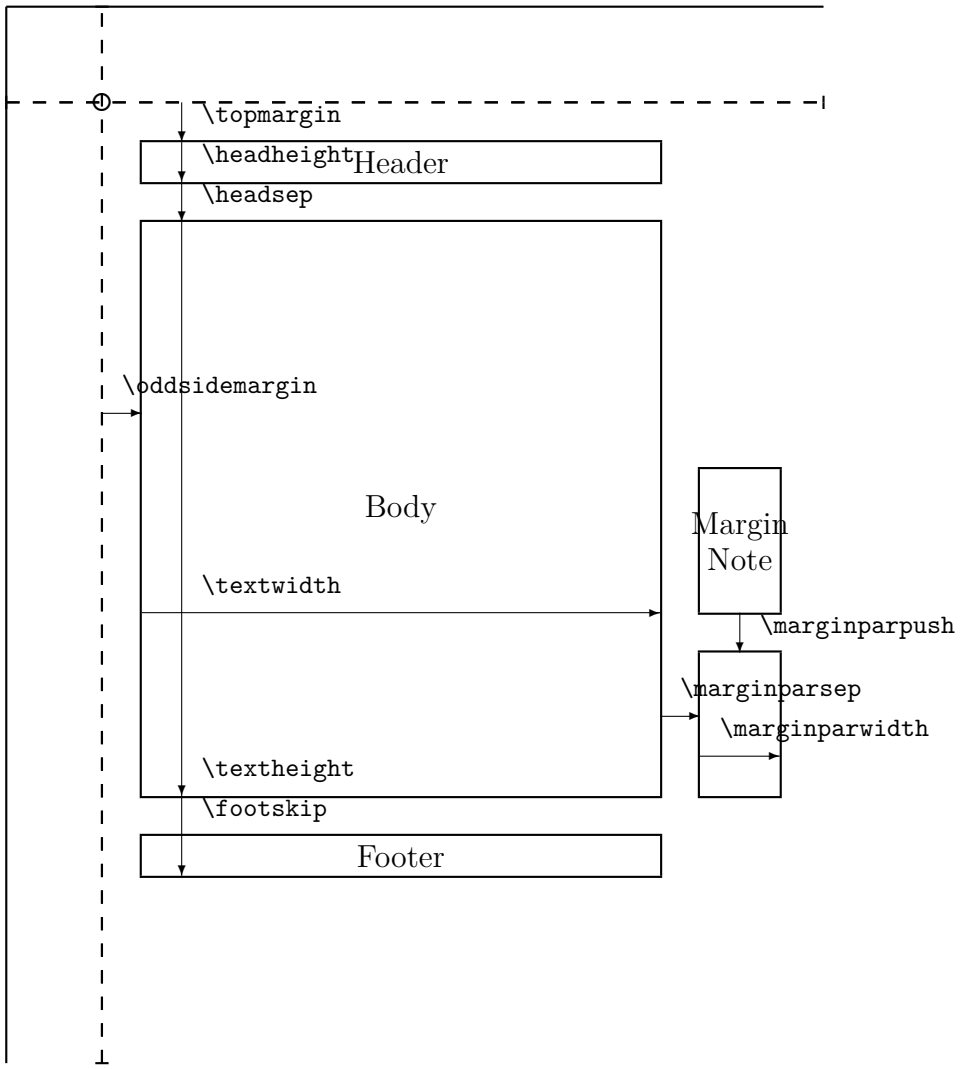
The circle is at 1 inch from the top and left of the page. Dashed lines represent (`\hoffset + 1 inch`) and (`\voffset + 1 inch`) from the top and left of the page.



Actual page layout values.

`\paperheight` = 296.99655mm	`\paperwidth` = 209.99756mm
`\hoffset` = 0mm	`\voffset` = 0mm
`\evensidemargin` = 1.56425mm	`\oddsidemargin` = 1.56425mm
`\topmargin` = -4.18565mm	`\headheight` = 5.60574mm
`\headsep` = 7.64415mm	`\textheight` = 223.35008mm
`\textwidth` = 161.06956mm	`\footskip` = 17.83636mm
`\marginparsep` = 4.51273mm	`\marginparpush` = 2.29323mm
`\columnsep` = 3.51456mm	`\columnseprule` = 0mm
1em = 4.12953mm	1ex = 1.81584mm

`\marginparwidth`: 14.64268mm

Hochschule Esslingen
University of Applied Sciences
Fakultät Informationstechnik

[HElikopter

Masterquad 2015

A Raspberry Pi based quadrocopter platform

A project work of the masters program
Software-based Automotive Systems (ASM-SB)
at Esslingen Graduate School

Oliver Breuning
Jürgen Schmidt
Martin Brodbeck
Phillip Woditsch
Chris Mönch

Period: Summer term 2015
Supervisor:
Prof. Dr. Jörg Friedrich
M Sc. Vikas Agrawal

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Versionshistorie

Version	Datum	Autor(en)	Änderungen
1.0	August 1, 2015	Juergen Schmidt, Oliver Breuning	created
2.0	October 15, 2015	Vikas Agrawal	Changes done for native Ubuntu machine
2.0	December 4, 2015	Vikas Agrawal	Rearranged the doucment for better readability

1 Raspberry Pi based hardware platform

1.0.1 I2C adressing

This section shows all necessary transmissions which are needed for a successful interfacing on the I^2C bus.

- **A/D Converter**

  I^2C slave address: `0b1001001 (0x49)`

  Read

  The read command get's the data from the address, which is stored in the pointer register (blue color). See figure 1.1

Abb. 1.1: Scheme to read data from the ADC

Write

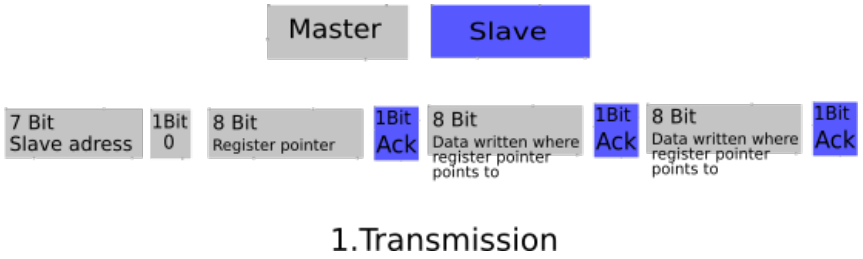


Abb. 1.2: Scheme to write data to the ADC

Read conversion register

To enable a read from a conversion register, several packages need to be sent. They can be seen in figure 1.3. All slave and master acknowledges are not shown because they are handled direct by the interface and so not important for the application.

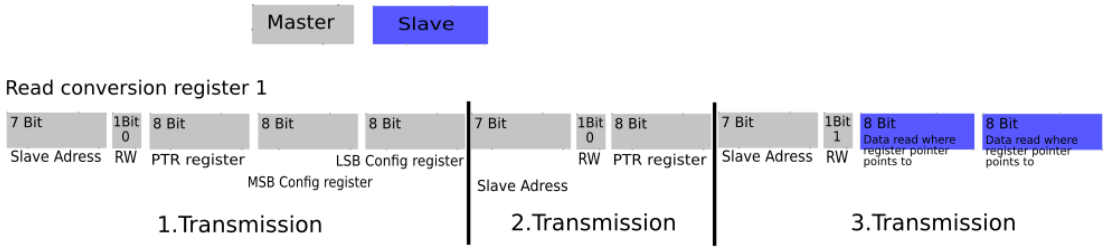


Abb. 1.3: Transmission scheme to read the ADC's conversion registers

- **Inertial Measurement Unit (IMU)**

  The Inertial measurement unit (IMU) has three different chips mounted. Each chip solves one of the measurements of this unit. Each chip has a different I^2C address. All slave and master acknowledges are not shown because they are handled direct by the interface and are not important to the application level.

  - **Acceleration and Magnet Sensor**

    I^2C slave address: `0b0011110 (0x1E)`

    There are several registers which have to be configured before reading and also several register where the acceleration, magnetic strength and if needed temperature can be read. To reduce the amount of pages of this document, they will be not listed here. All the registers can be found in the datasheet `IMU_LSM303D.pdf`, stored in the SVN directory `/doc/se/Datasheets/IMU`.

Read

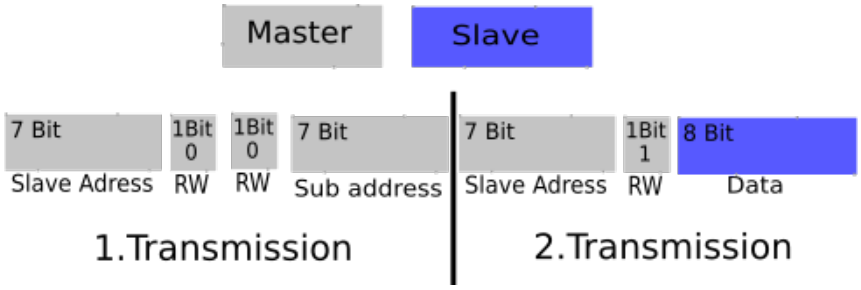


Abb. 1.4: Transmission scheme for a single byte read of the ACC-Sensor

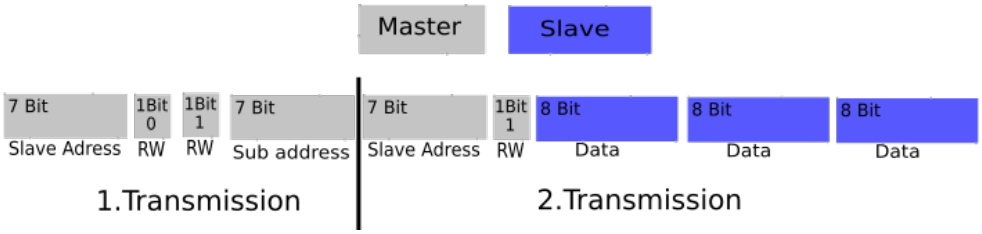


Abb. 1.5: Transmission scheme for multiple data read of the ACC-Sensor(burst read)

1. **Transmission**: Slave address including RW bit ('0'): `0x3C`
2. **Transmission**: Slave address including RW bit ('1'): `0x3D`

Write

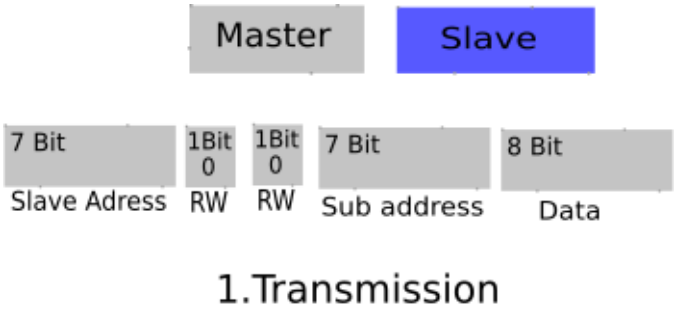


Abb. 1.6: Transmission scheme for a single byte write of the ACC-Sensor

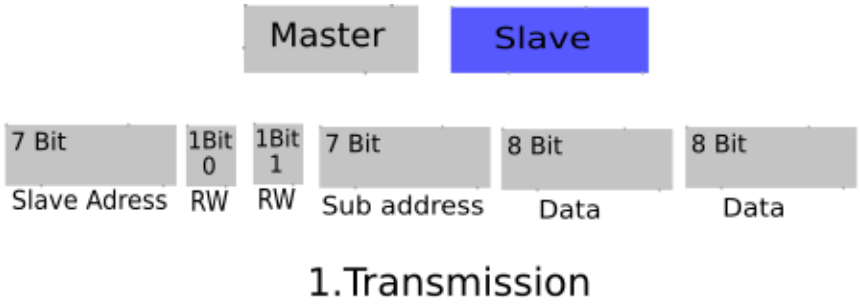


Abb. 1.7: Transmission scheme for multiple data write of the ACC-Sensor(burst write)

1.Transmission: Slave address including RW bit ('0'): `0x3C`

– **Gyroscope Sensor**

I²C slave address: `0b1101010 (0x6A)`

There are several registers which have to be configured before reading and also several register where the rotational speed and if needed the temperature can be read. To reduce the amount of pages of this document, they will be not listed here. All the registers can be found in the datasheet `IMU_L3GD20H.pdf`, which is stored in the SVN directory `\doc\se\Datasheets\IMU`.

Read

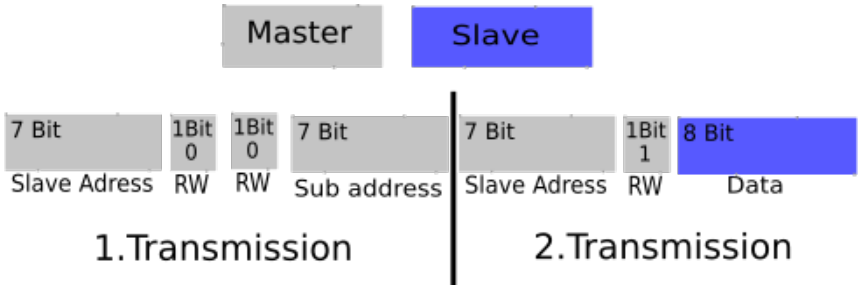


Abb. 1.8: Transmission scheme for a single byte read of the Gyro-Sensor

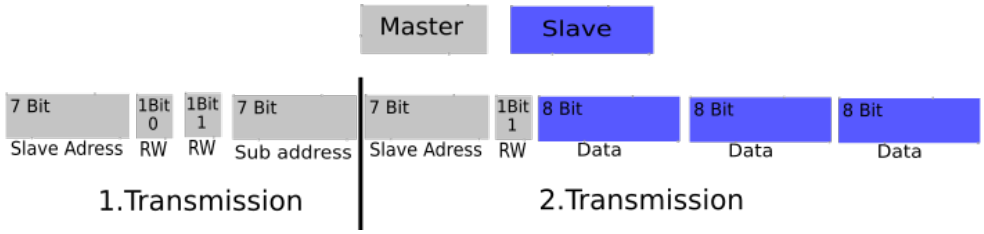


Abb. 1.9: Transmission scheme for multiple data read of the Gyro-Sensor (burst read)

1.Transmission: Slave address including RW bit ('0'): `0xD4`
2.Transmission: Slave address including RW bit ('1'): `0xD5`

Write

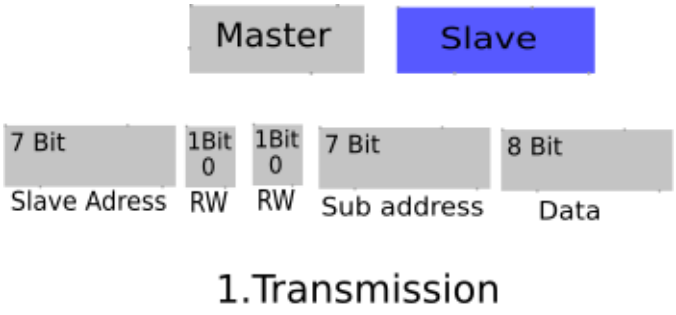


Abb. 1.10: Transmission scheme for a single byte write of the Gyro-Sensor

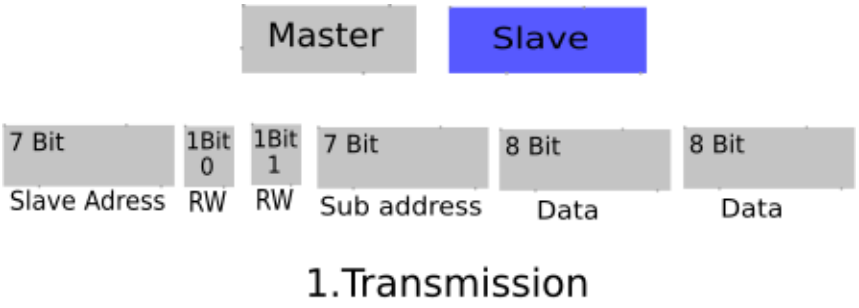


Abb. 1.11: Transmission scheme for multiple data write of the Gyro-Sensor (burst write)

1.Transmission: Slave address including RW bit ('0'): `0xD4`

– **Pressure Sensor**

I²C slave address: `0b1011100 (0x5C)`

There are several registers which have to be configured before reading and also several register where the pressure and if needed the temperature can be read. To reduce the amount of pages of this document, they will be not listed here. All the registers can be found in the datasheet `IMU_LPS331AP.pdf`, which is stored in the SVN directory `\doc\se\Datasheets\IMU`.

Read

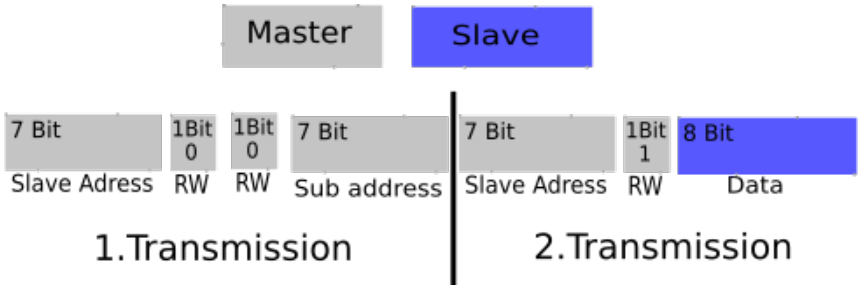


Abb. 1.12: Transmission scheme for a single byte read of the Pressure-Sensor

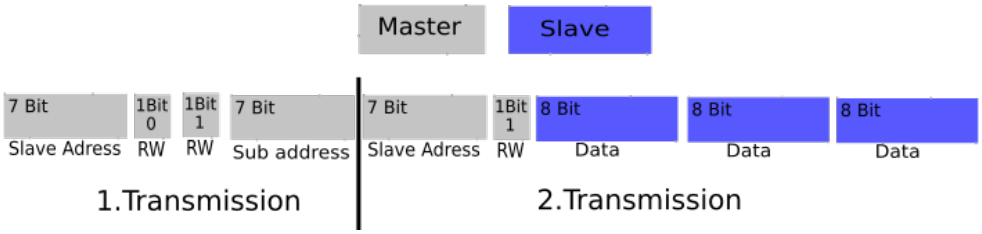


Abb. 1.13: Transmission scheme for multiple data read of the Pressure-Sensor(burst read)

1.Transmission: Slave address including RW bit ('0'): `0xB8`
2.Transmission: Slave address including RW bit ('1'): `0xB9`

Write

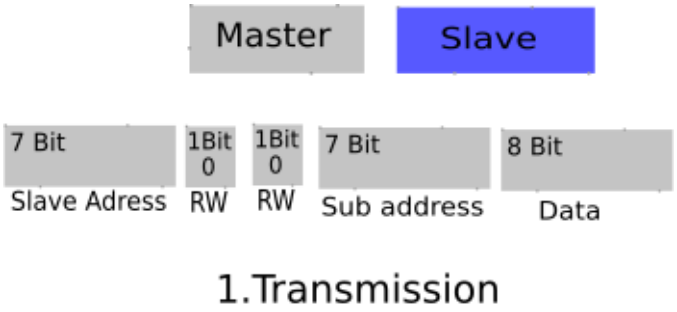


Abb. 1.14: Transmission scheme for a single byte write of the Pressure-Sensor

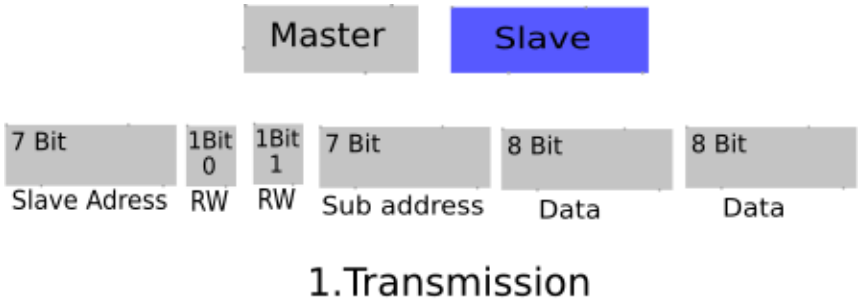


Abb. 1.15: Transmission scheme for multiple data write of the Pressure-Sensor(burst write)

1.Transmission: Slave address including RW bit ('0'): 0xB8

- **Motor Driver**

  All slave and master acknowledges are not shown because they are handled direct by the interface and so not important here. To enable flying with a Quadrocopter there are four motors and so four brushless drivers needed. Each of them has an individual address.

