoByte	Read register R and R+1. Data will be written to the address of the two variables
void i2c_SendByte	Send data to specified register
void srf10_starte_messung	Start a measurement with one of three ranging option Mode: 0 = inch 1 = cm 2 = µs
void srf10_lese_messwert	Read register 2 and 3 result have to be calculated result=(highbyte *256)+lowbyte;
void srf10_changeAddress	Change address of SRF10 sensor. *Caution: re-define address after change in implementation defines*
void srf10_changeRange	Change the max range of SRF10 *Caution: after power on reset this value is gone, because it's saved in RAM only.*
void srf10_changeGain	Change the gain of SRF10 *Caution: after power on reset this value is also gone.*

5.2 Implementation of Library

We show a code sample how to use the library functions.
In this sample you can calibrate the gain and range of the sensor without stopping the hardware.
You can measure in centimeters or in micro-seconds

5.2.1 Code Sample for Configuration

```
#include <hidef.h>                  /* common defines and macros */
#include <mc9s12xdt512.h>           /* derivative information */
#include <stdlib.h>
#pragma LINK_INFO DERIVATIVE "mc9s12xdt512"
#include "SRF10_LIB.h"
/* global variables*/
  uint8 hb; /* Hight Byte*/
  uint8 lb; /* Low Byte*/
  uint8 config, newGain,newRange;
  uint16 Entfernung_cm;
  uint32 Entfernung_us;

/* MAIN ROUTINE*/
void main(void)
{
  EnableInterrupts;
  hb=0;
  lb=0;
  config=1;
  newGain=0x06;
  newRange=139;
  Entfernung_cm=0;
  Entfernung_us=0;

  /* Init Section*/
  DDRB = 0xff; PORTB = 0;
  initI2CBus22();

  for(;;)
  {
    srf10_starte_messung(0xE0,1);
    delay10ms();
    srf10_lese_messwert(0xE0,& hb, & lb);
    Entfernung_cm=(hb *256)+lb;
```

```
  if (config==1) /* Set this value while debugmode and you can change the range and or gain*/
  {
    srf10_changeRange(0xE0,newRange);
    delay10ms();
    srf10_changeGain(0xE0,newGain);
    delay10ms();
    config=0;
  }
   /* Measure in µs*/
  //srf10_starte_messung(0xE0,2); /* measure in µs*/
  //delay10ms();
  //srf10_lese_messwert(0xE0,& hb, & lb);
  //Entfernung_us=((hb *256)+lb)/2;
  //delay10ms();

    }/* wait forever */
 /* please make sure that you never leave this function */
}
```

6 Mounting SRF10 Sensor

In this chapter we explain how to connect the sensor to the Evalboard. After that we show how to connect it to the Quadrocopter and flash the new firmware to the Quadrocopter.

6.1 Pin connections of Evalboard

Overview of the Evalboard. In the lower left part are the pins which are needed for the SRF10 sensor.

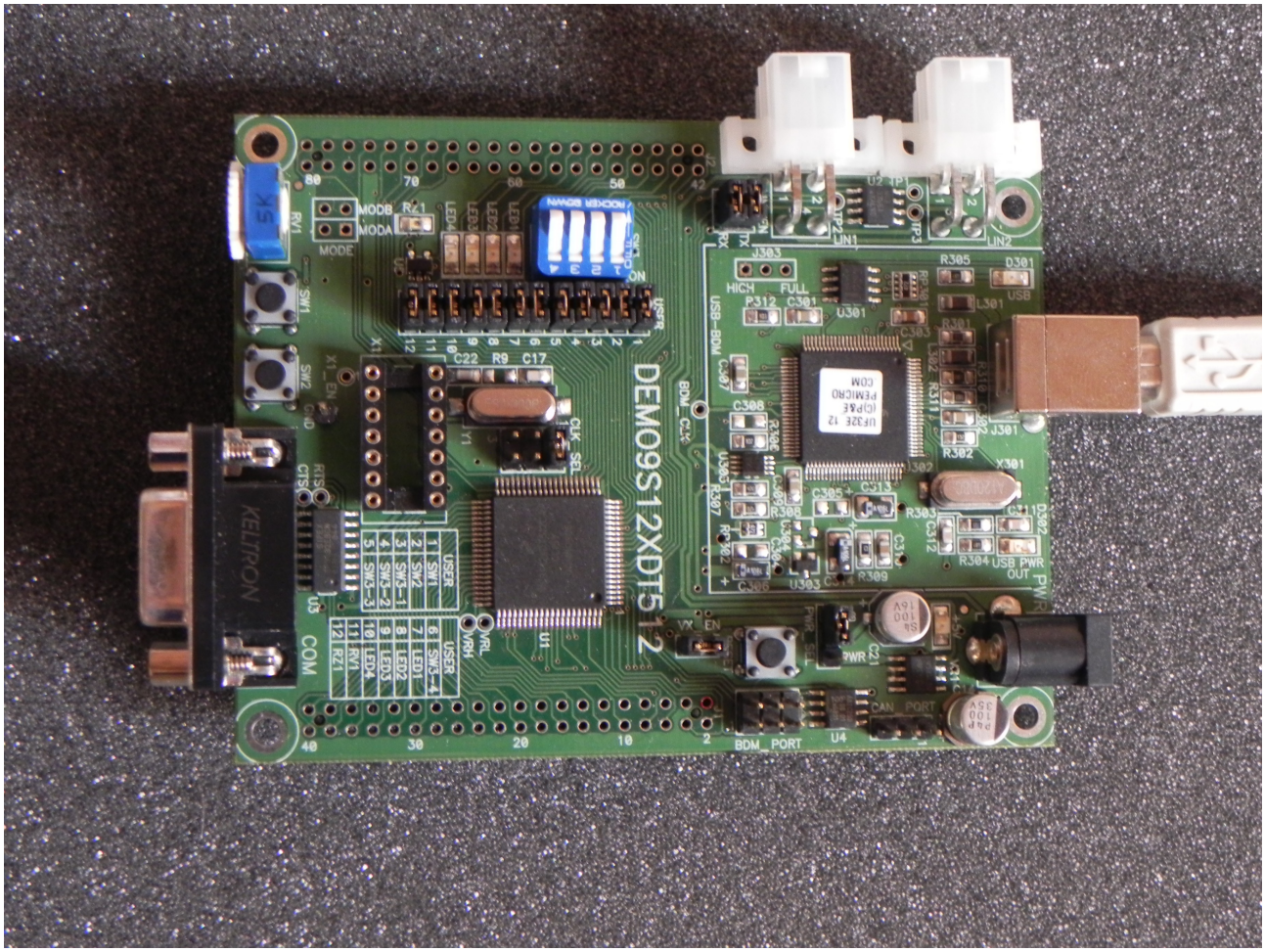


Illustration 8: Evalboard Overview

Pin # (Evalboard)	Input and output signals
1	5V
3	0V Gnd
26	SDA (I²C-Data)
28	SCL (I²C-Clock)