  I²C slave addresses:
  Motor 1 –> `0b0101001 (0x29)`
  Motor 2 –> `0b0101010 (0x2A)`
  Motor 3 –> `0b0101011 (0x2B)`
  Motor 4 –> `0b0101100 (0x2C)`

1.0.2 Read

Read operations: NOT DEFINED

1.0.3 Write

Write operations:

Possible data values are in the range of `10` (Decimal) up to `255` (Decimal) which refers to a hexadecimal range of `0x0A` to `0xFF`.

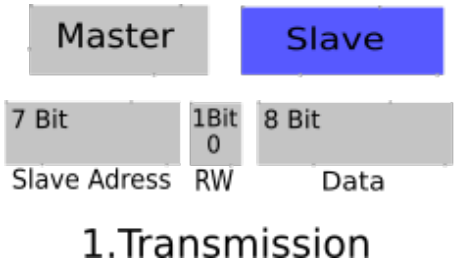


Abb. 1.16: Transmission scheme for a single byte write to a motor driver board

1.Transmission: Slave address including RW bit ('0'):
Motor 1 –> `0x52`
Motor 2 –> `0x54`
Motor 3 –> `0x56`
Motor 4 –> `0x58`

Possible Data values are in the range of 10 (Decimal) up to 255 (Decimal). So in the range from 0x0A to 0xFF.

1.0.4 UART communication

The GPS module uses the UART interface of the Raspberry Pi. The used GPS module has a fixed communication set up.
To ensure working the following values need to be used:

- Baudrate: 9600 baud
- Databits: 8 Bits
- Stopbits: 1 Bit
- Modem control: No

1.1 Technical drawings and packaging

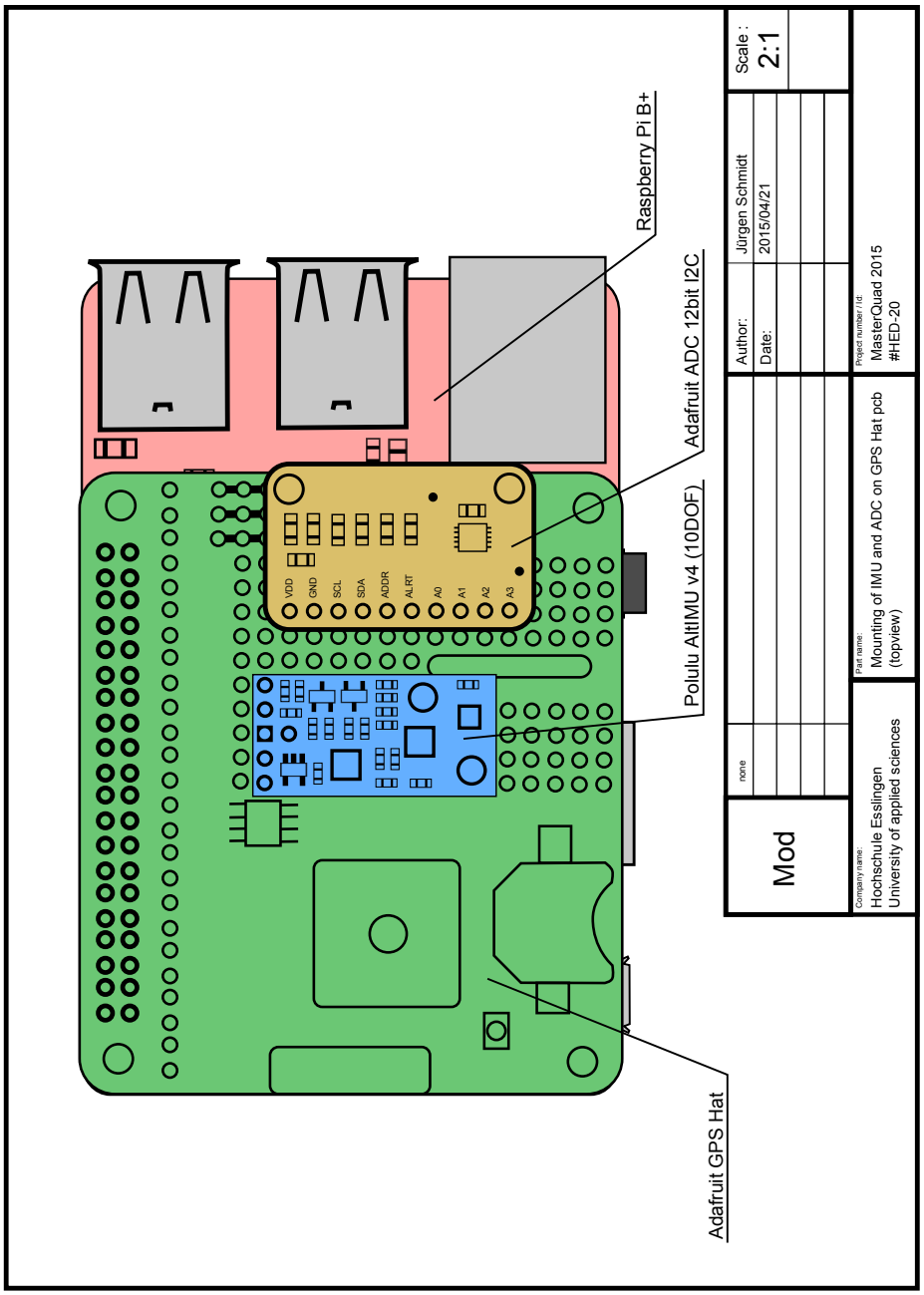


Abb. 1.17: Placing ADC and IMU on GPS Hat. The drawing shows the exact position of the sensor boards, aligned with matching soldering pads of the GPS Hat and the Sensor baords.

To mount the assembled GPS Hat on top of the Raspberry Pi (as depicted in fig. 1.18) the following parts are required:

- 4 pcs. polyamid standoffs, 10mm height (Bürklin, 18H5045)
- 4 pcs. threaded rod, M2.5 in 40mm pieces (Bürklin, 16H322)
- 16 pcs. polyamid shims, M2.5 (Bürklin, 16H942)
- 12 pcs. hexagonal nuts, M2.5 (Bürklin, 16H722)
- 1 pc. GPS Hat with assembled sensors/peripherals as shown in fig. 1.17 and soldered according to 1.19 and fig. 1.20
- 1 pc. Raspberry Pi A+/B+/2

All referenced parts are also mentioned in the Bill of Materials in Chapter 1.2.

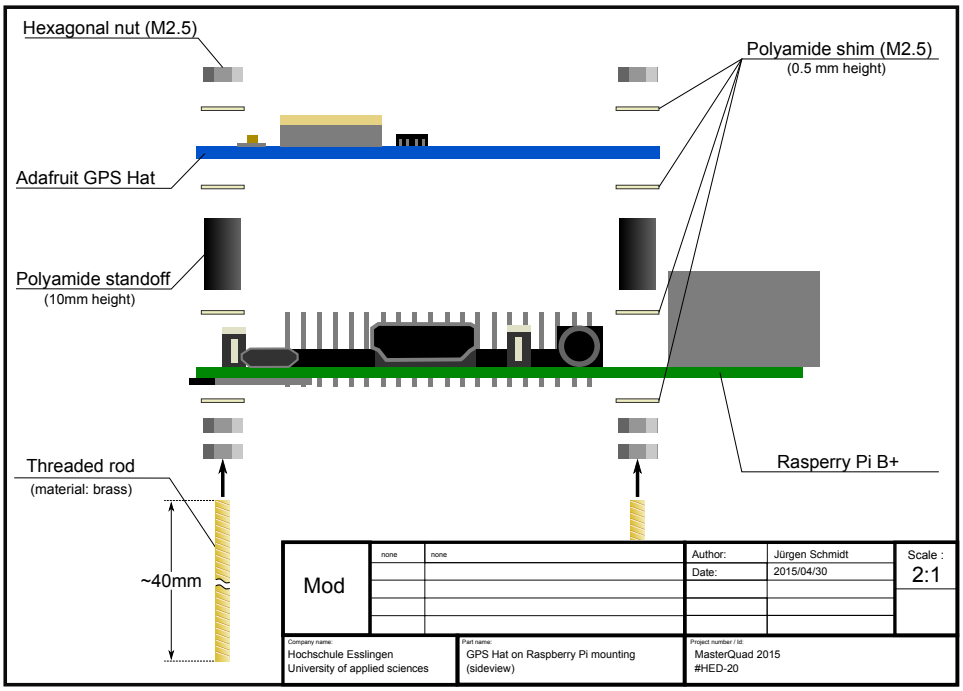


Abb. 1.18: Stacked mounting of Hat-Board on Raspberry Pi

To ease the soldering of the IMU and ADC on top of the GPS Hat board, a soldering layout has been created as shown in fig. 1.19 and fig. 1.20. The layout shows in a one-by-one view the soldering pads of the GPS Hat (one view onto the top, one view onto the bottom). According to the experience of the authors of this document, the soldering layout figures eases the soldering of the sensors to the GPS Hat for students, that are not used to solder a densely packed PCB.

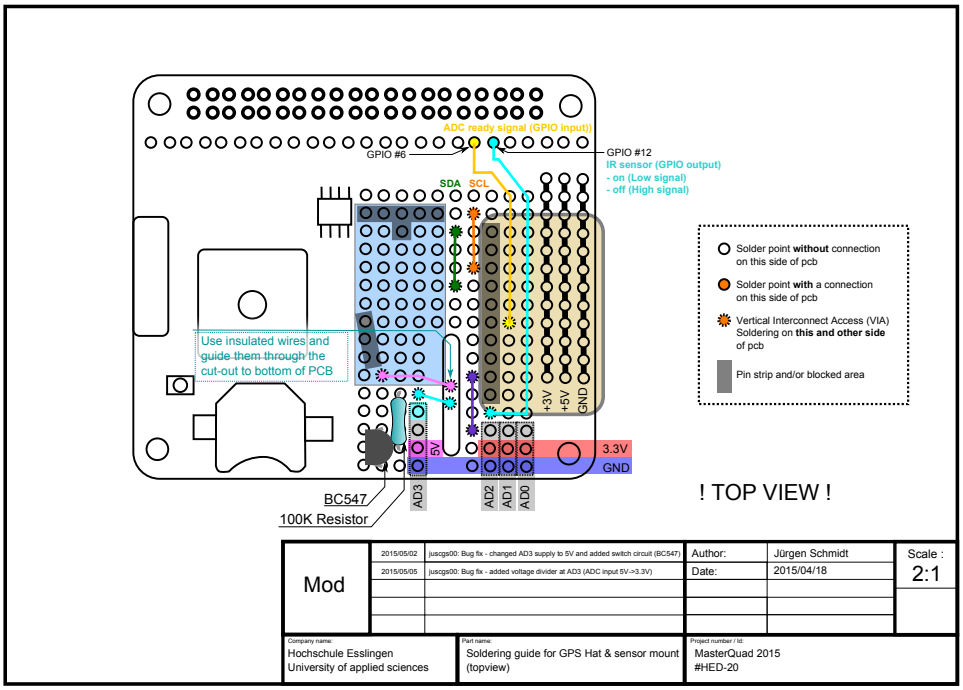


Abb. 1.19: Soldering plan for Raspberry Pi Hat (top view)

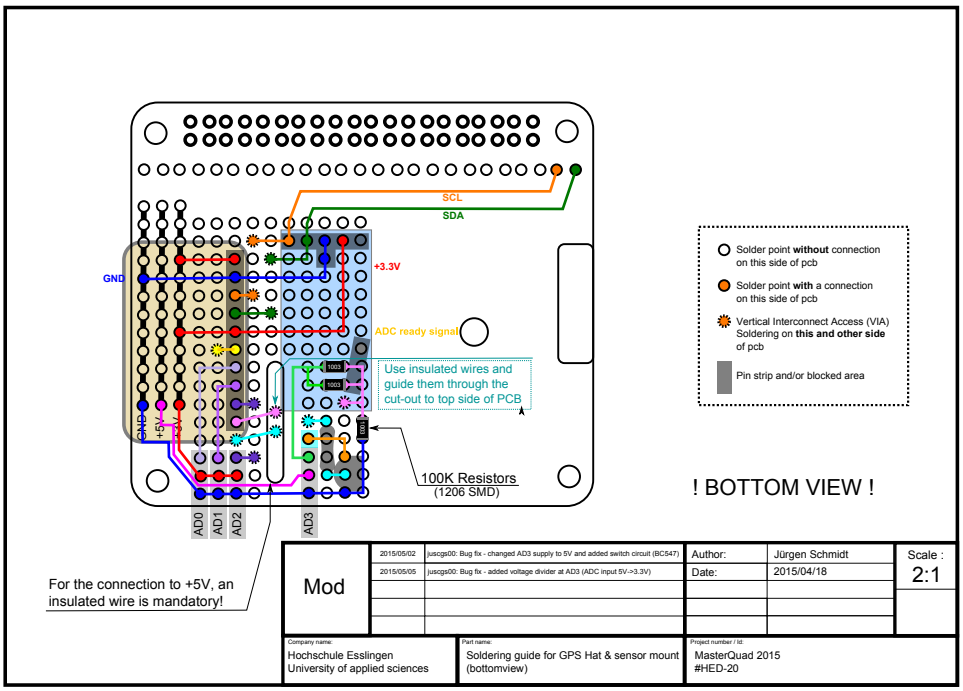


Abb. 1.20: Soldering plan for Raspberry Pi Hat (bottom view)

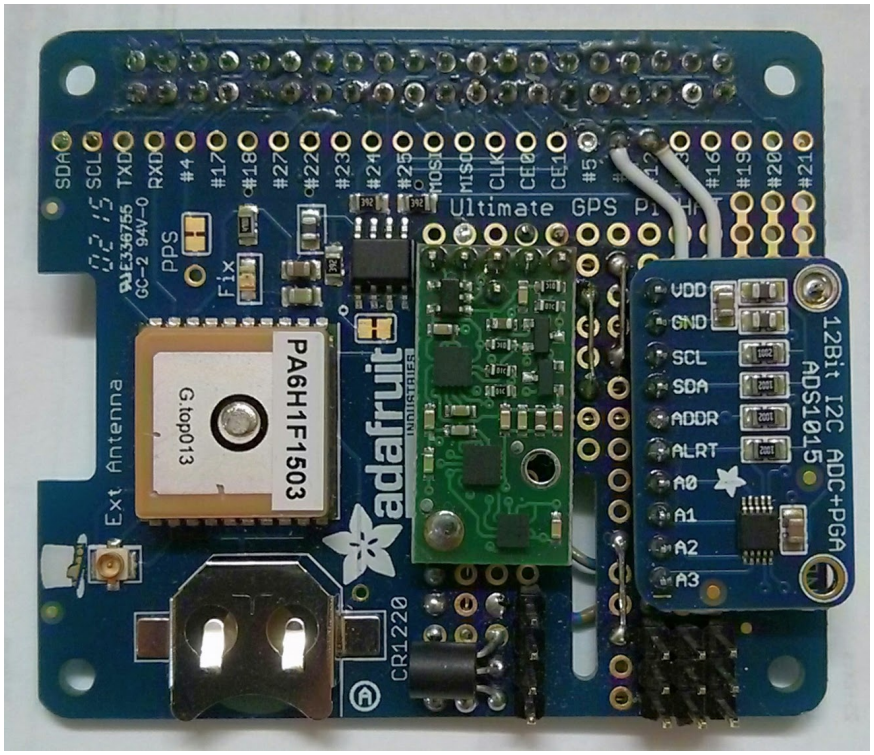


Abb. 1.21: Soldered Raspberry Pi Hat (top view)

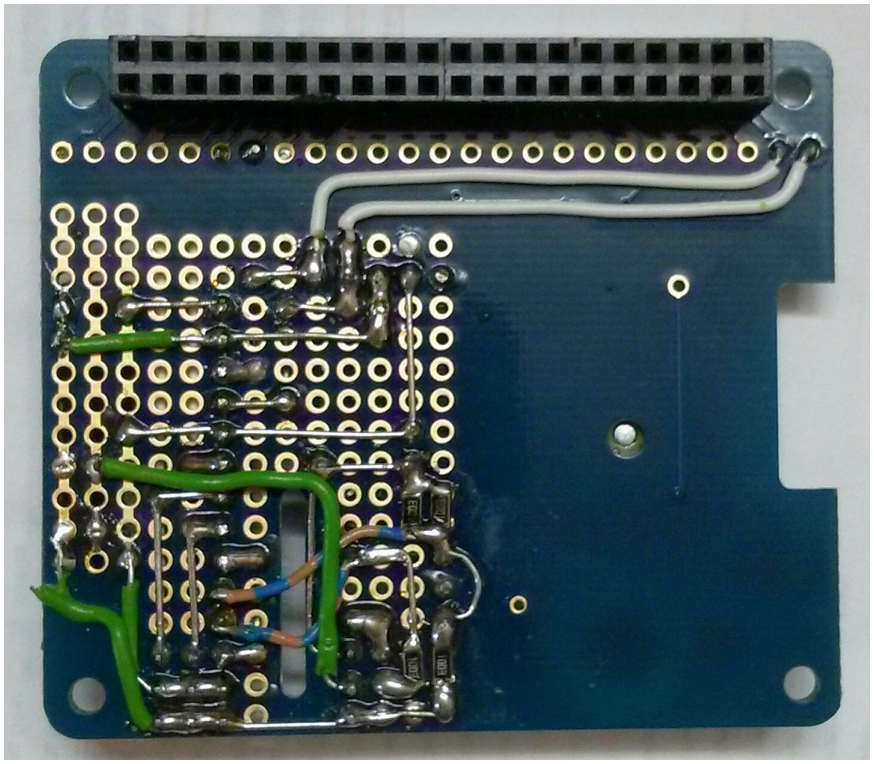


Abb. 1.22: Soldered Raspberry Pi Hat (bottom view)

1.2 Bill of materials

For this project, the parts nedded for the Raspberry Pi Platform has been ordered 5 times. The following Bill of Materials lists all parts including the distributor and parts numbers.

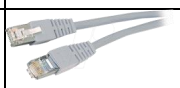
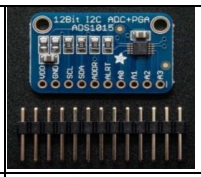
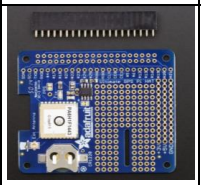
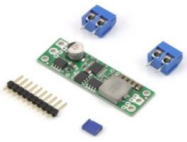
#	Shop	Picture	Description	Art.-Nr.	Price per Unit	Total price	Link
5	REICHELT		Raspberry Pi B+ (All-In Package) including: > Power supply (Micro-USB Stecker, 5V/1.2A) > MicroSD-Card 8GB > Housing for Rasp Pi B+ > WiFi USB-Dongle (EDIMAX 150MBit/s)	RASP B+ ALL IN	59,95 €	299,75 €	http://www.reichelt.de/Einplatinen-Computer/RASP-B-ALL-IN/3/index.html?&ACTION=3&LA=3&ARTICLE=148139&GROUPID=6666
5	REICHELT		HDMI-auf-DVI Cable	AK HDMI-DVI 2,0	3,80 €	19,00 €	http://www.reichelt.de/AK-HDMI-DVI-2-0/3/index.html?&ACTION=3&LA=446&ARTICLE=69657&artnr=AK+HDMI-DVI+2%2C0&SEARCH=hdmi+auf+dvi
5	REICHELT		Cable USB 2.0A to Micro-USB B (1,8m)	AK 676-AB2	1,95 €	9,75 €	http://www.reichelt.de/USB-Kabel/AK-676-AB2/3/index.html?&ACTION=3&LA=3&ARTICLE=133000&GROUPID=6099
5	REICHELT		Cross-Over-Kabel, doppelt geschirmt, 2 Meter	PATCHKABEL-X 2	1,20 €	6,00 €	http://www.reichelt.de/Patchkabel-Netzwerkkabel-cross-over/PATCHKABEL-X-2/3/index.html?&ACTION=3&LA=2&ARTICLE=25802&GROUPID=5848&artnr=PATCHKABEL-X+2
10	REICHELT		Stiftleisten 2,54 mm, 1X04, gerade	MPE 087-1-004	0,10 €	1,00 €	http://www.reichelt.de/Stiftleisten/MPE-087-1-004/3/index.html?&ACTION=3&LA=2&ARTICLE=119881&GROUPID=3220&artnr=MPE+087-1-004
20	REICHELT		Stiftleisten 2,54 mm, 1X05, gerade	MPE 087-1-005	0,12 €	2,40 €	http://www.reichelt.de/Stiftleisten/MPE-087-1-005/3/index.html?&ACTION=3&LA=2&ARTICLE=119882&GROUPID=3220&artnr=MPE+087-1-005
10	REICHELT		Stiftleisten 2,54 mm, 1X10, gerade	MPE 087-1-010	0,25 €	2,50 €	http://www.reichelt.de/Stiftleisten/MPE-087-1-010/3/index.html?&ACTION=3&LA=2&ARTICLE=119885&GROUPID=3220&artnr=MPE+087-1-010
10	REICHELT		Stiftleisten 2,54 mm, 1X20, gerade	MPE 087-1-020	0,33 €	3,30 €	http://www.reichelt.de/Stiftleisten/MPE-087-1-020/3/index.html?&ACTION=3&LA=2&ARTICLE=119888&GROUPID=3220&artnr=MPE+087-1-020
10	REICHELT		Buchsenleisten 2,54 mm, 1X04, gerade	MPE 094-1-004	0,19 €	1,90 €	http://www.reichelt.de/Buchsenleisten/MPE-094-1-004/3/index.html?&ACTION=3&LA=2&ARTICLE=119913&GROUPID=3221&artnr=MPE+094-1-004
10	REICHELT		Präz.-Buchsenleisten 2,54 mm, 1X05, gerade	MPE 115-1-005	0,35 €	3,50 €	http://www.reichelt.de/Buchsenleisten/MPE-115-1-005/3/index.html?&ACTION=3&LA=2&ARTICLE=119953&GROUPID=3221&artnr=MPE+115-1-005
10	REICHELT		Präz.-Buchsenleisten 2,54 mm, 1X10, gerade	MPE 115-1-010	0,71 €	7,10 €	http://www.reichelt.de/Buchsenleisten/MPE-115-1-010/3/index.html?&ACTION=3&LA=2&ARTICLE=119955&GROUPID=3221&artnr=MPE+115-1-010
5	REICHELT		USB2.0 Card Reader All-in-1	DELOCK 91471	6,15 €	30,75 €	http://www.reichelt.de/DELOCK-91471/3/index.html?&ACTION=3&LA=446&ARTICLE=106309&artnr=DELOCK+91471&SEARCH=sd+card+reader

Abb. 1.23: Bill of materials, part 1

5	EXP-Tech		4channel 12bit (3.3kSPS) ADC over I2C	EXP-R15-038	9,25 €	46,25 €	http://www.exp-tech.de/adafruit-ads1015-12-bit-adc-4-channel-with-progrmmable-gain-amplifier
5	EXP-Tech		Pololu AltIMU-10 v4	EXP-R25-371	26,61 €	133,05 €	http://www.exp-tech.de/pololu-altimu-10-v4-gyro-accelerometer-compass-and-altimeter-l3gd20h-lsm303d-and-lps25h-carrier
5	EXP-Tech		Adafruit Ultimate GPS HAT	EXP-R15-665	53,50 €	267,50 €	http://www.exp-tech.de/adafruit-ultimate-gps-hat-for-raspberry-pi-a-or-b-mini-kit
10	EXP-Tech		GPIO Header for Raspberry Pi B+ (stackable)	EXP-R15-621	2,50 €	25,00 €	http://www.exp-tech.de/gpio-header-for-raspberry-pi-b-extra-long-2x20-female-header
5	EXP-Tech		USB to TTL Serial Cable (GPS-Prototyping)	EXP-R15-015	9,47 €	47,35 €	http://www.exp-tech.de/usb-to-ttl-serial-cable-debug-console-cable-for-raspberry-pi?___SID=U
5	EXP-Tech		PULSEDLight LIDAR Lite (Distance-Measurement Unit)	EXP-R05-729	94,90 €	474,50 €	http://www.exp-tech.de/lidar-lite
5	EXP-Tech		Pololu Step-Down Spannungsregler D15V70F5S3	EXP-R25-037	22,80 €	114,00 €	http://www.exp-tech.de/pololu-step-down-spannungsregler-d15v70f5s3

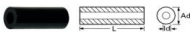
2	Bürklin		Gewindestangen DIN 975 Messing Typ BS. M2.5 (1000mm)	16 H 322	5,21 €	10,42 €	https://www.buerklin.com/default.asp?event=ShowArtikel(16H322)&context=subset:0;sel:SE;search:gewindestange;se_name:vt;patchid:;tags:;pagecount:100&l=d&jump=ArtNr_16H322&ch=25059
1	Bürklin		Schrauben-Sicherungsmittel Typ Ergo 4003/4052/4101, mittelfest (10g)	12 L 585	6,60 €	6,60 €	https://www.buerklin.com/default.asp?event=ShowArtikel(12L585)&context=subset:0;sel:SE;search:Typ%C2%A0Ergo%C2%A04003/4052/4101;se_name:vt;patchid:;tags:;pagecount:100&l=d&jump=ArtNr_12L585&ch=69315
1	Bürklin		Sechskantmuttern Mat. 4.8-Zn DIN 934/ISO 4032 Typ BS 3200, Gew. M 2,5 (100 Stk.)	16 H 722	1,67 €	1,67 €	https://www.buerklin.com/default.asp?event=ShowArtikel(16H722)&context=subset:0;sel:P;search:Sechskantmuttern;patchid:;tags:Sechskantmuttern;pagecount:100&l=d&jump=ArtNr_16H722&ch=61185
50	Bürklin		Distanzrollen aus Polyamid Typ Fastpoint 10 (10090BB0113.5, M 2,5, L 13,5, Ad 5,0, Id 2,7 mm)	18 H 5045	0,08 €	4,00 €	https://www.buerklin.com/default.asp?event=ShowArtikel(18H5045)&context=subset:0;sel:P;search:Distanzh%C3%BClsen;patchid:;tags:Distanzh%C3%BClsen;pagecount:100&l=d&jump=ArtNr_18H5045

Abb. 1.24: Bill of materials, part 2

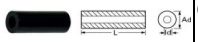
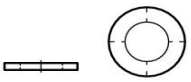
50	Bürklin		Distanzrollen aus Polyamid Typ Fastpoint 10 (10090BB0110.0, M 2,5, L 10, Ad 5,0, Id 2,7 mm)	18 H 5044	0,07 €	3,50 €	https://www.buerklin.com/default.asp?event=ShowArtikel(18H5044)&context=subset:0;sel:P;search:Distanzh%C3%BClsen;patchid:;tags:Distanzh%C3%BClsen;pagecount:100&l=d&jump=ArtNr_18H5044&ch=74532
1	Bürklin		Unterlegscheiben aus PA 6.6 DIN 125/ISO 7089 Typ 05., M2.5 (100 Stk.)	16 H 942	2,83 €	2,83 €	https://www.buerklin.com/default.asp?event=ShowArtikel(16H942)&context=subset:0;sel:SE;search:unterlegschreiben;se_name:vt;patchid:;tags:;pagecount:100&l=d&jump=ArtNr_16H942&ch=10217
				Summe		**1.523,62 €**	
				Price per Kit		304,72 €	

Abb. 1.25: Bill of materials, part 3

2 Individual parts

2.1 Raspberry Pi B+

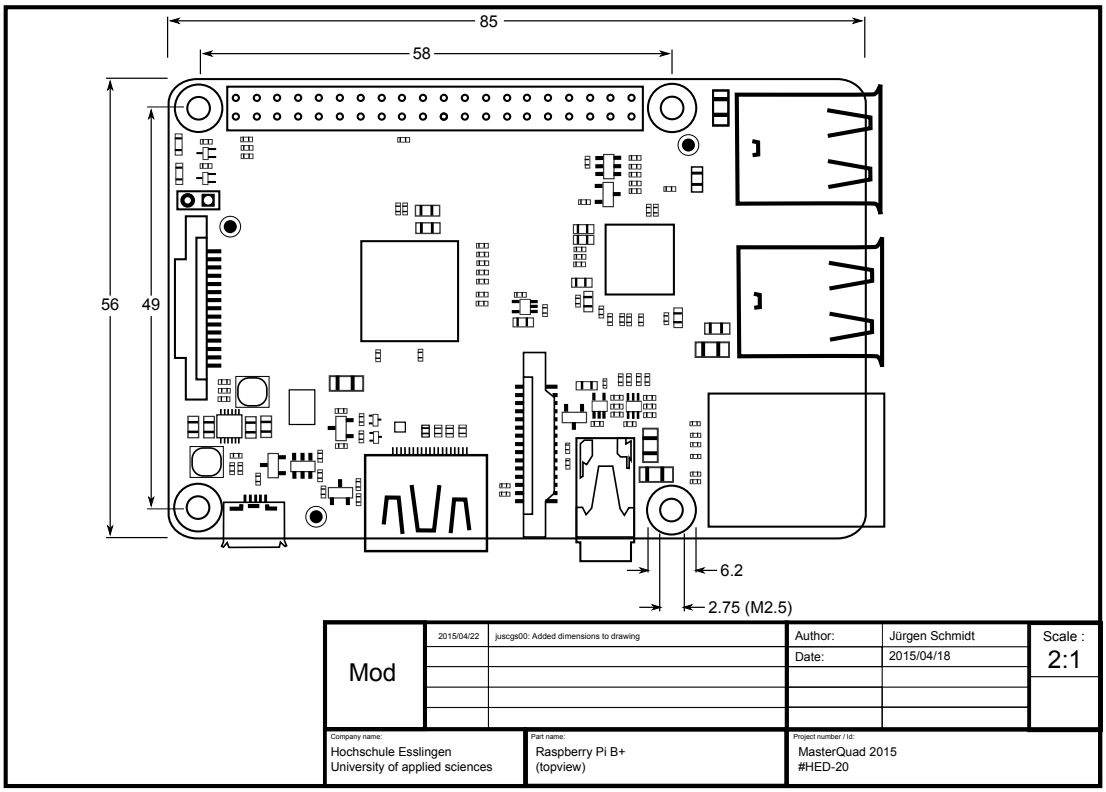


Abb. 2.1: Raspberry Pi B+ (topview)