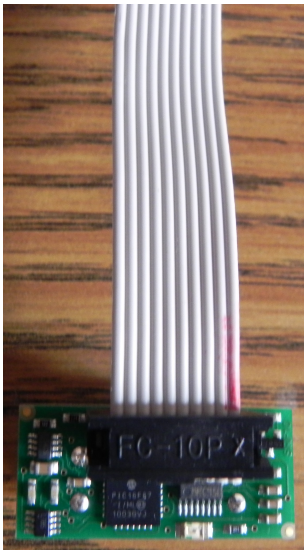


Illustration 9: SRF10

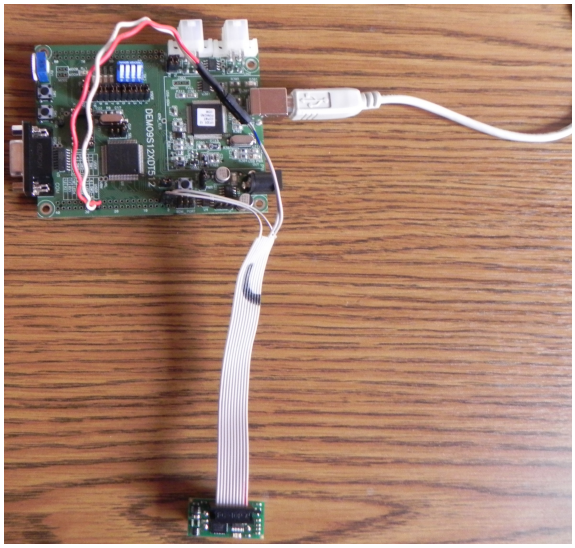


Illustration 10: Evalboard and SRF10

In case you want to use USB as power supply – use the jumper like shown.

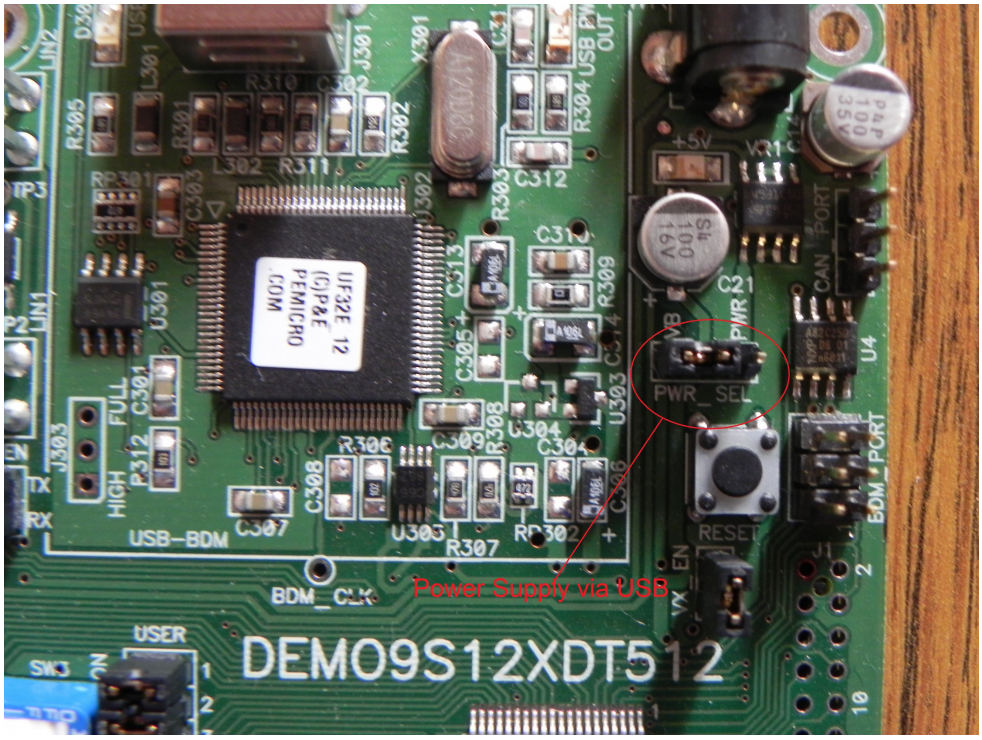


Illustration 11: Evalboard USB Power Supply

6.2 Pin Connections of Quadrocopter

The "black" Quadrocopter got no free I²C interface which is reachable from outside.

That's why a dismount of the black Quadrocopter is required.

We got told that a dismount takes too long and it is very complex.

The I²C pins are under the Xbee module.