2.2 Adafruit GPS Hat

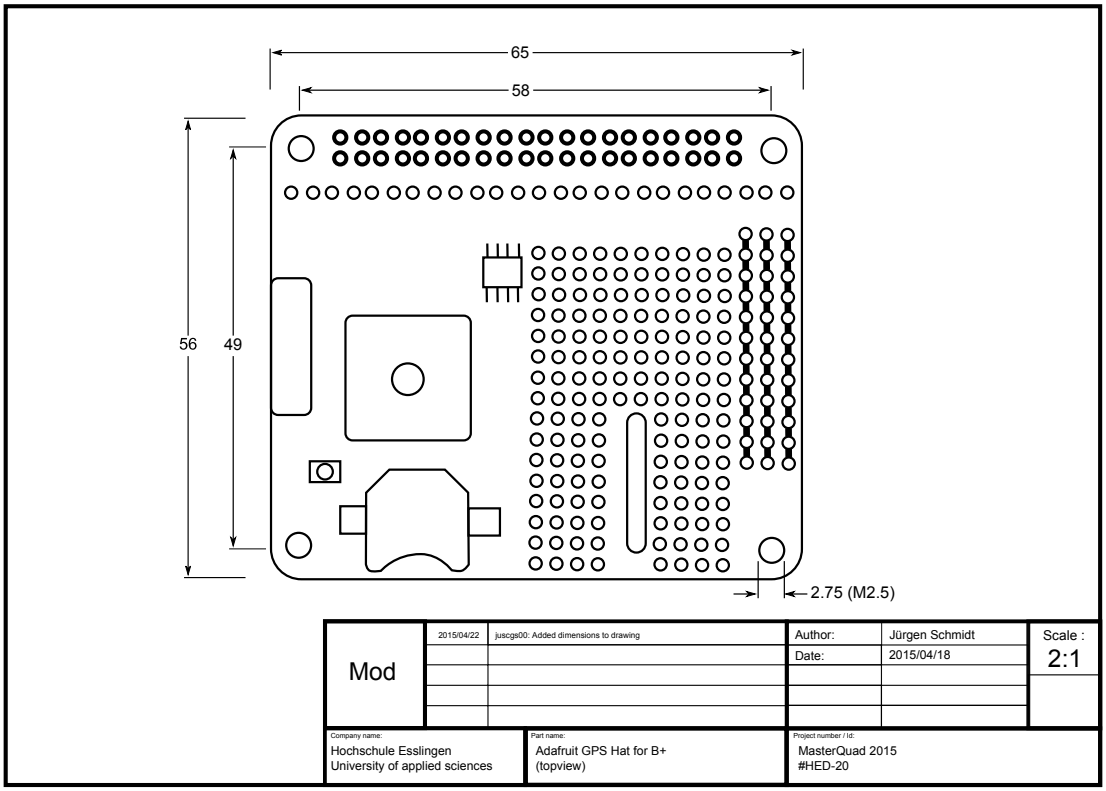


Abb. 2.2: Adafruit GPS Hat

2.3 Polulu AltIMU v4

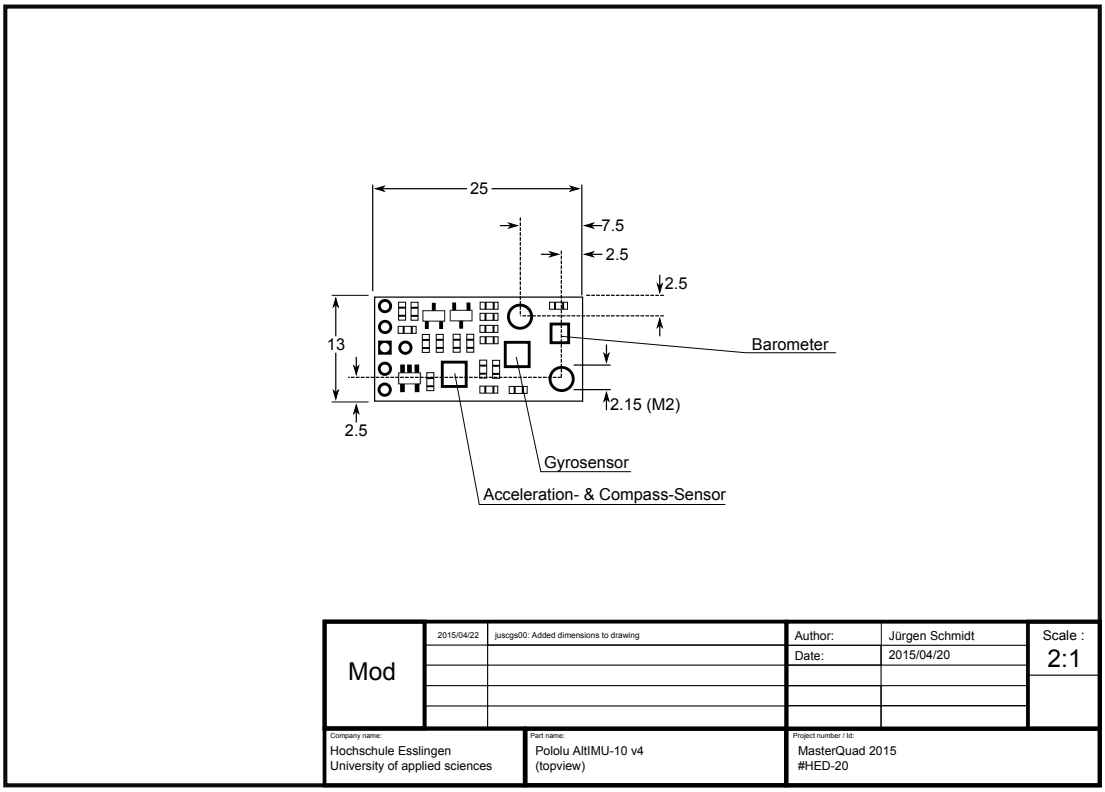


Abb. 2.3: Polulu AltIMU v4

2.4 Adafruit 12bit ADC over I2C

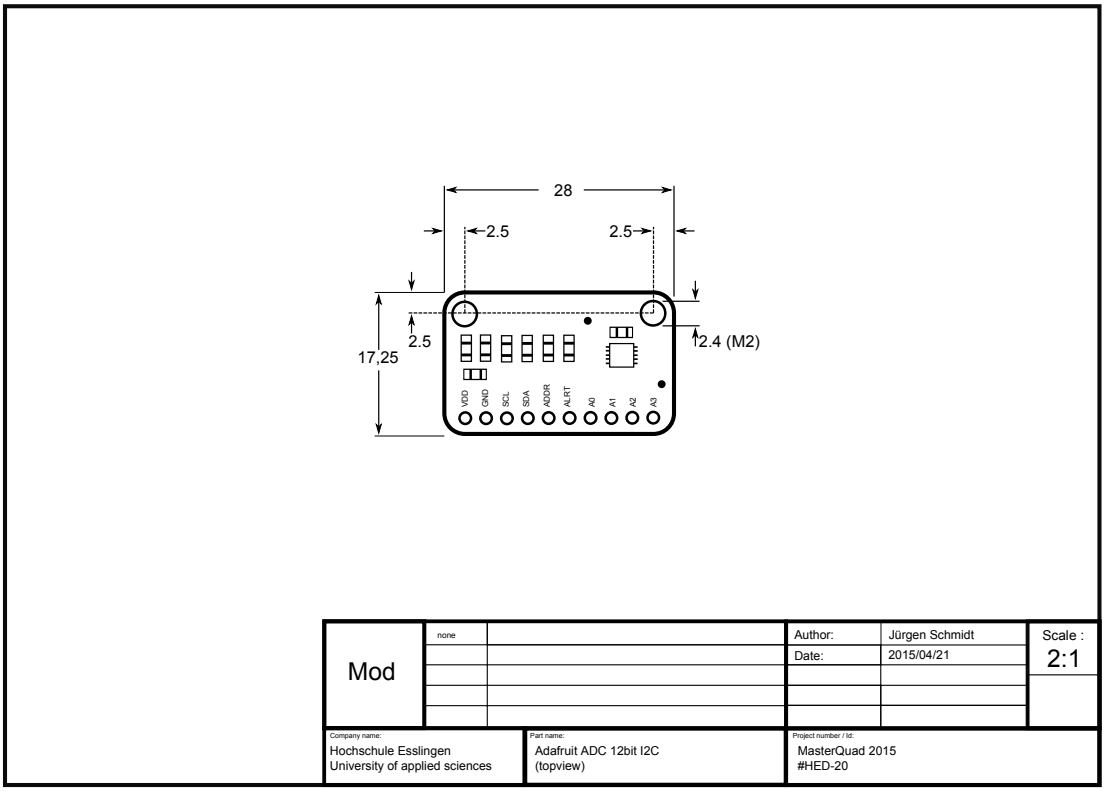


Abb. 2.4: Adafruit 12bit ADC over I2C

3 Assembly

3.1 Placing ADC and IMU on GPS Hat

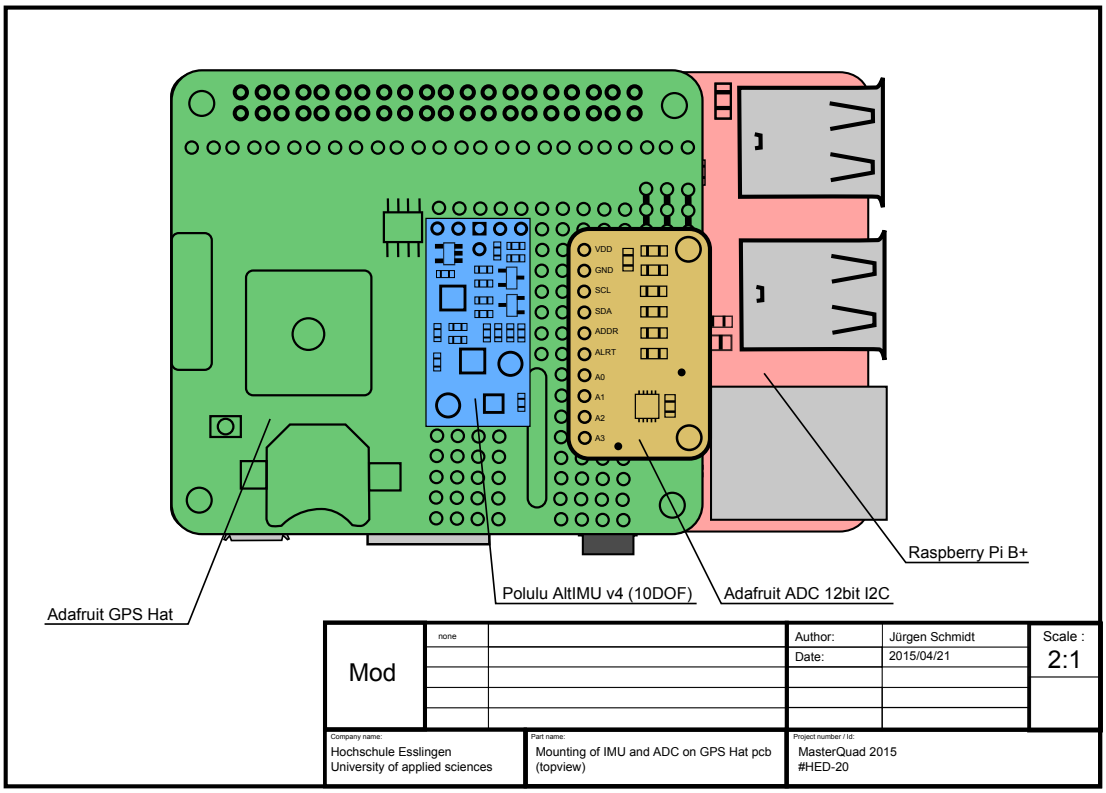


Abb. 3.1: Placing ADC and IMU on GPS Hat

3.2 Wedding of GPS Hat and Raspberry Pi B+

4 Software structure

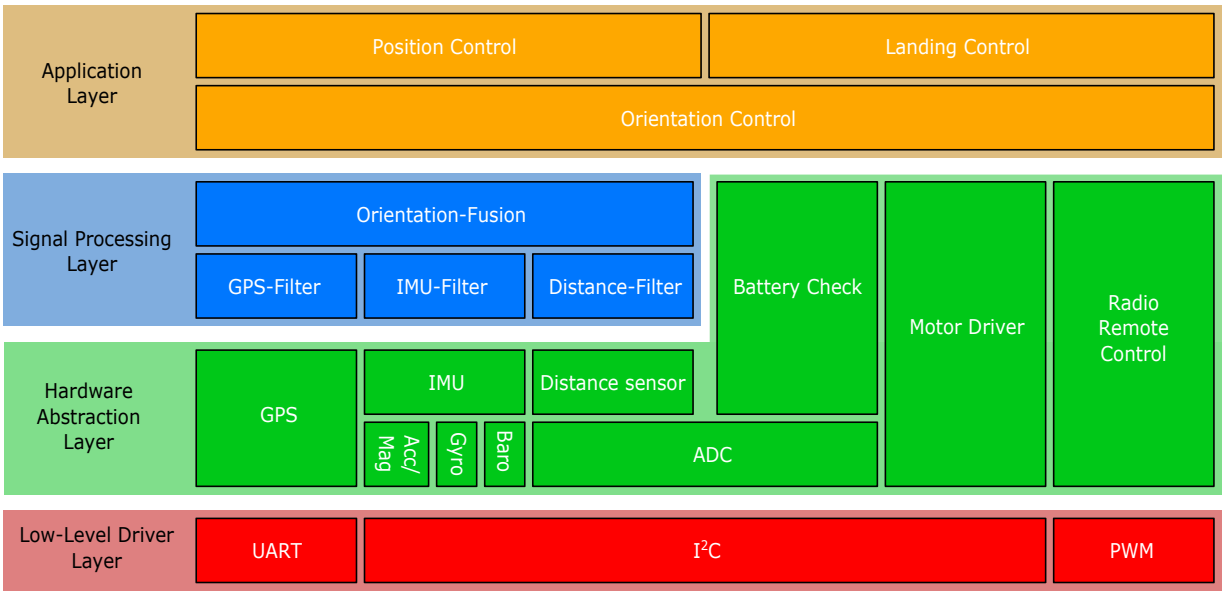


Abb. 4.1: Software layers with functional units of project MasterQuad 2015

5 Analysing Data and sensor fusion

5.1 Calculation of the orientation angles

The Kalman filter and the Complementary filter both need angles as input. Those angles are calculated with the acceleration sensor and with the gyroscope. The gyroscope sensor is used as main input and is corrected by the acceleration/magnetic sensor. The reason why both are used is already described in the Sensor fusion chapter.
To achieve an angle from the gyroscope, the values from the sensor needs to be integrated. The following formulas show the integration part of the function of the gyroscope which runs on the Raspberry Pi. It is located in the folder 'sig/Orientientation/' in the file Orientation.c in the function 'm_sigOri_calcGyroAnglePerStep_st()'. This function calculates the angle which since the last call is made.

$$roll_{angle} = roll_{rate} * deltaT \tag{5.1}$$
$$pitch_{angle} = pitch_{rate} * deltaT \tag{5.2}$$
$$yaw_{angle} = yaw_{rate} * deltaT \tag{5.3}$$

To make this code independent from any time step, it calculates locally the time difference 'deltaT' between the last call. To make this possible the 'gettimeofday' function is used. Because of that any jitter will not make a problem. Also the gyroscope sensor gets automatically offset corrected when the code is started.

To get an angle from the acceleration/magnetic sensor the following calculation is needed. It is located in the function 'm_sigOri_calcAccMagAngle_st()' in the same file like mentioned before.
First the roll angle is calculated:

$$roll_{angle\_rad} = atan2\left(\frac{acc_{y\_axis}}{acc_{z\_axis}}\right) \tag{5.4}$$
$$roll_{angle\_deg} = -roll_{angle\_rad} * 180/Pi \tag{5.5}$$

Then the pitch is calculated:

$$pitch_{angle\_rad} = atan\left(-\frac{acc_{x\_axis}}{acc_{y\_axis} * sin(roll_{angle\_rad}) + acc_{z\_axis} * cos(roll_{angle\_rad})}\right) \tag{5.6}$$

$$pitch_{angle\_deg} = -pitch_{angle\_rad} * 180/Pi \tag{5.7}$$

Last step is the calculation of the yaw angle. The yaw angle can not be calculated by the acceleration sensor, only the magnetic sensor can be used for this. But the magnetic sensor can not directly be used. Additional a magnetic tilt compensation needs to be done. The following formulas show the calculation steps.

$$divider = mag_{x\_axis} * cos(pitch_{angle\_rad}) + \tag{5.8}$$

$$mag_{y\_axis} * sin(pitch_{angle\_rad}) * sin(roll_{angle\_rad}) + \tag{5.9}$$

$$mag_{z\_axis} * sin(pitch_{angle\_rad}) * cos(roll_{angle\_rad}) \tag{5.10}$$

$$yaw_{angle\_rad} = atan2\left(\frac{mag_{z\_axis} * sin(roll_{angle\_rad}) - mag_{y\_axis} * cos(roll_{angle\_rad})}{divider}\right) \tag{5.11}$$

$$yaw_{angle\_deg} = yaw_{angle\_rad} * 180/Pi \tag{5.12}$$

With this, the implementation of the fusion filter were done and proved. First the results seem to be correct like can be seen in figure 5.1.

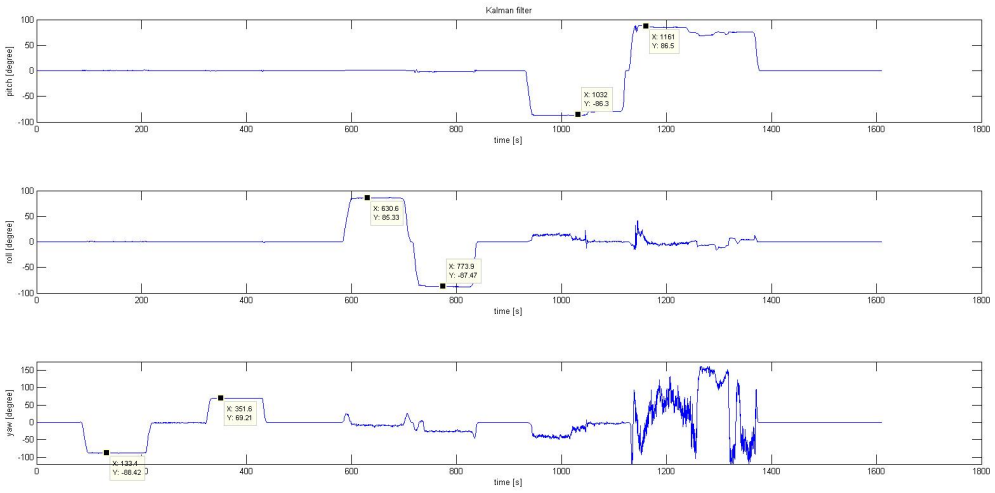


Abb. 5.1: First result Kalman filter

5.1.1 Magnetic sensor test

As can be seen there is a problem with the yaw angle when a pitch angle is applied. This is no solution which can be used in the Quadrocopter. So additional tests are made. After some time the problem was detected. Because just the yaw angle makes some problem, the main observation layed on the magnetic sensor. Figure 5.2 shows the measured magnetic field.

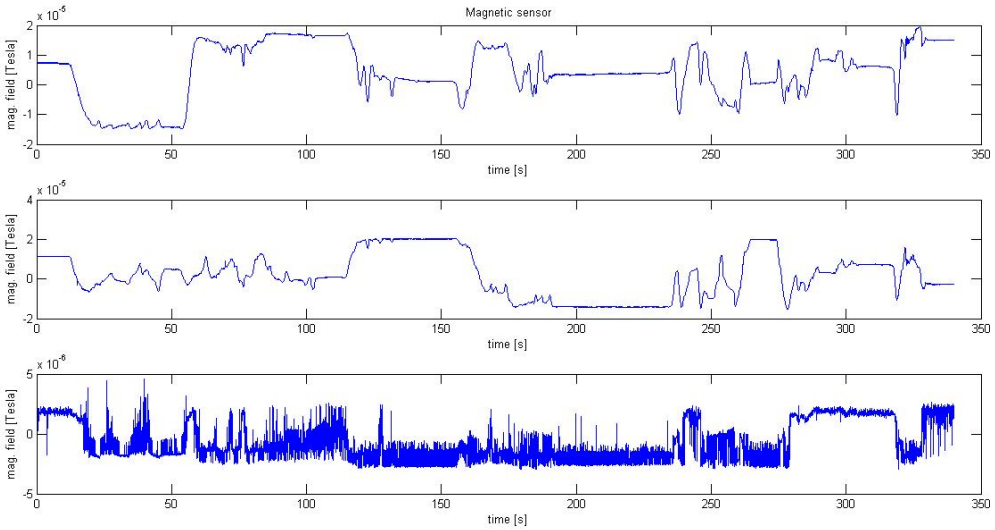


Abb. 5.2: Weak magnetic field strength

The problem can directly be seen. The measured field strength should be nearly the same on all axis. The field strength on the x-axis and y-axis is the same. The z-axis delivers only just one tenth of the normal value. Because of that the noise on the sensor is in the same height like the signal and therefore acquiring not possible. First the position of the mounted IMU is disputed. After changing the following sensor values of the magnetic sensor can be achieved.

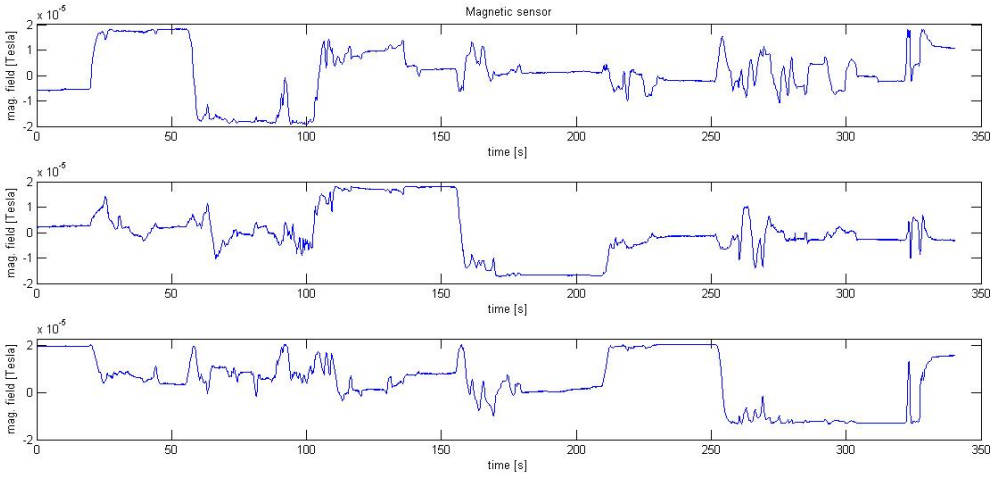


Abb. 5.3: Strong magnetic field strength

By comparing the measurement of the initial positioning of the IMU with the new one, the strength on the z-axis is significantly higher. Also the strength on the three axis are nearly the same. So for further usage the new position is used.

5.1.2 Improvement magnetic sensor

The figures 5.4 and 5.5 shows the comparison of the mounting.

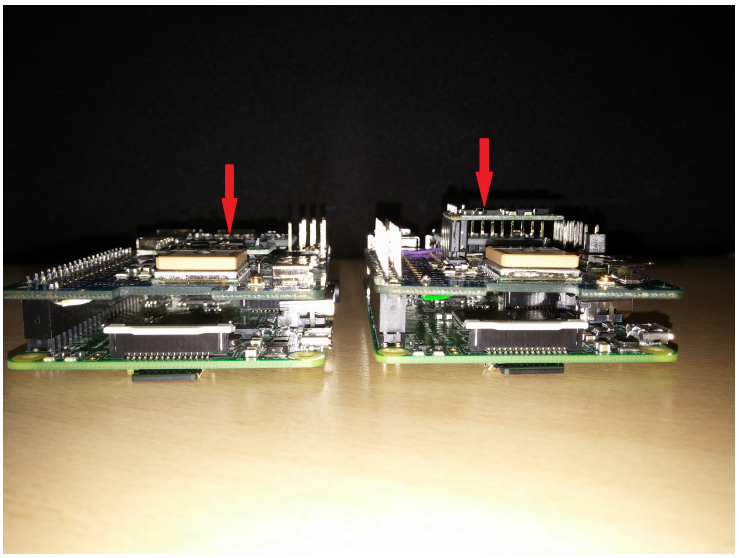


Abb. 5.4: New positioning of IMU 1

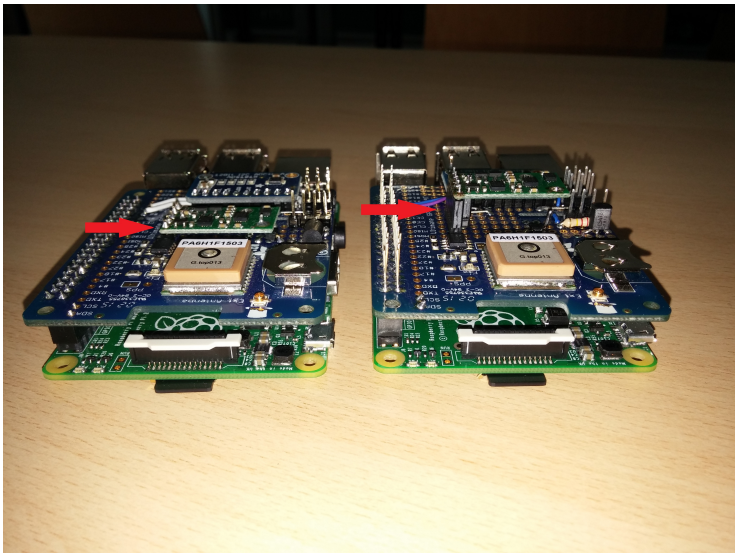


Abb. 5.5: New positioning of IMU 2

The left PCB shows the old position and the right the new position. As can be seen just the height of the mounted IMU needs to be changed. Due to the tests a distance of approximately 1cm seems to be enough. So the wiring can be kept as it is.
Additionally to reduce the really high influences of near metallic parts, a hard magnetic offset compensation needs to be done. Also a scaling is done. The following calculations which run in the function 'm_sigOri_calcAccMagAngle_st()' are done within every step.

$$mag_{x\_axis} = (mag_{x\_axis} - logged_{min\_x})/(logged_{max\_x} - logged_{min\_x}) * 2 - 1 \quad (5.13)$$
$$mag_{y\_axis} = (mag_{y\_axis} - logged_{min\_y})/(logged_{min\_y} - logged_{min\_y}) * 2 - 1 \quad (5.14)$$
$$mag_{z\_axis} = (mag_{z\_axis} - logged_{min\_z})/(logged_{min\_z} - logged_{min\_z}) * 2 - 1 \quad (5.15)$$

To achieve the full range, maximum and minimum value of all three magnetic sensors the scope of Matlab can be used. First the data transmission between Matlab and the Raspberry Pi needs to be started. Then the scopes of the magnetic sensor should be opened. Now the Raspberry Pi should be rotated by parallel trying to find the maximum and minimum with the opened scopes. The m-file which is mentioned before creates directly the header-file for the code. The improved results due to the changes which are made here can be seen in the second output of the Kalman filter and complementary filter in figure 5.11 and 5.18.