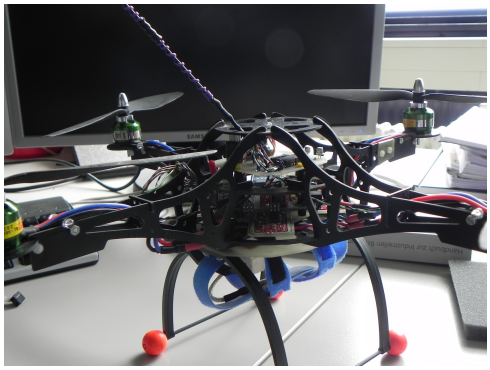
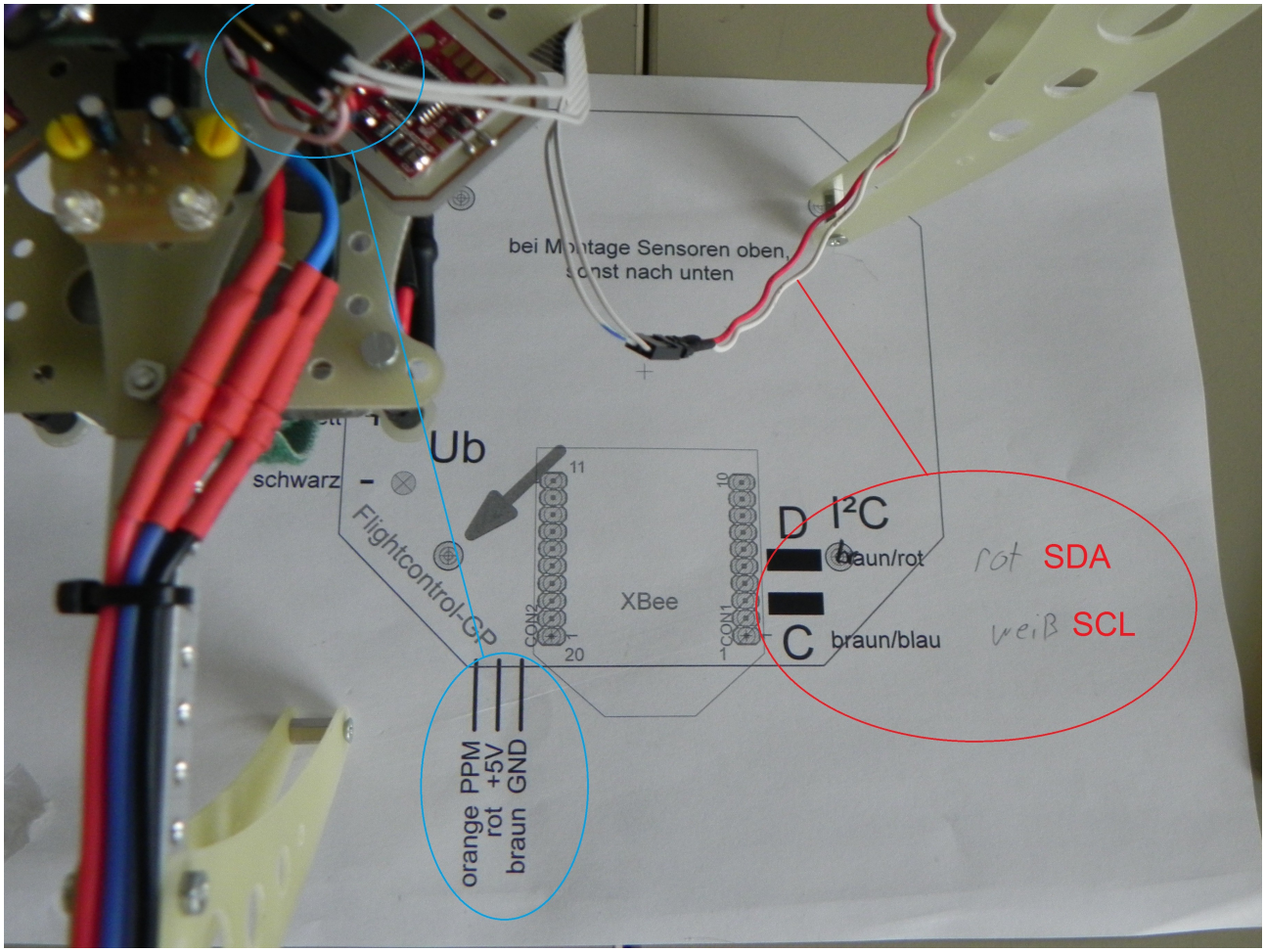


Illustration 12: Black Quadrocopter

To DO: dismount of black Quadrocopter for I²C interface

The "white" Quadrocopter is very simple to dismount. That's why we did choose this one.



Mount the SRF10 sensor below the center of the Quadrocopter. The sensor must be vertical to the ground. If it isn't vertical it could be that the sensor measures a wrong distance!

After that measure distance to the ground. This value is a "#define" in the implemented software.

6.3 Firmware download to Quadrocopter Board

Connect the USB BDM MULTILINK to your computer. Install the driver. Automatically on Windows XP Professional. If this works then the blue LED in on.

Illustration 13: USB BDM MULTILINK

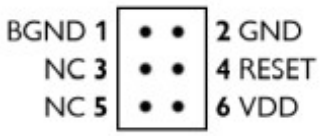


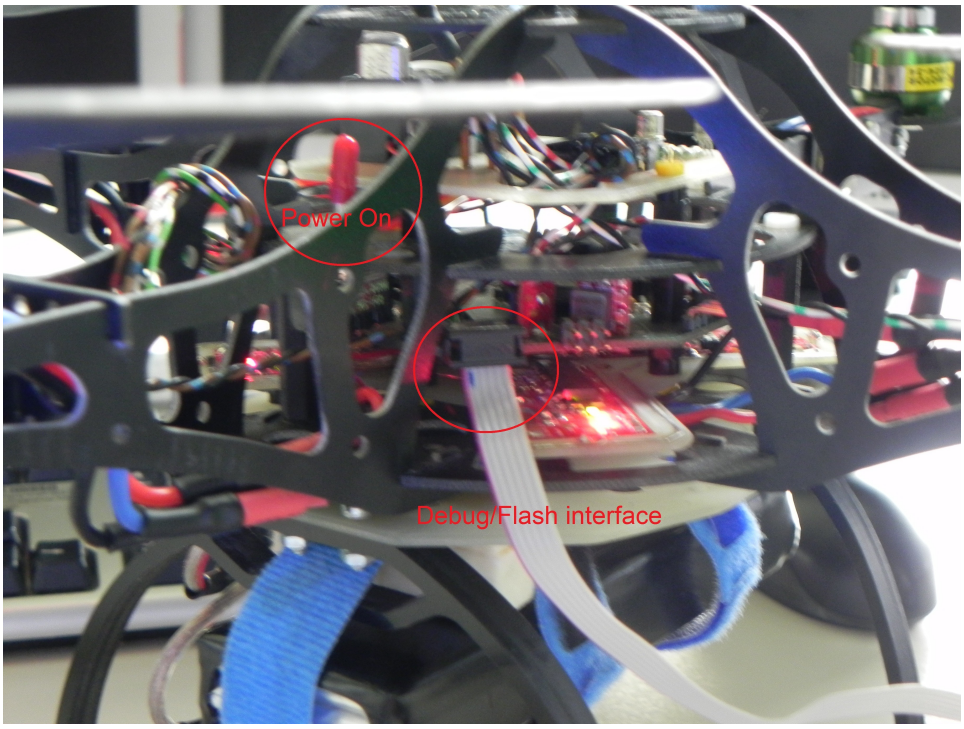
Illustration 14: Debug Pin Connection

After that you can connect the debug cable to the Quadrocopter.

You find the debug interface on the right side of the “power on”.

All 6 Pins must have soldered. Connect the cable that the red one is left.

*Caution: Summer Term 2011 only there exist just one * 6 Pins which are soldered.*



The yellow LED of USB BDM MULTILINK switch to “on” if the Quadrocopter is “on”, too.

7 I²C-Bus

7.1 General information about I²C

I²C (Inter-Integrated Circuit) is a serial data bus developed by Philips Semiconductors (today NXP Semiconductors)

There are three types of messages:

- master writes data to slave
- master reads data from slave
- master write/read more than 2 bytes to one or more slaves

7.2 Clock and cable length

Standard Mode	100 kHz
Fast Mode	400 kHz
Fast Mode Plus	1 MHz
High Speed Mode	3,4 MHz

SRF10 cable length has to be < 30 cm. Otherwise the bus clock has to be reduced to 100 kHz.

Caution: current bus clock is 200 kHz (Summer Term 2011)

7.3 Addresses in use

Currently (Summer Term 2011) following I²C addresses are in use.