5.2 Sensor fusion for Inertial Measurement Unit

To use all positive features of the sensors and reduce the negative drawbacks a sensor fusion is the needed solution. There are many possibilities for fusion algorithms. In this

project a Complementary-Filter and Kalman-Filter is implemented.

For enabling a autonomous flight, the Raspberry Pi has to know the orientation. Figure 5.6 shows the axes and the naming of the rotation around the axes. These rotations are later used for the calculation for the roll, pitch and yaw angles.

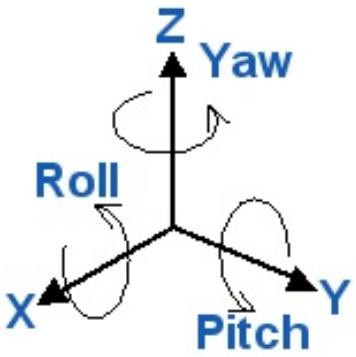


Abb. 5.6: Roll, Pitch, Yaw [BorEng]

To get reliable and stable orientation of the Raspberry Pi and so from the Quadrocopter several sensors can be used. Either a acceleration sensor, a magnetic sensor or a gyroscope can be used. But each of them have some drawbacks.
The acceleration sensor is very fast and delivers reliable pitch and roll angles, but the yaw angle itself can't be calculated. Another problem is, that due to vibrations a smooth angle is not possible to calculate.

The magnetometer can be used to calculate the heading, this means the yaw angle but not the roll and pitch angle. Another problem when the sensor has a roll and pitch angle, the heading can not be easily calculated. In this case a tilt compensation has to be done. In the figure 5.7 the logging of the pitch and roll angle calculated from the acceleration sensor and the yaw angle calculated with the magnetometer can be seen. When those sensors are not combined errors occur during the calculation. In the time from 250 seconds to 450 seconds a yaw angle is applied which leads just to a yaw angle change. In the time area from 500 seconds to 600 seconds a roll angle is applied which also leads to a yaw change which is an error. In the time area from 700 seconds to 800 seconds a pitch angle is applied which also leads to a yaw change which is an error.

When designing a system using multiple MEMS sensors, it is important to understand the advantages and disadvantages of accelerometers, gyroscopes, magnetometers, and pressure sensors.

Sensor fusion solves key motion sensing performance issues of 6-axis modules consisting of a 3-axis accelerometer and a 3-axis gyroscope or a 3-axis accelerometer and a 3-axis magnetic sensor. 1) A 6-axis inertial module with an accelerometer and a gyroscope loses its absolute

orientation as the gyro drifts over time, requiring calibration to restore accurate heading reference. 2) A 6-axis module with accelerometer and magnetometer is prone to data corruption in the presence of ferrous materials in the environment. 3) A 9-axis module with an accelerometer, a gyroscope and a magnetometer eliminates the drift that occurs with stand-alone sensor solutions. But these can be subject to magnetic interference. Algorithms to fuse the sensor data are required to compensate for the magnetic interference.

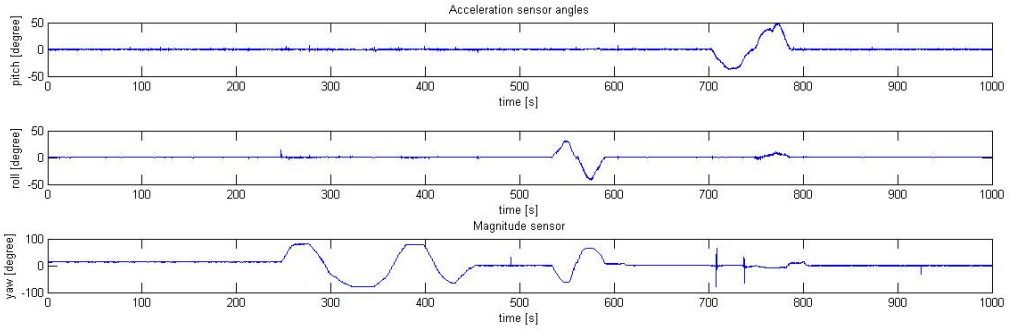


Abb. 5.7: Magnetic / Acceleration angles

The last sensor is the gyroscope. The angle can be easily calculated by integrating the rates of the gyroscope. The sensor is not as fast as the acceleration sensor. So vibrations make no problems for the calculation. But due to the problem of the offset of the gyroscope, the angles will be drifting because of the integration of the rotation rates. This can be seen in figure 5.8

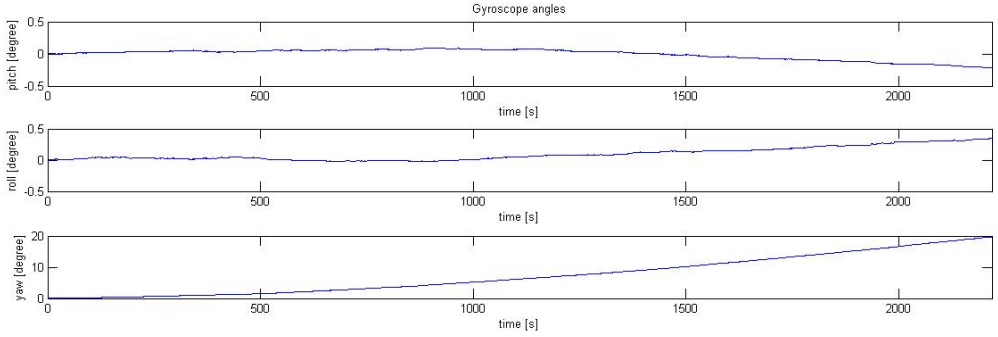


Abb. 5.8: Gyroscope angles

5.2.1 Complementary-Filter

The first approach for a sensor fusion is the complementary filter. This filter uses a high-pass filter for the gyroscopes and a lowpass filter for the acceleration sensor. The highpass filter is used after the integration of the rotation rates. This can be seen in figure 5.9.

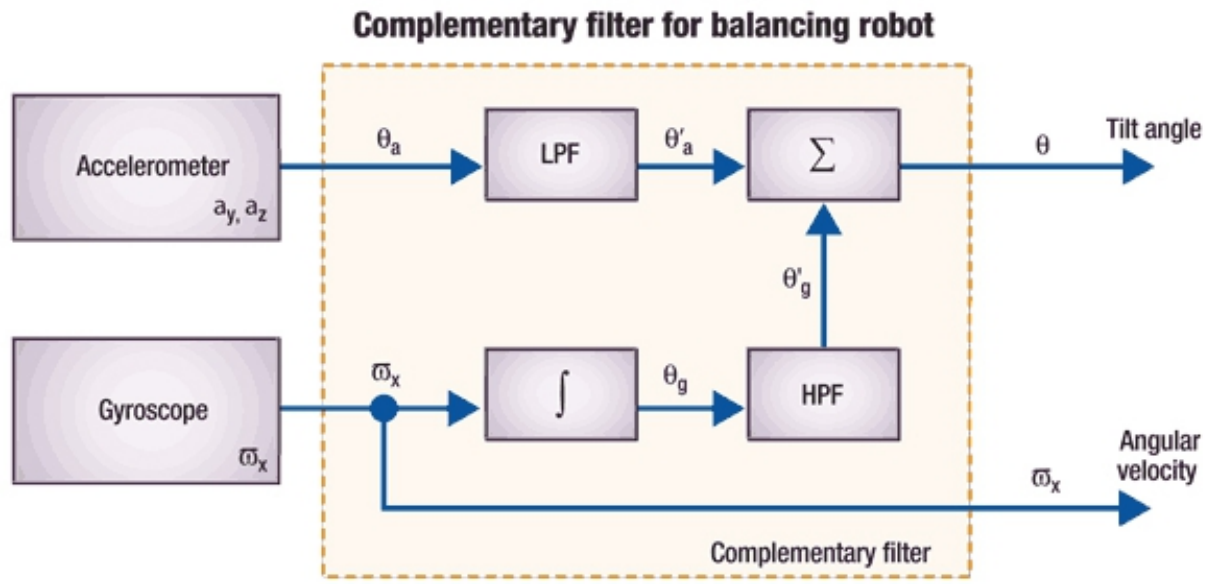


Abb. 5.9: Complementary-Filter[STM]

The implementation effort is extremely lower than with the Kalman Filter. Because here no matrices and matrix operations are needed. Also are there less calculations and so fits this algorithm better to microprocessors. The only thing what needs to be checked is the lowpass/highpass-filter coefficient. Next the needed calculations are mentioned to show the difference of the complementary and Kalman filter.

- Calculation of the accelerometer and magnetometer angles
- Calculation of the gyroscope angles
- Complementary-Filter usage with defined filter time of the lowpass and highpass-filter

As written in the chapter about the sensor fusion for the complementary filter a filter time needs to be chosen. This constant is used for the lowpass filter for the acceleration/magnetic sensor and for the highpass filter of the gyroscope.

The complementary filter uses the following calculation:

$$angle = alpha * (angle + integrated\_Gyro) + (1 - alpha) * Acc\_Mag\_angle \tag{5.16}$$

As an initial guess for the first try the complementary filter uses the filter constant of 0.995. The sampling period of the sensors is 800 Hz. Usage of the filter constant of 0.995 and the sampling period of 800 Hz leads to a cut off frequency of 4 Hz.

$$alpha = \frac{time\_constant}{time\_constant + sample\_period} \tag{5.17}$$

$$alpha = 0.995 \tag{5.18}$$

$$sample\_period = \frac{1}{800Hz} = 0.00125sec \tag{5.19}$$

$$\tag{5.20}$$

This leads to:

$$time\_constant = 0.2488sec \approx \frac{1}{4Hz} \tag{5.21}$$

With this constant the following result was achieved. To make a test, first the yaw angle is changed in the range of ±90°, after that a roll angle is changed in the range of ±90° and finally the pitch angle is changed in the range of ±90°.

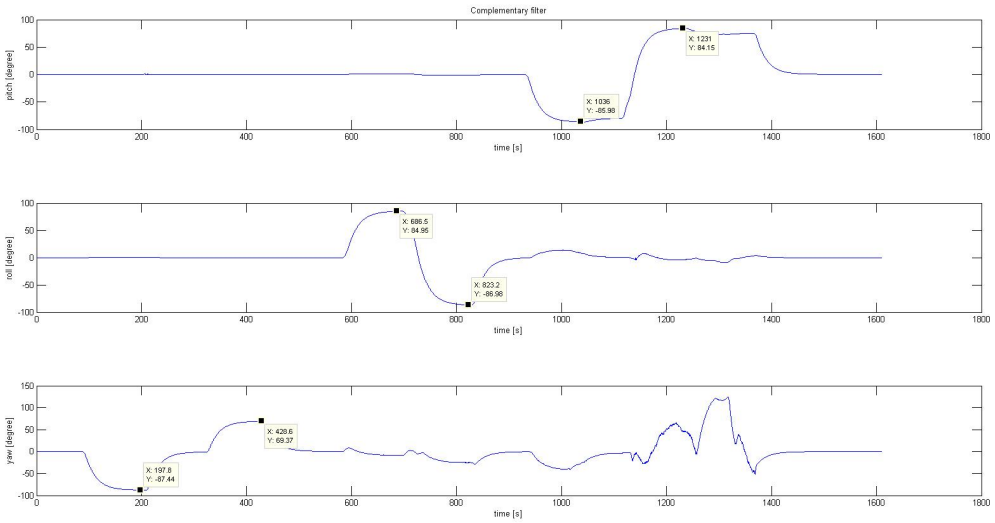


Abb. 5.10: First result complementary filter

First what catches somebody's eye is that the filter is to slow. So the steady state value is achieved after consuming to much time. Also the yaw angle is not equally distributed over the whole 360 degree. In one direction the angle changes just around 70 degree. When changing roll the influences in yaw is just because the rotation was not straight in roll direction. Changing the pitch angle to much lead to a change in yaw. The next step is to improve the yaw angle, so that it will reach also the +90 degree. Also the yaw change during pitch will be observed and compared to the initial measurement. Also the time constant will be changed that a faster response can be seen.

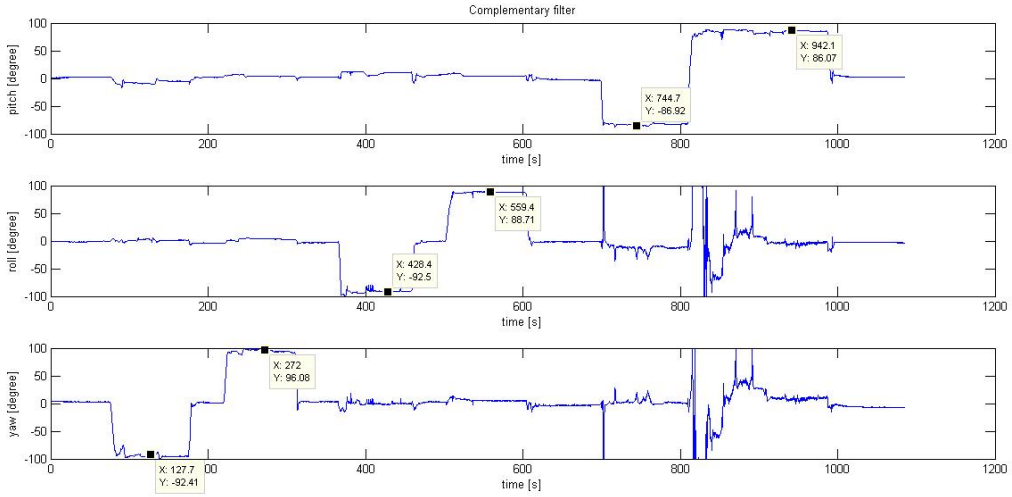


Abb. 5.11: Final result complementary filter

The comparison of the first result and the final result shows the increased filter speed. The signal needs not so much time to reach the final value. Also the yaw angle reaches the 90 degree when turning 90 degree. The extreme influences on yaw directly after changing of the rotation angle results from a hand made rotation. The influences on yaw while an other angle is applied is extremely reduced.

The last figure shows how off an angle can be when just an gyroscope is used. On the left side the integrated gyroscope can be seen. On the right side, the 3D representation of the fusioned sensors are displayed.

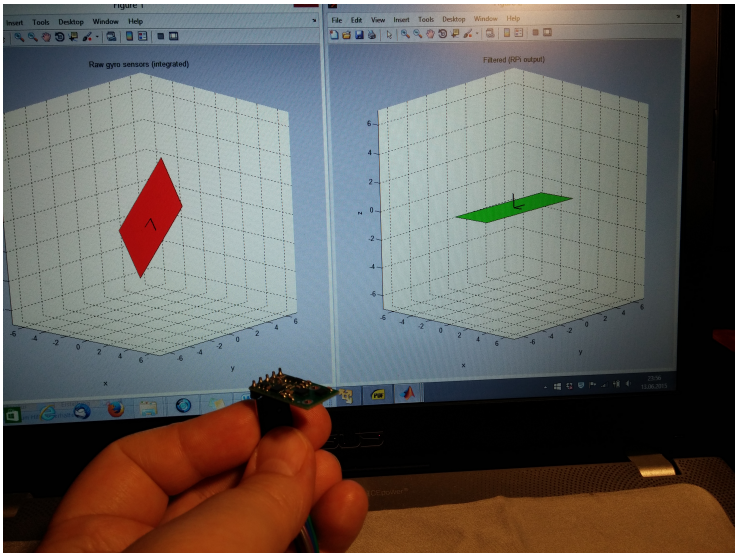


Abb. 5.12: 3D representation of the integrated gyroscope raw values on the left and the fusion filtered on the right

5.2.2 Kalman-Filter

The second approach for a sensor fusion is the Kalman filter. This filter is based on the state space modeling where between the dynamics of the system and the process of the measurement is differentiated. Although the measuring is faulty and the system state is noisy, this filter guesses with the help of a set of equations the correct true state of the system. Instead of using the absolute value it uses the mean value and variance of the normal distribution. The mean value is the perfect measurement and the variance tells the uncertainty of the measurement.

The following state space description shows the calculation of the new states which in the case of this project represent the angles pitch, roll and yaw. Equation 5.22 shows the defined state vector.

$$\vec{x} = \begin{pmatrix} pitch \\ roll \\ yaw \end{pmatrix} \tag{5.22}$$

The calculation of a new state and the output can be seen in the formulas 5.23 and 5.24.

$$\vec{x_k} = \underline{A}\vec{x}_{k-1} + \underline{B}\vec{u}_{k-1} + W_{k-1} \tag{5.23}$$

$$\vec{y}_k = \underline{H}\vec{x}_k + V_k \tag{5.24}$$

Legend of the formula shown above:

- $\vec{x}_k$ state vector of actual step
- $\vec{x}_{k-1}$ state vector of previous step
- $\underline{A}$ system matrix
- $\underline{B}$ input matrix
- $\vec{u}_{k-1}$ input vector of previous step
- $\underline{W}$ process noise
- $\vec{y}_k$ output vector
- $\underline{H}_k$ output matrix
- $\underline{V}$ measurement noise
- $\underline{P}$ Output Covariance
- $\underline{Q}$ Process Noise Covariance
- $\underline{R}$ Measurement Noise Covariance

Because of the measurement and the process noise the new state is not good. The Kalman filter takes the process and measurement noise into account to improve the state estimation. To do so the Kalman filter is split into two parts, the prediction step (time update) and correction step (measurement update). In the prediction step the filter estimates the states in the next step and calculates the new covariance. Those values are calculated for the states which in the following step are expected to be reached. In the next step, the correction update, the filter checks if the pre-calculated state is reached. According to the difference the correction for the following prediction is done.

Prediction step:

Predict the next state:

$$\vec{x}_k = \underline{A}\vec{x}_{k-1} + \underline{B}\vec{u}_{k-1} \tag{5.25}$$

Predict the covariance for the next step:

$$\underline{P}_k = \underline{A}\underline{P}_{k-1}\underline{A}^T + \underline{Q} \tag{5.26}$$

Correction step:

Computation of the Kalman gain:

$$\underline{K}_k = \underline{P}_k\underline{H}^T(\underline{H}\underline{P}_k\underline{H}^T + \underline{R})^{-1} \tag{5.27}$$

Updating state prediction with new measurement:

$$\vec{x}_k = \vec{x}_k + \underline{K}_k(\vec{z}_k - \underline{H}\vec{x}_k) \tag{5.28}$$

Updating the error covariance:

$$\underline{P}_k = (\underline{I} - \underline{K}_k\underline{H})\underline{P}_k \tag{5.29}$$

These steps have to be done more than just ones, so when the correction is finished, the update has to be called again and so on.
As can be seen the implementation effort is higher than with the Complementary Filter. Because matrices and matrix operations are needed, more calculation steps are needed and the functionality of the Kalman filter is not as intuitive like with the complementary filter. Also the uncertainty of the process itself and the measurement error needs to be known. Nevertheless the results from the Kalman filter are better than those of the complementary filter. This can be seen in the results showing chapter of this project.