Device	I²C address
Brushless Controller Front	0x52
Brushless Controller Rear	0x54
Brushless Controller Right	0x56
Brushless Controller Left	0x58
SRF10 Height Sensor	0xE0

7.4 Data transmission

SDA: serial data line

SCL: serial clock line

Reading one byte from SRF10 slave:

- Master sets START condition
- Master sends slave address
- Slave acknowledges
- Master sends register number
- Slave acknowledges
- Master sets STOP condition
- and after a short delay the START condition
  (repeated start is possible too)
- Master waits for data
- Slave sends data
- Master reads dummy data
- Master sets STOP condition
- Master reads data

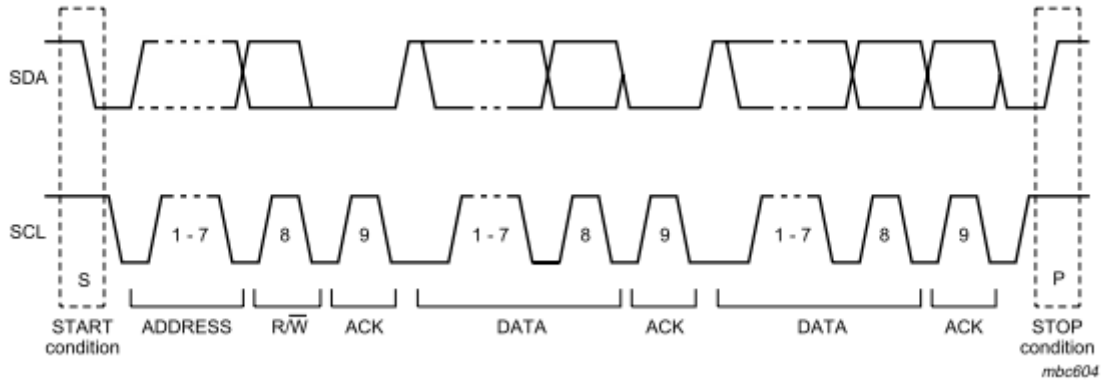


Illustration 15: a complete Data Transfer [7.4.1]

8 Simulation

While developing the height controller, it should be tested before implementing it into the Quadrocopter software.

To test the controller, there is a system simulation, implemented in Matlab. This simulation simulates the dynamics of the Quadrocopter with possibilities of various input methods, for example the real remote control, a 3D mouse, or steering signal patterns.

8.1 Requirements

The requirements for the simulated model are very similar to requirements of the real Quadrocopter.

The system receives as input a target height.

If the actual height, measured by the sensor, in simulation a calculated value, differs from the target height, this error is to be corrected. The Quadrocopter should keep his actual height while flying or changing directions and it should reach the new target height, if the target height is changed.

If the Quadrocopter overshoots while achieving target height, there is a possibility that it crashes into the ground even if target height is larger than 0. Therefore the system must not overshoot.

There are no actual requirements for the dynamics of the system. The Quadrocopter should reach the target height always "fast enough".

8.2 Model Overview

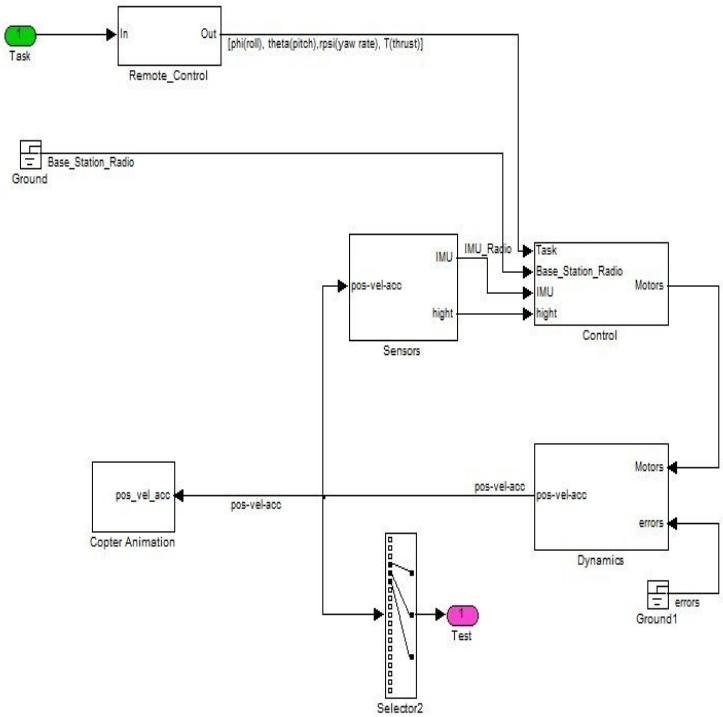


Illustration 16: Quadrocopter Simulink Model

8.2.1 Control

The control block collects signals from sensors, input signals from e.g. the remote control, contains systems to process these signals and outputs to the motors.

8.2.2 Copter Animation

Copter animation visualizes simulated movements of the Quadrocopter in a figure. Position, direction and movement are shown in real time.

8.2.3 Dynamics

Dynamics gets as input the output from the control block. These are the signals for the motors, used to steer the turning speed of every single motor.

The dynamics block converts these turning speeds into forces, accelerations, lateral speeds, turning speeds and eventually position and direction of the Quadrocopter.

It simulates inaccuracies and delays of the motors.

8.2.4 Remote_Control

In Remote_Control the type of input is chosen and input signals can be generated.

The real remote control can be connected to the PC to create input signals for the simulation.

Or a 3D- mouse can be used instead of the remote control.

Actually chosen is a system to create stimuli via repating sequence generators or constants. There are four generators, three for the turning axes of the Quadrocopter and one for the thrust. The sequences should contain values between 0 and 255, while 128 is the neutral value for the turning axes.

For simulating command sequences are used. These sequences can be interpreted as a pilot using a remote control in order to fly the Quadrocopter.

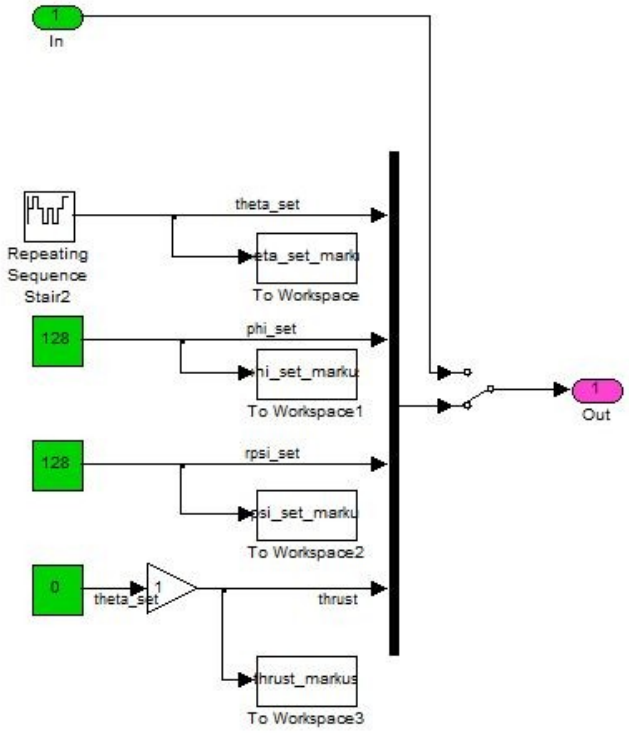


Illustration 17: Remote Control: Stimuli

The illustration shows the commands used to control movements of the Quadrocopter. The sequence stairs and values simulate outputs of the receiver. In this example phi-, psi- axes and thrust are constant, the value for theta is a sequence of different values. This command sequence causes the Quadrocopter to tilt and fly sideways.