Figure 5.13 shows a measurement of an acceleration sensor.

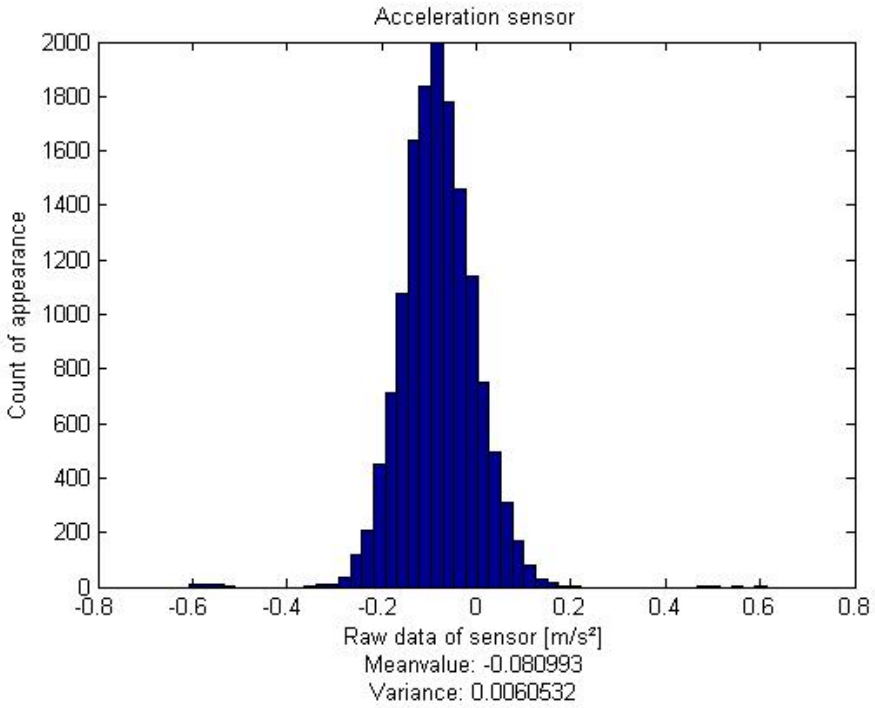


Abb. 5.13: Variance and Meanvalue of an acceleration sensor

For the Kalman filter the noise of the measurement and the process has to be set up. This noise values are the variances of the used sensors and of the process. In the calculation process are those matrices named $\underline{Q}$ and $\underline{R}$. The matrix $\underline{Q}$ uses the variances of the process noise and the matrix $\underline{R}$ uses the variances of the noise of the sensors.

As an initial guess the matrices are set to:

$$\underline{Q} = \begin{pmatrix} 0.005 & 0 & 0 \\ 0 & 0.005 & 0 \\ 0 & 0 & 0.0001 \end{pmatrix} \tag{5.30}$$

$$\underline{R} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.01 \end{pmatrix} \tag{5.31}$$

With those matrices the following result was achieved. To make a test, first the yaw angle is changed in the range of ±90°, after that a roll angle is changed in the range of ±90° and finally the pitch angle is changed in the range of ±90°.

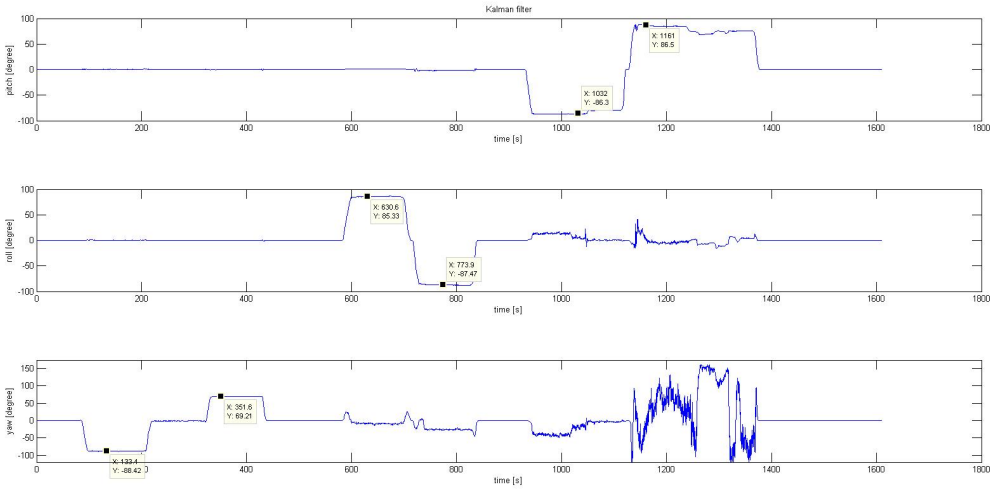


Abb. 5.14: First result Kalman filter

The filter responses in a adequate time. Yaw angle change is like with the complementary filter from -90 degree to just 70 degree. The roll angle changes sufficient. But when changing the pitch angle to high a extreme yaw angle change can be seen. So the next step is to improve the yaw angle, so that it will reach also the +90 degree. Also the yaw change during pitch will be observed. After measuring the mean value and variances of all sensors they can be used to improve the Kalman filter. The figures 5.15, 5.16 and 5.17 show the logged data.

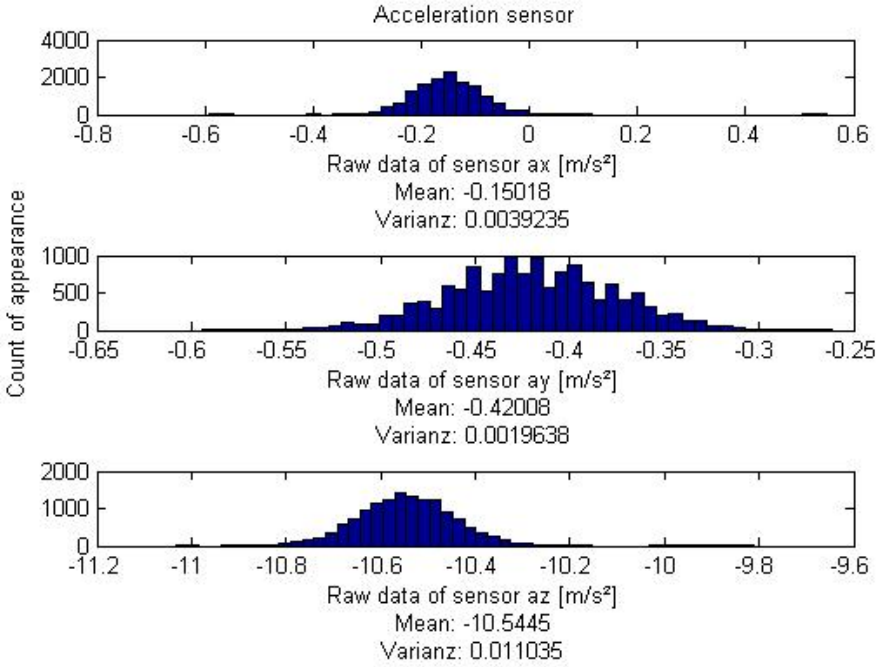


Abb. 5.15: Analyzing the acceleration sensor

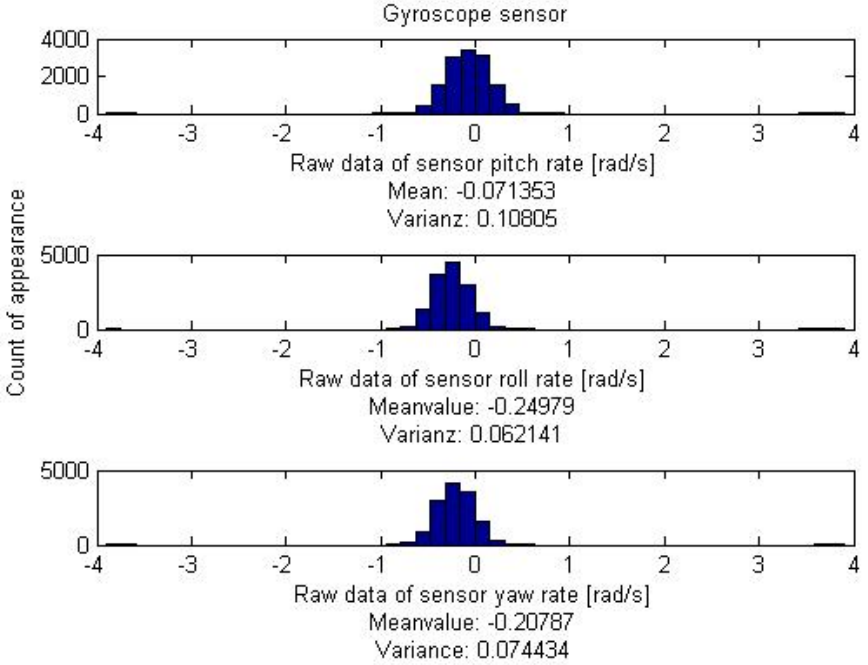


Abb. 5.16: Analyzing the gyroscope sensor

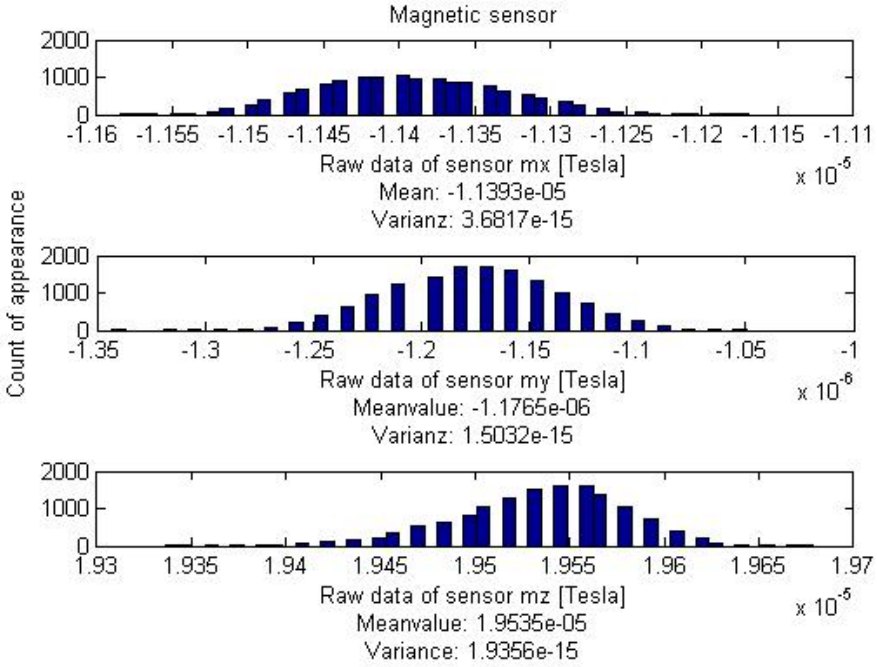


Abb. 5.17: Analyzing the magnetic sensor

First the matrix $\underline{Q}$, the variances of the process noise and the matrix $\underline{R}$, the variances of the measured sensor noises are changed by using the measured values. The new values are:

$$\underline{Q} = \begin{pmatrix} 0.005 & 0 & 0 \\ 0 & 0.005 & 0 \\ 0 & 0 & 0.005 \end{pmatrix} \tag{5.32}$$

$$\underline{R} = \begin{pmatrix} 0.06 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.07 \end{pmatrix} \tag{5.33}$$

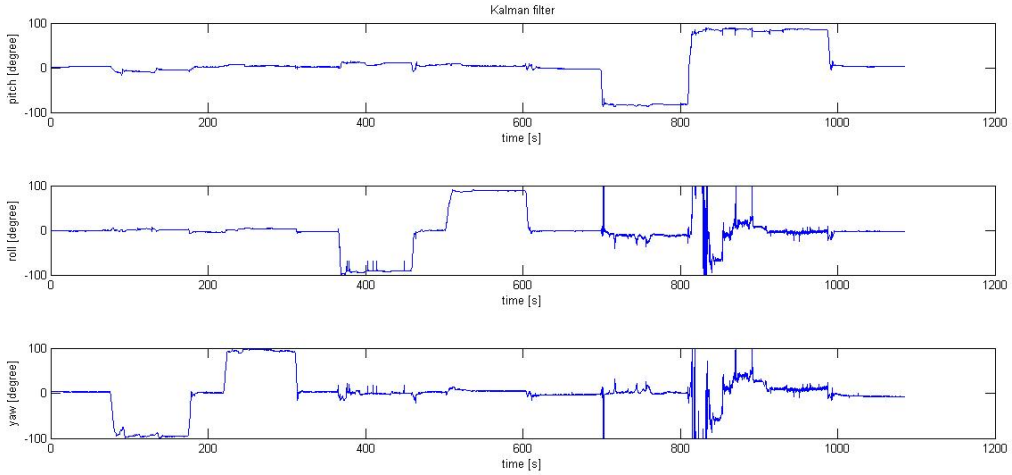


Abb. 5.18: Final result Kalman filter

The wanted improving of the yaw angle in reaching the 90 degree when turning 90 degree is successfully reached. The extreme influences on yaw directly after changing of the rotation angle results from a hand made rotation. The influences on yaw while an other angle is applied is extremely reduced.

The last figure shows how off an angle can be when just an gyroscope is used. On the left side the integrated gyroscope can be seen. On the right side, the 3D representation of the fusioned sensors are displayed.

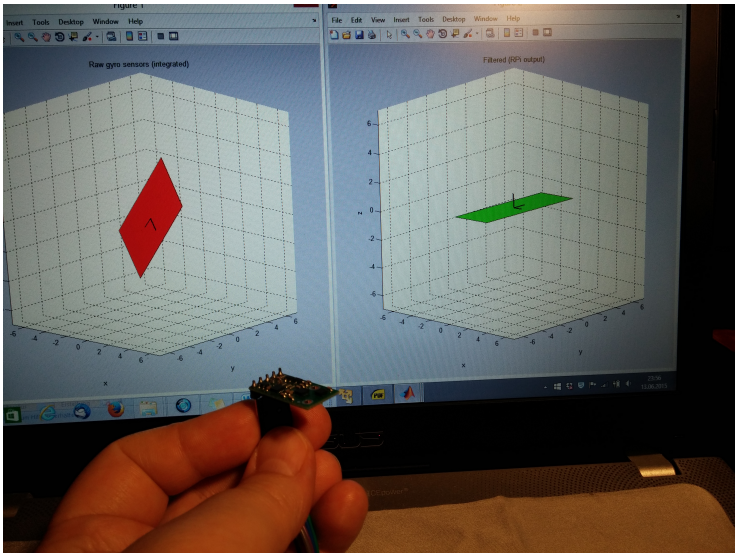


Abb. 5.19: 3D representation of the integrated gyroscope raw values on the left and the fusion filtered on the right

5.3 Matrix library

Like already mentioned in the Kalman chapter various matrix operations are needed. So the simple assignment from a matrix to another, the addition and subtraction of a matrix from another. Additionally the multiplication of two matrices, the initialization of a matrix and building of the identity matrix. Finally the transpose of a matrix and the inversion of a matrix.

Assignment:

$$\underline{A} = \underline{B} \tag{5.34}$$
$$A(0,0) = B(0,0) \tag{5.35}$$
$$... \tag{5.36}$$
$$A(n,m) = B(n,m) \tag{5.37}$$

Addition/Subtraction:

$$\underline{A} = \underline{B} \pm \underline{C} \tag{5.38}$$
$$A(0,0) = B(0,0) \pm C(0,0) \tag{5.39}$$
$$... \tag{5.40}$$
$$A(n,m) = B(n,m) \pm C(n,m) \tag{5.41}$$

Multiplication:

$$\underline{A} = \underline{B} \cdot \underline{C} \tag{5.42}$$
$$A(0,0) = B(0,0) \cdot C(0,0) + ... + B(0,m) \cdot C(n,0) \tag{5.43}$$
$$... \tag{5.44}$$
$$A(n,m) = B(n,0) \cdot C(0,m) + ... + B(n,m) \cdot C(n,m) \tag{5.45}$$

Initialization:

$$A(0,0) = xx.xx \tag{5.46}$$
$$A(0,1) = xx.xx \tag{5.47}$$
$$... \tag{5.48}$$
$$A(1,0) = xx.xx \tag{5.49}$$
$$... \tag{5.50}$$
$$A(n,m) = xx.xx \tag{5.51}$$

Identity matrix:

$$A(0,0) = 1 \tag{5.52}$$
$$A(0,1) = 0 \tag{5.53}$$
$$... \tag{5.54}$$
$$A(1,0) = 0 \tag{5.55}$$
$$A(1,1) = 1 \tag{5.56}$$
$$... \tag{5.57}$$
$$A(n,n) = 1 \tag{5.58}$$

Transpose a matrix:

$$A(0,0) = A(0,0) \tag{5.59}$$
$$A(0,1) = A(1,0) \tag{5.60}$$
$$... \tag{5.61}$$
$$A(3,0) = A(0,3) \tag{5.62}$$
$$A(4,0) = A(0,4) \tag{5.63}$$
$$... \tag{5.64}$$
$$A(n,n) = A(n,n) \tag{5.65}$$

Invert a matrix:
To ensure that the inversion not only works with small matrices, the implementation uses a different way. So first the Cholesky method is used to build a lower triangular matrix. Next step is solving a linear system to get the inverse of the matrix. With this method all sizes of matrices can be inverted. It just has to be positive definite.

5.4 Sensor fusion controlling

To obtain the statistical data of all sensors of the inertial measurement unit a Matlab model is used. This model also helps to check the results of the two different fusion filters, the Kalman filter and the complementary filter. The data is send via UDP-packets from the Raspberry Pi to the Matlab model on the Host computer. Figure 5.20 shows the used Matlab model. Additional a 3D representation of the rotations is visualized and can be directly compared with the integrated offset corrected gyroscope data.