Possible values range from 0 to 255

8.2.5 Sensors

This block simulates the sensors of the Quadrocopter. From the model the values for acceleration and turning speed are processed, e.g. noise and delay are added.

The output of this block is used in Control.

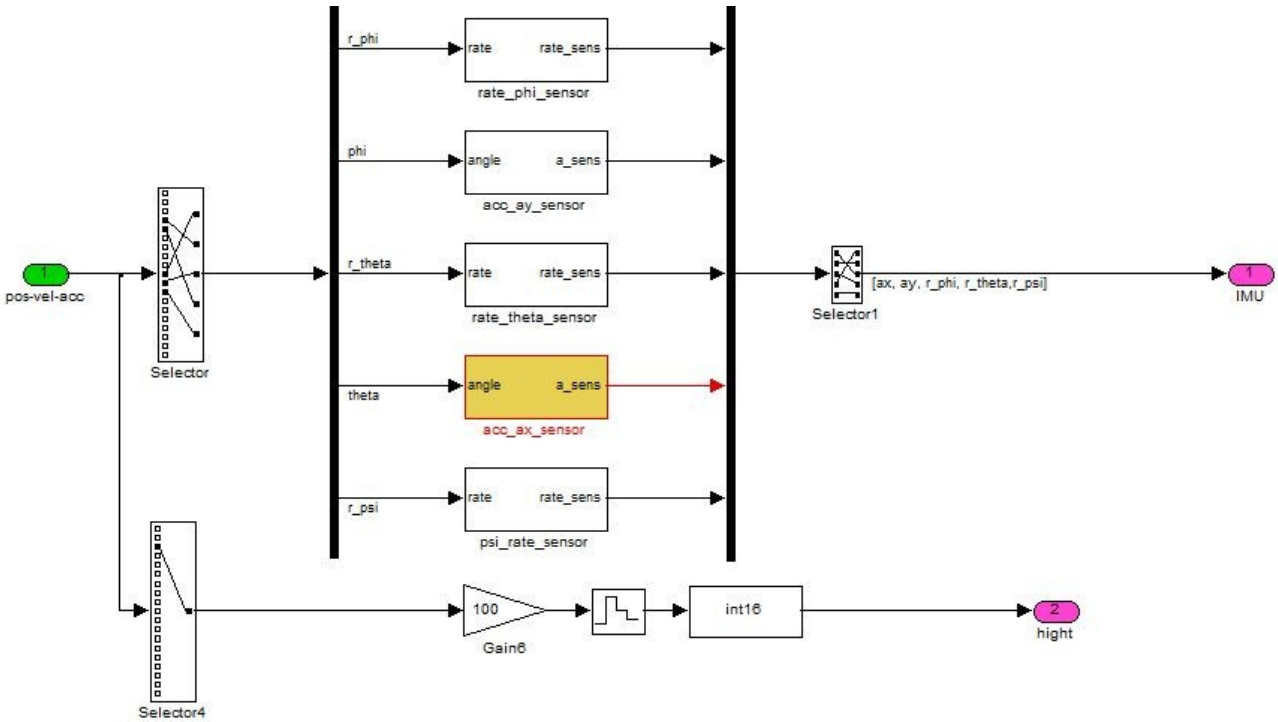


Illustration 18: Sensors

The illustration shows the principles for generating sensor readings. The input pos-vel-acc is calculated by the dynamics block of the simulation.

Pos-vel-acc also contains height. This value is in meters. The sensor measures in centimeters, so height is multiplied by 100, then sampled to simulate measure time. Sample time is Tshight = 0.07.

8.3 Height Controller

The output of the height controller is an abstract value. This value and the values from the position controllers are accumulated, then translated into a rotation speed. The rotation speed causes accelerations for every motor. All four motors cause resultant accelerations in vertical, horizontal acceleration and turning accelerations.

The only relevant acceleration is in vertical direction, for future purpose called z- direction.

Integrated the acceleration az equals velocity vz, a second time integrated height z.

$$z = vz' = az''$$

For simulating the height controller, the Control block of the simulation is modified:

Until now, the input thrust_set from Remote_Control is, accumulated with values from position controllers, the output for the motors. thrust_set input will now be used as input for target_height.

The thrust_set output is replaced by thrust_ctrl, this is the output of the height controller.

The value hight must be set as a new input in control.

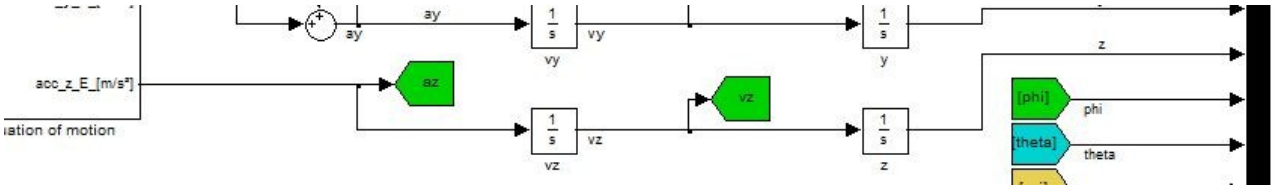


Illustration 19: image section from dynamics: integrators

8.3.1 PID Controller

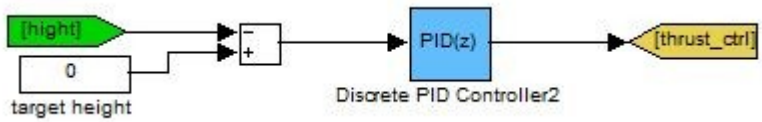


Illustration 20: PID- controller

This is a simple PID- controller.

It is used as P- controller with an amplification of 150. This value is inspired by already existing position controllers.

Following illustrations demonstrate the behaviour of the system.

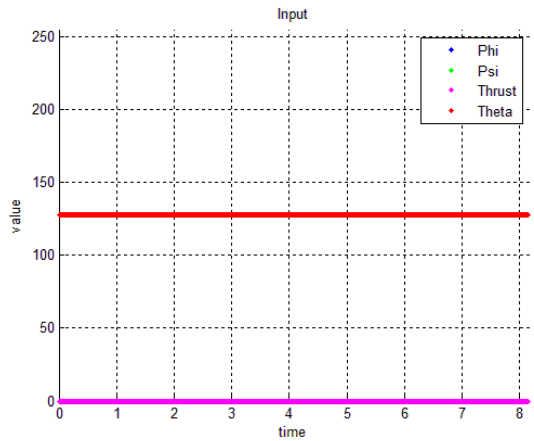


Illustration 21: PID- controller: stimuli

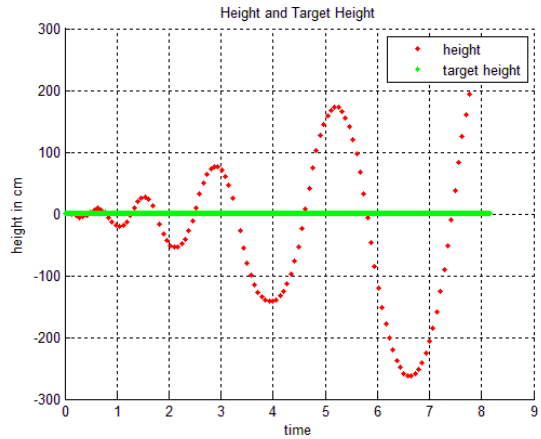


Illustration 22: PID- controller: height

The illustration of height and target height shows: the system is not stable.

The reason is: there are two integrators from the output to the input of the controller. Every integrator causes a phase shift of -90°, two integrators cause a phase shift of -180°. For a system to be stable with a phaseshift of -180° : amplification P < 1.

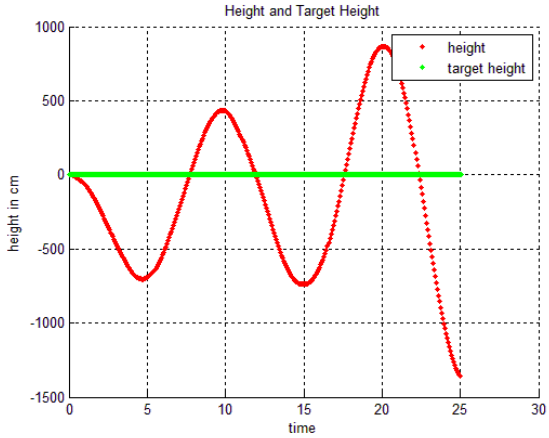


Illustration 23: PID- controller: height, P = 0.5

For values of P < 1, for example P = 0.5, the system reacts too slow, also still the system is not stable:

This means:

a simple PID- controller ist not able to control this system.

8.3.2 PID Controller with looped back velocity

For a stable system the velocity hight_vz is looped back as input for the controller.

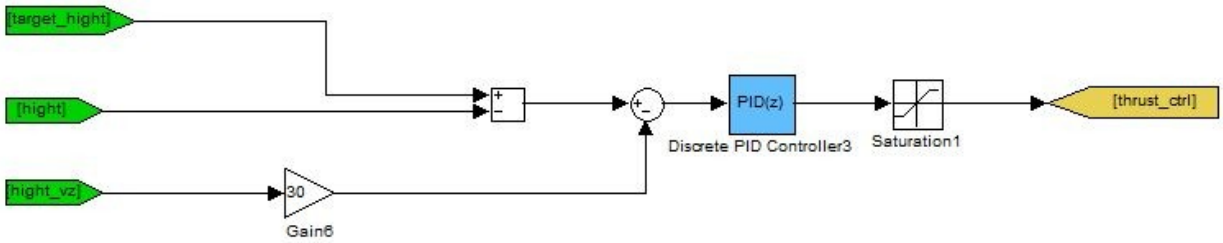


Illustration 24: PID controller, Velocity looped back

There is a new saturation for the controller output. Experiments with the former PID-controller demonstrated, that great output values of the height controller overpower the positon controllers rendering them disabled. The saturation clips the output to values between -1500 and 1500.

Gain6 detemines the weight the velocity influences the controller with.

The velocity hight_vz is a virtual value, it can be calculated, but not measured.

The simulation block dynamics calculates the velocity as intermediate result while calculating the height from acceleration ax.

For the real Quadrocopter the velocity has to be calculated, in the simulation the intermediate result can be used as an abstraction.

The following illustrations show an experiment with P = 150 and Gain6 = 30:

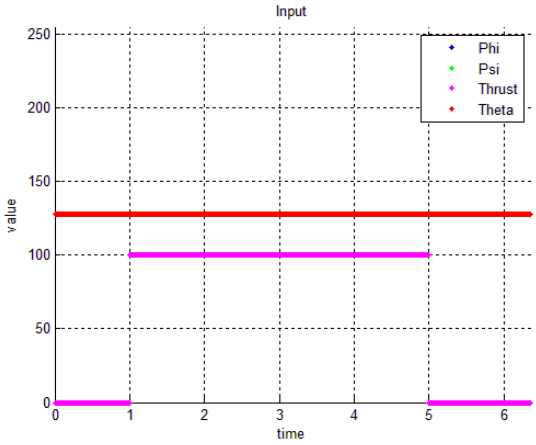


Illustration 25: Cascade PID: stimuli

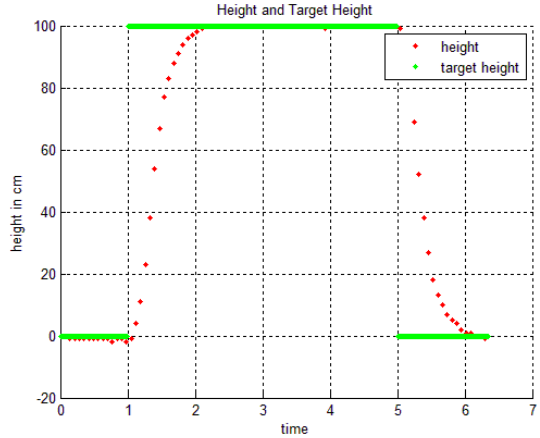


Illustration 26: Cascade PID: height

The first illustration shows a change in the thrust value. This is the input for the controller, in this experiment the target height is changed to detemine, wether the simulated Quadrocopter follows the change as seen in the second illustration: the simulated Quadrocopter reaches the new target height, overshooting can be neglected.

The behaviour of the system meets the required behaviour.