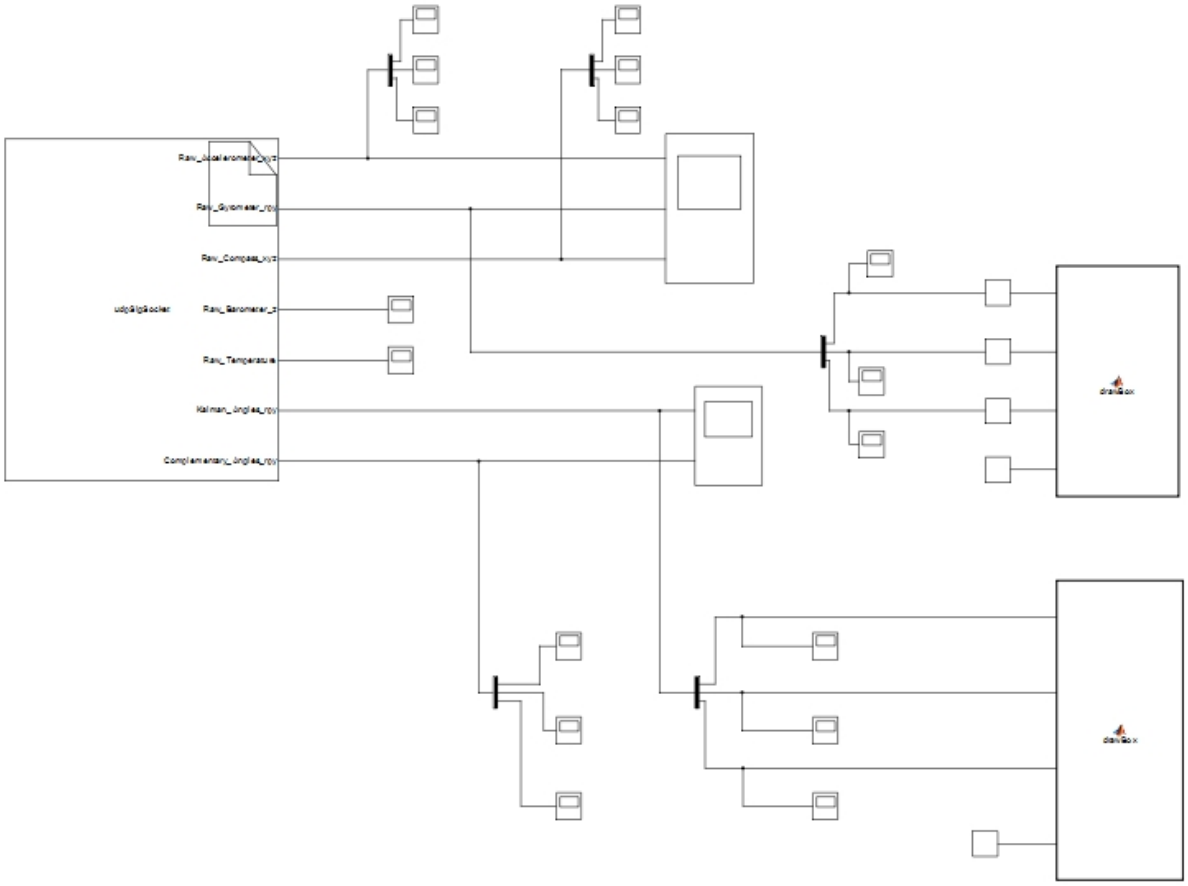


Abb. 5.20: Matlab model

With the help of the scopes within this model, the logged data is stored in the Matlab workspace and then it is analyzed with a Matlab file. With the help of this file, a histogram from every sensor is generated. Also the mean value and the variance from those sensor is calculated. Finally a plot from the complementary filter and Kalman filter is generated. Also the needed minimum and maximum values of the magnetic sensor is stored. This needs to be done to reduce the influences of hard magnetic parts to the magnetic sensor. Those stored minimum and maximum values are then written in a header file (Orientation-Defines.h) which needs to be copied to the folder of the Orientation.c and Orientation.h files.

6 Remote Control Unit: PPM Decoding

6.1 Graupner PPM sum signal in a nutshell

The HElikopter Team of Hochschule Esslingen uses the Graupner radio remote control system to control the Quadrocopters. Therefore, it is necessary to decode the Graupner PPM Signal delivered by the radio receiver to give a user's input to the controller platform of the Quadrocopters.

The Graupner remote control is a multi-channel system that comprises the transmission of up to 12 channels. Each individual channel is encoded by a PWM signal with a periode time of 20ms (resulting in a update rate of 50Hz) as depicted in fig. 6.1. According to the transmitted value of each channel, the on-time T_{on} varies between 1ms (min. value) and 2ms (max. value).

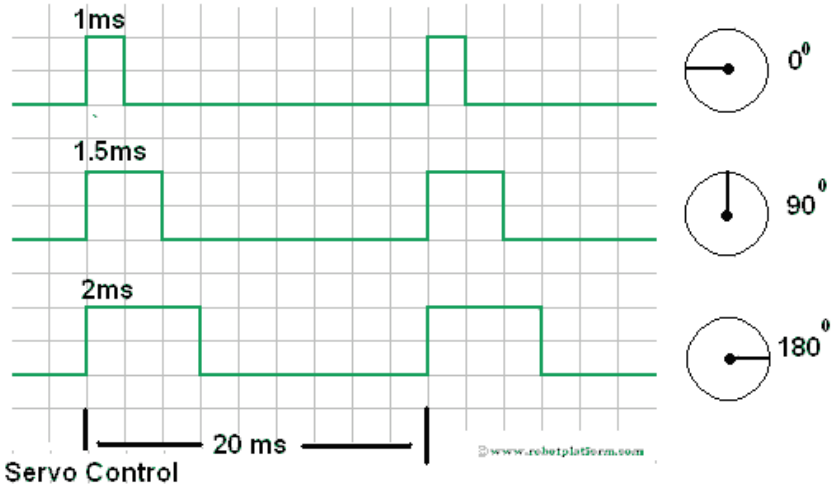


Abb. 6.1: PWM signal to control a RC servo [RPL]

In order to transmit several (up to 12) PWM channels simultaneously, all channels are combined sequentially in one sum signal as depicted in fig. 6.2. This results in a PPM signal with a fixed pulse width. The time between the rising edges of two individual pulses encodes the on-time of the PWM signal of the corresponding channel. The remaining pause gap at the end of a PPM frame is used to resynchronize the transmission. Without this re-synchronization, it would be impossible to map from a received pulse to the corresponding channel number.

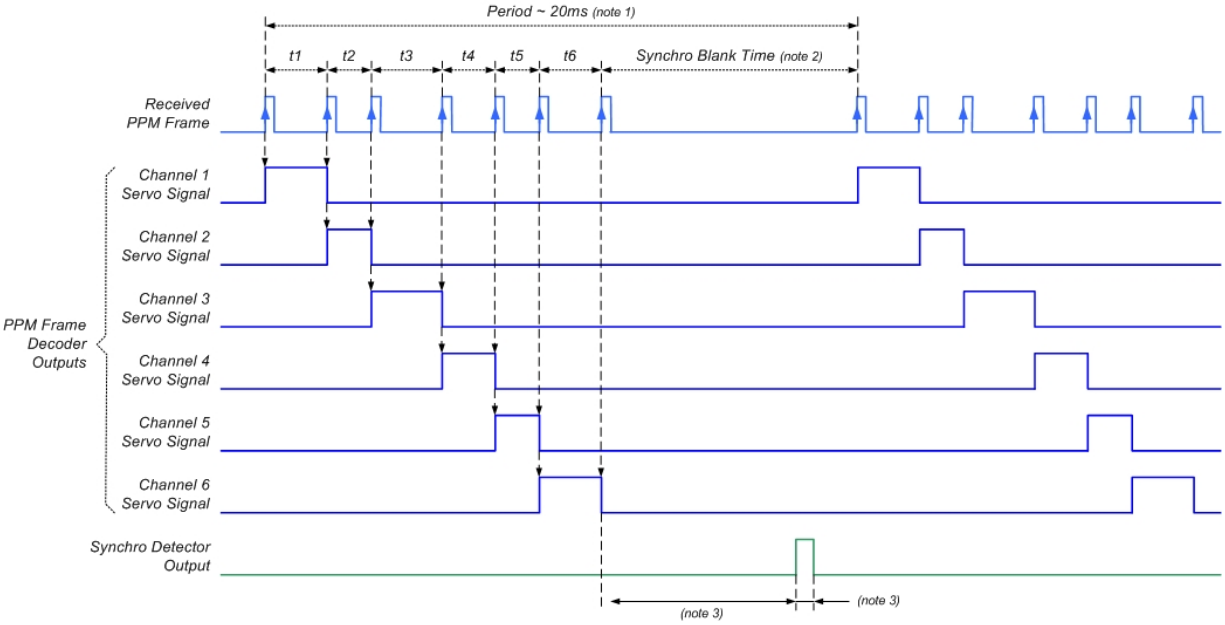


Abb. 6.2: PPM sum signal scheme of an exemplary 6 channel remote control [REP]

Some Graupner systems have the same sum signal encoding, but with a inverted out signal (high-state becomes low-state, and vice-versa). But if triggered on rising or falling edges only, this fact will not harm the general encoding/decoding approach.

6.2 Building a custom kernel driver

To build a custom kernel driver it is mandatory to have all Linux kernel sources on your development machine. If this is not already the case, please see chapter **??** on how to get the latest kernel sources and how to build a custom kernel.

Furthermore, it is crucial that the kernel sources have the same version number as the kernel installed on the Raspberry Pi's firmware! Due to that fact, it is recommended to perform a kernel update and subsequently compile the custom kernel driver.

Below in listing 6.1, a generic `Makefile` is shown to cross-compile a custom kernel driver for the Raspberry Pi. The variable `KDIR` in line 7 specifies the path to the kernel sources. The variable `CROSS_COMPILE` in line 6 specifies the path to the raspberry pi cross-compile tools. Depending on the distribution and development environment, these paths may have to be adapted.

In line 11 of listing 6.1 the object file (with file extension `.o`) of the compilation process that shall be linked as a kernel module file (file extension `.ko`) gets defined. If another kernel driver file shall be compiled, put the c-code file name there and change the file extension to `.o`.

```
1  ARCH := arm
2  UNAME := $(shell uname -m)
3  ifeq ($(UNAME),armv6l)
4    KDIR := /usr/src/linux/
5  else
6    CROSS_COMPILE:=/home/user/rpi/tools/arm-bcm2708/gcc-linaro-arm-linux-
       gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-
7    KDIR := /home/user/rpi/linux/
8  endif
9
10 ifneq ($(KERNELRELEASE),)
11   obj-m := ppmDemux.o
12 else
13
14 default:
15   $(MAKE) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) -C $(KDIR) M=$(PWD)
       modules
16 endif
17
18 clean:
19   rm -rf *.ko *.o .*.cmd .tmp_versions Module.symvers
20   rm -rf modules.order *.mod.c
```

List. 6.1: Simple Makefile to compile a custom kernel driver (here for the kernel module `ppmDemux`)

In the case of the `ppmDemux` kernel driver, a free running 64bit counter (called `System Timer`[1]) of the Raspberry Pi's System on Chip (SoC) has been used to determine a precise timing. This free running counter runs on $1Mhz$ independently of all other peripherals. This results in a maximum accuracy of $1\mu s$ of all measurements that utilize the `System Timer`.

The hardware address of the `System Timer` is defined by the base address `0x7E003000` plus the offset of the required data register (see [BCM], p. 175 ll.). Due to the `MMU` of the BCM2835 chip, the physical hardware address gets translated to the I/O base address `0x20003000` (as defined in line 27 of listing **??**). This address has to be used in the kernel module driver to access the free running $1Mhz$ `System Timer`.

Based on the articles [LMK70] and [LMK81], a kernel driver has been developed, to process a hardware trigger on GPIO pin 24. Whenever a rising edge is detected, a hardware interrupt is issued to the Kernel. The interrupt behavior in rising edges is defined in line 96 of listing **??**. The Interrupt Service Routine (ISR) is called in consequence (defined as function `hard_isr` in line 60 ll. of listing **??**).

```
60 static irqreturn_t hard_isr( int irq, void *dev_id )
61 {
62   /* read free running timer (64bit) @ 1MHz */
63   modIrqObj.timer_l = readl(timer_low);
```

1 See the peripherals datasheet on Raspberry Pi's SoC Broadcom BCM2835, p. 172 ll.

```
64   modIrqObj.timer_h = readl(timer_high);
65
66   /* counts rising edges since last read */
67   modIrqObj.irqNum += 1;
68
69   return IRQ_WAKE_THREAD;
70 }
```

This Interrupt Service Routine is the first function called whenever a rising edge event on GPIO24 occurs. The value of the 64bit `System Timer` get read out and an interrupt counter gets increased. That way, even for the unprobable case that one interrupt gets unprocessed, the synchronization of the Graupner PPM sum signal can be guaranteed.

After the hard ISR[1] has been processed, the threaded interrupt handler `rpi_gpio_isr` (line 52 ll. of listing **??**) gets called. For complex kernel drivers, in this function can be implemented complex algorithms to process the captured signals. This function is threaded and can be prioritized by the kernel's scheduler to guarantee a minimal latency time of the system.

```
60 static irqreturn_t hard_isr( int irq, void *dev_id )
61 {
62   /* read free running timer (64bit) @ 1MHz */
63   modIrqObj.timer_l = readl(timer_low);
64   modIrqObj.timer_h = readl(timer_high);
65
66   /* counts rising edges since last read */
67   modIrqObj.irqNum += 1;
68
69   return IRQ_WAKE_THREAD;
70 }
```

When all statements of the soft (threaded) ISR `rpi_gpio_isr` has been processed, the read function of the kernel driver get waken up (see listing **??** in line 55 and line 118 for the counterpart in the read function).

```
52 static irqreturn_t rpi_gpio_isr( int irq, void *data )
53 {
54   /* wake up read function to retrieve new data */
55   wake_up( &sleeping_for_ir );
```

```
117  /* wait for an rising edge event (caught by ISR) */
118  wait_event_interruptible( sleeping_for_ir, modIrqObj.irqNum);
```

1 Code inside of the hard ISR should be reduced to the absolute minimum and only necessary tasks should be performed. Every unnecessary CPU cycle in this section will slow down the system performance dramatically!

The complete code of the `ppmDemux` kernel module driver can be seen in listing **??** and in the SVN repository in `/impl/trunk/kern/`.

In order to compile the kernel driver, navigate to folder where the c code files and the `Makefile` are located and execute the following command in the bash command line:

```
cd ../path/to/kernelDriverFiles/
make
```

This will produce the compile kernel module object file `ppmDemux.ko`. If the kernel module has been cross-compiled on the Development Environment (VM) then copy this `.ko`-file to the Raspberry Pi's file system by using scp.

```
scp ./ppmDemux.ko pi@192.168.2.1:~/
```

In the above shown command, the kernel module object file will be copied to the home directory of the user `pi` on the Raspberry Pi's Linux system. If the IPv4 address has been changed, this has to be adapted when using the above stated command (here used: standard IP address of pre-configured Raspberry Pi Firmware).

After the file transfer is finished, the kernel module can be loaded to the Linux System. Navigate to the location of the stored kernel module object file (in this example the home directory of the user `pi`) and use the command `insmod`.

```
cd /home/pi/
sudo insmod ppmDemux.ko
```

If the kernel module is successfully loaded (no error messages occurred), then a character file handler `/dev/gpioirq24` has been created by the kernel module driver `ppmDemux`. Get noticed of rising edges on GPIO pin 24, read the character file handler `/dev/gpioirq24` as shown in the exemplary test code as shown in listing **??** or file `getPpmDemuxValues.c` in the SVN repository folder `/impl/trunk/kern/tst/`.

6.3 Stimulating the GPIO-Pins for validation

For testing and validation purposes of the custom built Kernel Driver `ppmDemux` of chapter 6.2, it was required to stimulate the GPIO pin 24 with a PPM signal similar to a Graupner sum signal.

For this purpose, a STM32F4 Discovery board was used to run a specifically written firmware that produces a variable PPM signal as depicted in fig. 6.3. The firmware of the STM32F4 board is capable to produce PPM signals with a pulse-to-pulse with of 480μs

up to 2560μs. By pressing the `blue button` of the STM32F4 Discovery board as shown in fig. 6.3, the pulse-to-pulse width can be altered in the sequence as shown on table 6.1.

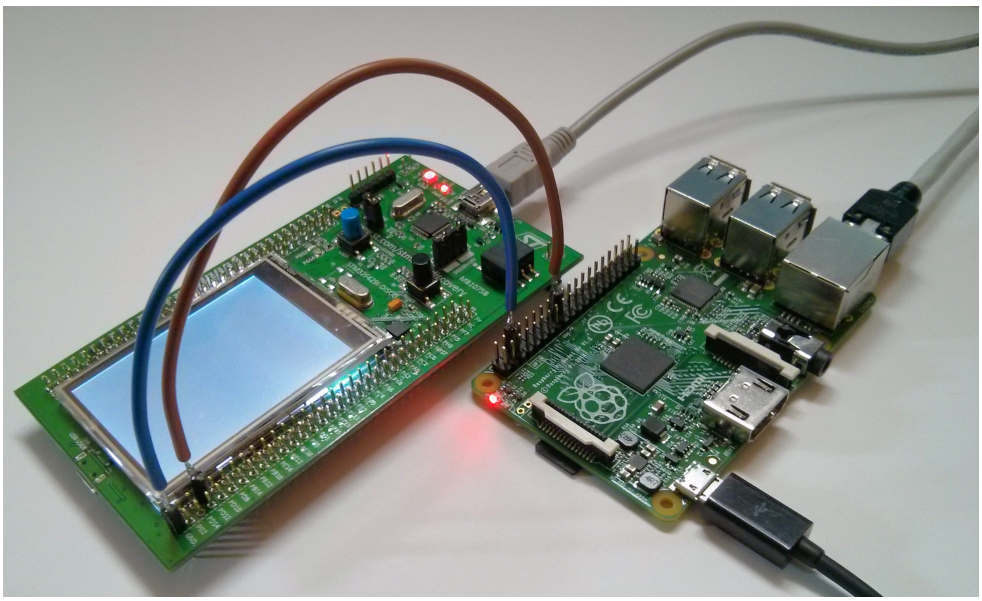


Abb. 6.3: In order to test the PPM kernel driver, a STM32F4 Discovery Board was used to stimulate the GPIO Pin 24 of Raspberry Pi Board. The signals got also observed by a 50 MHz Oscilloscope to validate the measured PPM length.

The firmware developed for the STM32F4 Discovery board can be found in the SVN repository in `/impl/trunk/kern/tst/STM32F4_PWM_Stimulator`. To use the firmware, first the CooCox IDE[1] has to be installed on the development machine as well as the GCC Toolchain for ARM Embedded Processors[2]. Then the provided project file in the SVN repository can be opened and customized, if needed.

State No.	Pulse-to-pulse witdth	State No.	Pulse-to-pulse witdth
1	480μs	10	1590μs
2	620μs	11	1710μs
3	730μs	12	1840μs
4	860μs	13	1960μs
5	970μs	14	2080μs
6	1100μs	15	2200μs
7	1220μs	16	2320μs
8	1350μs	17	2440μs
9	1460μs	18	2560μs

Tab. 6.1: Pulse-to-pulse width in microseconds for each state of the STM32F4 firmware

1 http://www.coocox.org/software/coide.php
2 https://launchpad.net/gcc-arm-embedded/+download

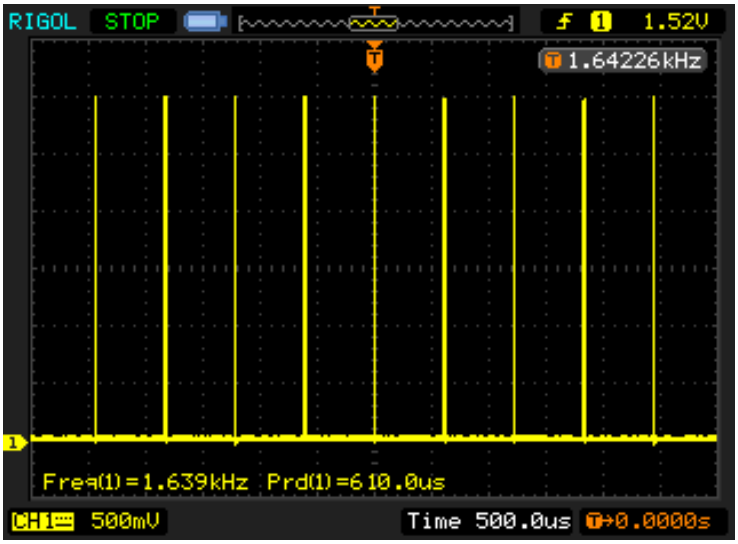


Abb. 6.4: Oscilloscope graph of stimulus signal emitted by the STM32F4 Discovery board (Custom STM32F4 firmware for 610μs pulse-to-pulse width).