Another simulation is to test the systems ability to reach and hold the target height while compensating for more changes in target height and position of the Quadrocopter:

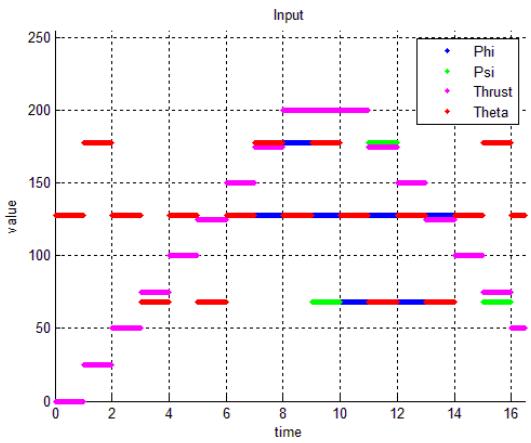


Illustration 27: Cascade PID: stimuli

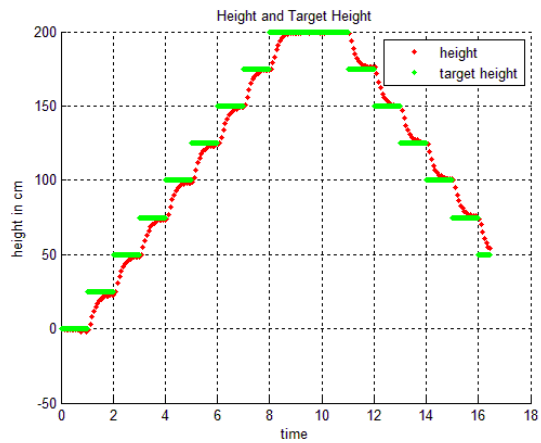


Illustration 28: Cascade PID: height

The right illustration shows the Quadrocopter, following the target height, despite other commands for turning and tilting shown in the left illustration.

8.4 Velocity Calculation

As mentioned before, velocity vz can not be measured, it has to be calculated.

It can be calculated by using acceleration az by integration. But in every step there is an error. This error will sum up over time and can cause problems controlling flight height.

Another method calculating velocity is by differentiation.

As the Quadrocopter moves along it's z- axes, the sensor will measure different heights. So the difference in height z at one moment t1 and another moment t2 equals the average velocity from t1 to t2: vz = z(t1) – z(t2)

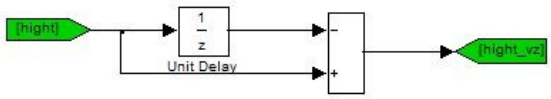


Illustration 29: Velocity Calculation

This illustration shows the calculation of the actual velocity for the simulation.

Calculated is not the actual velocity, but an average velocity during the last simulation cycle.

This is the behaviour of the Quadrocopter, if the velocity is calculated:

The behaviour of the system is like the behaviour of a PID controller without loopback.

The velocity hight_vz is calculated correctly. The problem is: the values for height z are discrete in cm. But the Quadrocopter does not always travel more than 1cm per measurement. This would require a velocity of: $vz = \frac{z(t1) - z(t2)}{t} = \frac{0.01}{0.07} = 0.14$

This means, the Quadrocopter has to travel more than 14cm per second. But near target height, the Quadrocopter is slower, so hight_vz = 0 the more often the lower vz is. If hight_vz = 0, it causes the same effect as if it was not looped back, the behaviour of the system becomes as in a PID controller without loopback, which is, as already mentioned, instable.

To solve this problem, the assumption is made that velocity never becomes 0 because the Quadrocopter will never be perfectly at target height, so it will always rise and sink a few centimeters.

Because of this, the velocity does not need to be exact, it can be approximated.

An approximation is a calculated mean value of hight_vz, calculated from the latest few values. This method creates a phase shift, so the system becomes more and more instable, the more values are used to calculate the mean value. The less values are used, the more hight_vz becomes 0, causing the system to become unstable again.

Caution: Simulations using this approximations were all unstable.

Another approximation is, to store the last value of the velocity hight_vz and hold this value, until a new value other than 0 is calculated. This approximation is very imprecise, but as mentioned earlier: the velocity does not need to be exact.

Caution: This approximation was not implemented and remains to be tested.

9 Software for HIL- Tests and Quadrocopter

For a hardware in the loop (HIL) test and for the Quadrocopter to use the controller it has to be implemented. This chapter shows the changes made in the Quadrocopter controller software.

9.1 Changes

9.1.1 main.c

The main.c contains the starting point of the software.

The functions in the SRF10 library have to be included into the project:

```
#include "QH_SRF10_LIB.h"
```

To initialise the sensor, set range and gain, the init function has to be called:

```
srf10_init(0xE0,139, 0x06);  /* slaveAddress,range, gain */
```

9.1.2 copter.h

This file contains structs describing the copters actual state.

Variables for the height controller have to be added:

Struct copterstate

```
/* Altitude Sensor & Control*/
uint16 height; /*0...6000 cm*/
uint8 heightSensor_low, heightSensor_high;
bool heightControl_flag;
float height_vz;
float e_height_1,height_I_State, height_Filter_State;
```

Struct copterconfig (not in use!)

```
/* Altitude Sensor*/
uint8 SRF10_Gain;
uint8 SRF10_Range;
uint8 SRF10_Address;
/* Height Control*/
int16 kpHeightControl;
int16 kiHeightControl;
int16 kdHeightControl;
```

9.1.3 copter.c

In this file the structs are initialised.

As the structs have been changed, this initialization has to be changed accordingly, all variables for the height controll are initilized as 0.

Two static variables are needed.

```
static uint8 heightCtrlcounter=0;
static bool heightSensorRWFlag=1;
```

This function updates the struct copterState. It has to be changed in order to store the actual height.

```
void copterActualizeState(void)
{
   if(heightCtrlcounter==2)
            {
                        heightCtrlcounter = 0;
                      if (heightSensorRWFlag == 1)      //the flag is 1 if the measurement can be startet
                      {
                                  srf10_starte_messung(0xE0,1);
                                  heightSensorRWFlag = 0;
                      }
                      else
                      {
                                    srf10_lese_messwert(0xE0,& (state.heightSensor_high), & (state.heightSensor_low));
                                    state.height=(state.heightSensor_high *256)+state.heightSensor_low;
                                  state.height= state.height-6; /* Befestigung/Konstruktionsbedingter Offset*/
                                  state.heightControl_flag=1;
                                  heightSensorRWFlag = 1;
                       }
            }
            heightCtrlcounter++;
   /*end*/
...
}
```

The software calls this function every 5ms. To complete a measure cycle, the sensor needs longer. So heightCtrlCounter is used to call the procedure every third time, giving the sensor enough time to complete measurement.

Also heightSensorRWFlag is used to alternate access on the sensor. If the measurement is completed, this flag is set to 1, causing the method to start the next measurement. After starting the measurement, the flag is set to 0, the method will not start another measurement before the actual is completed.

If the measurement is completed and returned a valid value, the flag heightControl in the copterState struct is set to 1, this is needed for the controller.

9.1.4 flightcontrol.c

Flightcontrol.c contains the controllers and additional code.

Following code defines values for the PID function, used for the height controller.

```
#define P_Height (1.5)
#define I_Height (0.075)
#define D_Height (0)
#define N_Height (0)
#define heightCtrl_counter_max 5
#define remoteForceRaw_deadzone 5
#define maxTargetHeight 2000
```

Variables needed for height control.

```
static float heightControl_force; //output for the motors
static float heightControl_heightOld; //height of last measurement, needed for velocity
static float m_r_height;
static float heightCtrl_vz;
static float height_vz_puffer[10];
static uint8 hvz_counter;
```

This function contains all controllers. Shown is only code regarding the height controller.

```
void Controll(CopterState* copterState, CopterConfig* copterConfig)
```

The height controller should only be called, if a new height value is available. To ensure this, the flag heightControl_flag is set to 1 in copter.c, if a new value is available.

```
  if (copterState->heightControl_flag == 1)
   //control only if new value is available
  {
        copterState->heightControl_flag = 0;
```

If the actual value for height is 0, this means there has been an error in measurement. In this case, the new height is calculated using the old height and the velocity.

If there is a certain amount of concurring errors as determined in heightCtrl_counter_max, the height controller will be deactivated, the input from the remote controller is directly fed into the motors.

To reactivate the height controller there has to be more than two valid measurements.

```
 if (copterState->height == 0)      //height measuring returns an error
```

```
  {
        heightCtrl_successCounter = 0;
  }
if (heightCtrl_successCounter < 2)
  {
    heightCtrl_errorCounter++;
          heightCtrl_successCounter = 0;
        if (heightCtrl_errorCounter >= heightCtrl_counter_max)
        {
                //deactivate height control
                m_r_height = copterState->remoteForceRaw;
                heightCtrl_errorCounter = (heightCtrl_counter_max + 1); //avoid overflow of counter variable
        }
        else
        {
                copterState->height = (uint16)(heightControl_heightOld + heightCtrl_vz);
                heightControl_heightOld = copterState->height;
                //no change of speed, there's no data
        }
  }
  else //height measuring without error (hopefully...)
  {

          heightCtrl_vz = copterState->height - heightControl_heightOld;
          height_vz_puffer[hvz_counter]=heightCtrl_vz;
          heightCtrl_vz=0;
          for(i=0;i<=9;i++)
          {
            heightCtrl_vz=heightCtrl_vz+height_vz_puffer[i];
          }
          heightCtrl_vz=heightCtrl_vz/10;
          hvz_counter++;
          if (hvz_counter>=9)
          {
          hvz_counter=0;
          }
          copterState->height_vz= heightCtrl_vz;   /*geschwindigkeit anzeigen lassen*/

          heightControl_heightOld = copterState->height;
          heightCtrl_errorCounter = 0;
          heightCtrl_successCounter++;
```

```
        if (heightCtrl_successCounter>=2)
        {
                heightCtrl_successCounter = 2;
        }
    }
```

The target height is directly linked to the input signal from the remote control.

The error in flight height is calculated from actual height, target height and the calculated velocity.

```
//height Error
    heightControl_targetHeight = (copterState->remoteForceRaw)/3;
    e_height= (float)heightControl_targetHeight - (float)(copterState->height);
    e_height=e_height-(200*heightCtrl_vz);
```

The PID Controller is implemented in a function. This is the call to the function.

All variables and defines are defined and initialized.

```
    heightControl_force = PID (P_Height, I_Height, D_Height,N_Height,e_height, &(copterState->e_height_1),
            &(copterState->height_I_State), &(copterState->height_Filter_State),Ts);
```

The motors are directly linked to the input from the remote controller. This has to changed, the variable remoteForceRaw has to be replaced by the output from the height controller: heightControl_force

```
    //Calculate new setpoints of all motors
    M1 = (float)heightControl_force-m_r_Pitch+m_r_Yaw;
    M2 = (float)heightControl_force+m_r_Roll-m_r_Yaw;
    M3 = (float)heightControl_force+m_r_Pitch+m_r_Yaw;
    M4 = (float)heightControl_force-m_r_Roll-m_r_Yaw;
```

10 Summary

The SRF10 ultrasonic sensor was selected for height measuring.

To connect the sensor to the I²C bus we soldered a new interface to the Quadrocopter board.

The Quadrocopter can now be equipped with this sensor.

The implementation of the ultrasonic sensor is completed. There are functions for setting the I²C address, setting gain and range of the sensor, to start measuring, to read bytes from registers and to read the measured distance.

A prototype for an height controller was developed and tested in Matlab/ Simulink.

There are also experiments to implement the height controller for a HIL- test and the Quadrocopter controll software.

11 Future Prospects

Development and implementation of a method to calculate actual vertical speed.

Furthermore implementation and verification of the accelerometer for z direction.

Implementation of a working height controller using a new method for velocity calculation.

12 Addendum

12.1 Bibliography

Chapter 2

[2.1]	http://de.wikipedia.org/wiki/Ultraschall (28.05.2011) http://en.wikipedia.org/wiki/Ultrasound (28.05.2011)
[2.2]	http://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Ultrasound_range_diagram.svg/360px-Ultrasound_range_diagram.svg.png (18.06.2011)
[2.1.1]	http://en.wikipedia.org/wiki/Ultrasound (28.05.2011)
[2.1.2]	[http://en.wikipedia.org/wiki/Ultrasound#Ultrasound_Identification_.28USID.29 (18.06.2011)]
[2.2.1]	http://de.wikipedia.org/wiki/Ultraschall (28.05.2011)
[2.3.1]	http://de.wikipedia.org/wiki/Laufzeitmessung (01.06.2011)
[2.3.2]	http://upload.wikimedia.org/wikipedia/commons/thumb/0/07/Sonar_Principle_EN.svg/400px-Sonar_Principle_EN.svg.png (01.06.2011)
[2.3.3]	http://de.wikipedia.org/wiki/Schallgeschwindigkeit (18.06.2011)

Chapter 3

[3.2.1.1]	http://www.krause-robotik.de/service/download/doku/gp2y0a02.pdf (19.06.2011)
[3.2.2.1]	http://www.produktinfo.conrad.com/datenblaetter/150000-174999/156449-da-01-de-DISTANZSENSOR_VDM28_8_L_IO_73C_110_122.pdf (19.06.2011)
[3.2.3.1]	http://www.roboter-teile.de/datasheets/srf04.pdf (19.06.2011)
[3.2.3.2]	http://www.roboterbastler.de/Downloads/DB/srf08.pdf (19.06.2011)

Chapter 4

[4.1.1]	SRF10 Data Sheet
[4.7.2.1]	SRF10.pdf

Chapter 7

[7.4.1]	http://www.nxp.com/documents/user_manual/UM10204.pdf (20.06.2011)