To validate the stimulus signal output of the STM32F4 Discovery board, the output signals where observed with a 50Mhz oscilloscope. Screenshots of two PPM signals are shown in fig. 6.4 and fig. 6.5.

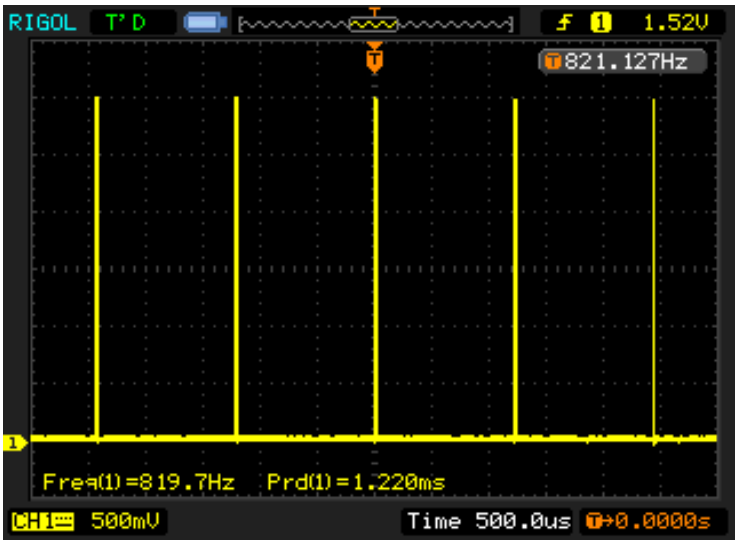


Abb. 6.5: Oscilloscope graph of stimulus signal emitted by the STM32F4 Discovery board (Custom STM32F4 firmware for 1220μs pulse-to-pulse width).

6.4 Results on experimental kernel driver

Before the custom kernel module driver `ppmDemux` has been developed by the authors of this document, a performance test of the standard Kernel's GPIO interface has been evaluated. For this purpose, a 1220μs PPM signal was generated (as shown in chapter 6.3) and measured with a simple test script.

The measured pulse-to-pulse time was measured under system idle and system under full load. The 'system under full load' scenario was produced by executing 3 threads that run at full load as well as 3 threads that produce full I/O-load. To achieve this, the program `stress` was installed on the Raspberry Pi's Linux system by issuing the command shown below on the Raspberry Pi's bash command line

```
1 sudo apt-get install stress
```

To set the test system under a full load scenario, the command `stress` got issued as shown below:

```
1 sudo stress -c 3 -i 3 &
```

This command starts 3 threads that each consume the max. CPU cycles available[1]. Additonally 3 threads are started that each put a max. I/O load on the system. This setup can be considered as a maximum load scenario for a the single core Raspberry Pi B+ board.

For each scenario (idle and full load), approximately 10000 measurements where taken and stored. In fig. 6.6 and fig. 6.7 the results of these measurements can be seen.

When using the standard Kernel's GPIO interface, the accuracy of the measurement under system idle is still in a acceptable range although latency jitter exceeds the acceptable range in rare cases (compare with fig. 6.6).

1 The ampersand character (`&`) at the end of the line runs the processes in the background of the system. To stop the program `stress` type the command '`sudo killall stress`' to the bash command line. This will terminate all processes of the program `stress` and set the system back to an idle state.

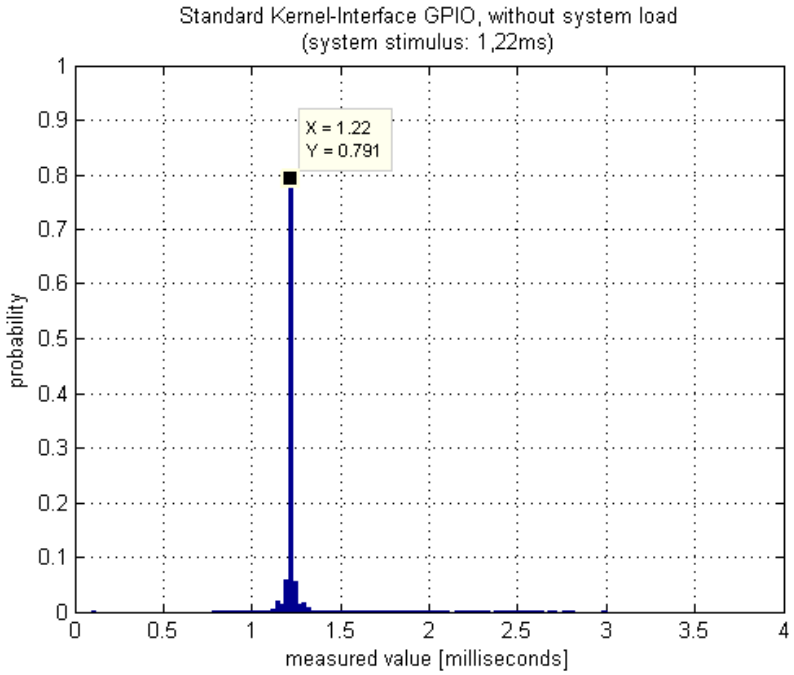


Abb. 6.6: Measured pulse period time with standard kernel's GPIO interface (system on idle)

On the other hand, when the system is under full load the latency jitter exceeds the acceptable range by far. As in fig. 6.7 can be seen, approximately 25% of all measured pulse-to-pulse widths are missed. This results in measured pulse-to-pulse width that is double the width of the stimulus signal. The second spike at $2.5ms$ indicates this effect. Unfortunately, for a mechanism to measure the user's remote control input, this is a unacceptable situation.

Therefore, the development of a custom kernel module driver called `ppmDemux` has been started. The proof-of-concept approach of chapter 6.2 has been tested under a Preempt_RT patched Kernel environment, as shown in chapter **??**.

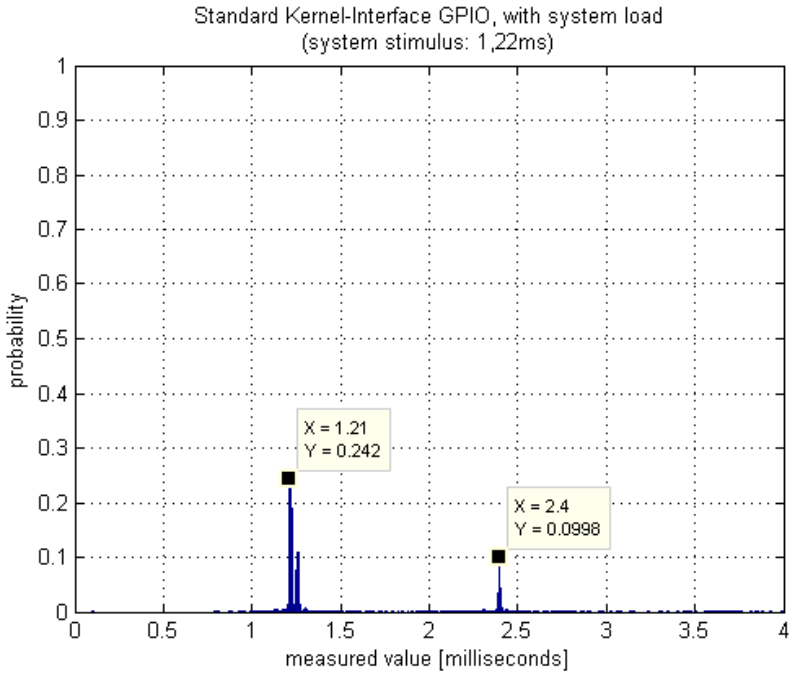


Abb. 6.7: Measured pulse period time with standard kernel's GPIO interface (system under full load). The second spike at 2.4ms shows that many rising edges of the stimulus signal are missed. This is a non-acceptable result for a reliable system.

The custom kernel driver `ppmDemux` has been test again under the two test scenarios: system idle and system under full load (using the command `stress`). Under system idle conditions, fortunately the measured results are very good. The latency is negligible small and the latency jitter has a variance of roughly $10\mu s$ for a stimulus PPM signal with a pulse-to-pulse width of $1220\mu s$ (as shown in fig. 6.8). With some post-processing filters, a measurement accuracy of 1%-3% is a very realistic range for a future implementation.

Fortunately, even under heavy system load (full load scenario), the latency is again negligible. Also the latency jitter rises only marginally. A variance of not more than $53\mu s$ for a stimulus PPM signal under full load can be assumed (as depicted in fig. 6.9). No single edge of the PPM signal has been missed. Since the measurement shows a robust and reliable measurement behavior of the custom kernel driver, this approach is considered to be acceptable for the high reliability requirements of this project.

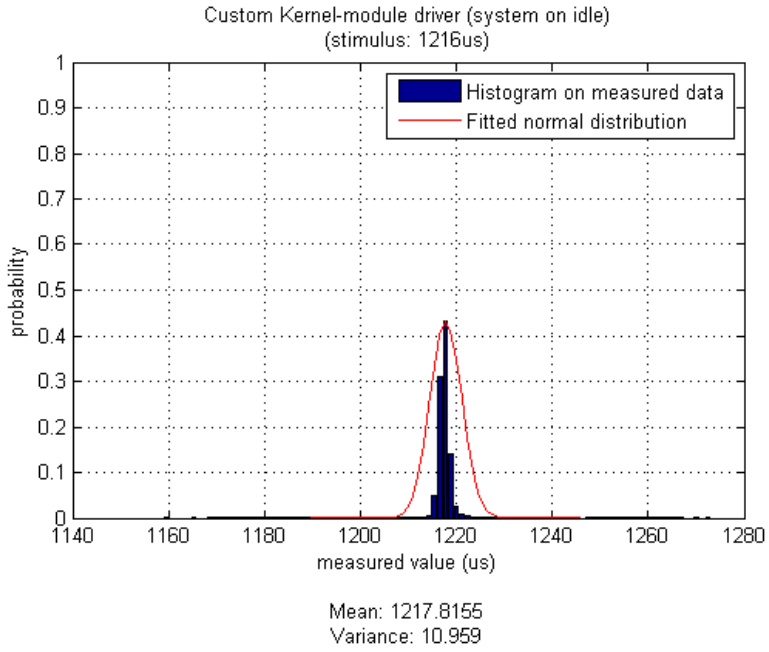


Abb. 6.8: Measured pulse period time with `ppmDemux` (system on idle). The very low latency jitter shows the superior performance of the kernel driver approach.

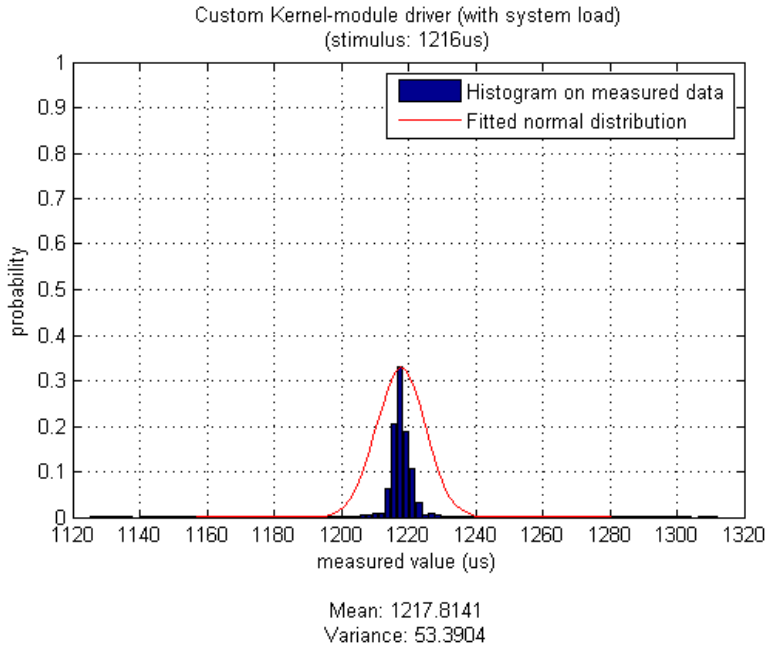


Abb. 6.9: Measured pulse period time with `ppmDemux` (system under full load). Even under system's full load, the latency jitter is in an acceptable range. No single rising edge of the stimulus signal is missed. This shows that the kernel driver approach is the best choice for the high reliability requirements of this project.

7 Defined Tests for our sensors

7.1 Infrared Distance Sensor

For detailed informations see also 9 or Data sheet.

Infrared Ranging Sensor, intended to measure the distance to ground.

We tested our Infrared Sensor with three different distances: 16cm, 49cm and to the ceiling (which represent the maximum of 150cm). Gathering a lot of data (10 000 values) for each measurement gets a an impression of the jitter and the precision of the value.

In the second steps we did the measurement with different surfaces. They were changing that much, that we search for another possibility to measure the distance. See xxxxxx

7.2 Ultrasonic Distance Sensor

These tests were only done in theory. Before we finally decided to proceed with that concept, we stoped it.

As we knew that the use of one ultrasonic sensor gives us not that good data and depening of the horizontal angel of the HElikopter, we thought that using three ultrasonic sensors could give us better distance data, independent from the flight angel. But merging 3 sensor to get one good value would a lot of effort and still don't provide us that good data.

While thinking about that concept, we found a better alternative.

7.3 LIDAR-lite Laser Distance Sensor

For detailed information about the Sensor see also the datasheet or Chapter 8.5.

We choose several distances, to validate the measurement. Always taking at least 10 values, to get an impression of the jitter. As the sensor is very good, there is only very small jitter above 20cm.

As the sensor supports distance up to 40 meters, we only tested it with 4 certain distances:
very small distance: under 1 meter (0,69m/0,20m)
small distance: between 1 and 2 meters (1,46/1,68m)
middle distance: above 3 meters ()
high distance: above 10 meters ()

Even the measurements diagonal through a room, with a big impact angle, worked really good.
Of course we also tested the sensor to different surfaces.

7.4 Inertial Measurement Unit (IMU)

The IMU consists of these 3 sensors:

7.4.1 Acceleration and Magnet Sensor(Compass)

The easiest way to test is, is manually. You trace, gather and display the data from the sensor and do some movements. You can move it as fast as possible over different distances. It should be possible to see at the different curves, how/weather the sensor works.

The Magnet sensor should be tested in a similar way. Depending on the magnetic fields in building, you can display the data and observe how the values change, when you turn the sensor/board/quadrocopter in different directions. In this case it is only relevant in which speed and with which divergence the sensor shows the data. The sensor delivers an absolute value, in which direction your sensor is looking at.

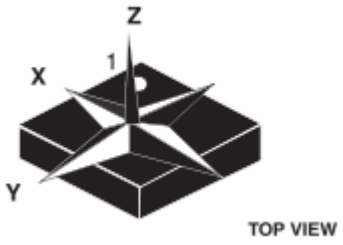


Abb. 7.1: Compass, Source: Data sheet

7.4.2 Gyroscope Sensor

This sensor gives information about the angles of all axis, how the quadrocopter lays in the air.
We check the work of sensor with observing the live data, as before.

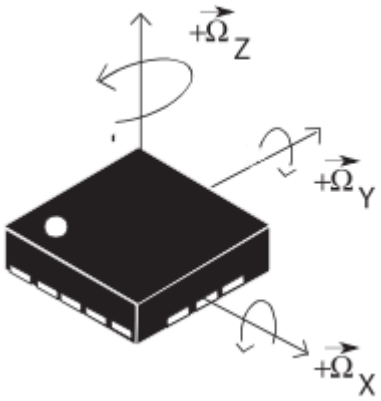


Abb. 7.2: Gyroscope, Source: Data sheet

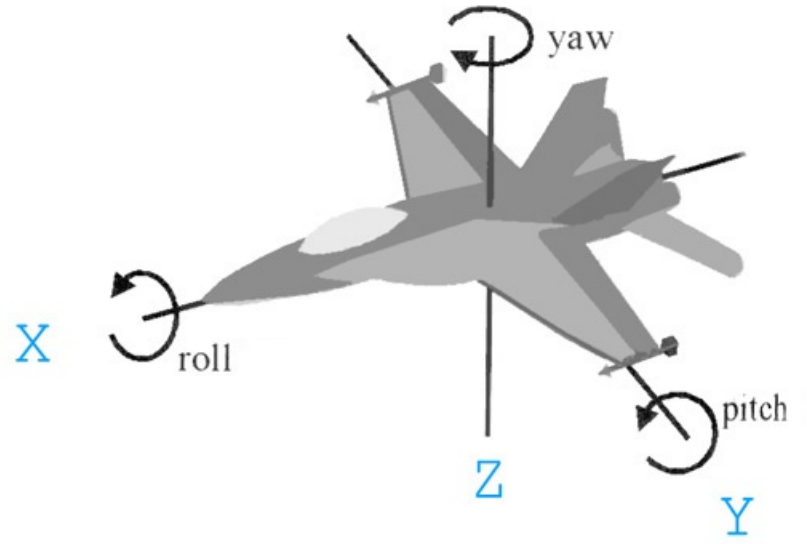


Abb. 7.3: Example at plane, Source: timzaman.com

7.4.3 Pressure Sensor

This sensor delivers a height compared to the sea level. You can say it is a absolute value. For that reason it easy to place the sensor at certain heights, and took a lot of data. So we can see the jitter. When lifting it up/down fast for several meters, we can check how fast we get the updated data.

In future this sensor will be controlled with the data form the distance measurements to the ground under us. So we can eliminate the influences from the weather for example.

8 Sensors and limits

This chapter is about the sensors we use and their maximum and minimum borders. Compared with the physical needed values, we try to realize a autonomous flight.

The sensors will be connected to a Raspberry Pi Board mounted on a Quadrocopter (HElicopter).

8.1 Analog- Digital Converter

Used Model: Adafruit ADS1015

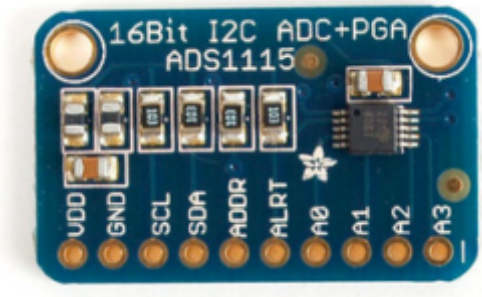


Abb. 8.1: Example of ADC, ADS1115

Resolution: 12 Bits
Programmable Sample Rate: 128 to 3300 Samples/Second
Power Supply/Logic Levels: 2.0V to 5.5V
Low Current Consumption: Continuous Mode: Only 150µA Single-Shot Mode: Auto Shut-Down
Internal Low-Drift Voltage Reference
Internal Oscillator
Internal PGA: up to x16
I2C Interface: 4-Pin-Selectable Addresses
Four Single-Ended or 2 Differential Inputs
Programmable Comparator

8.2 Infrared Analog Distance Sensor

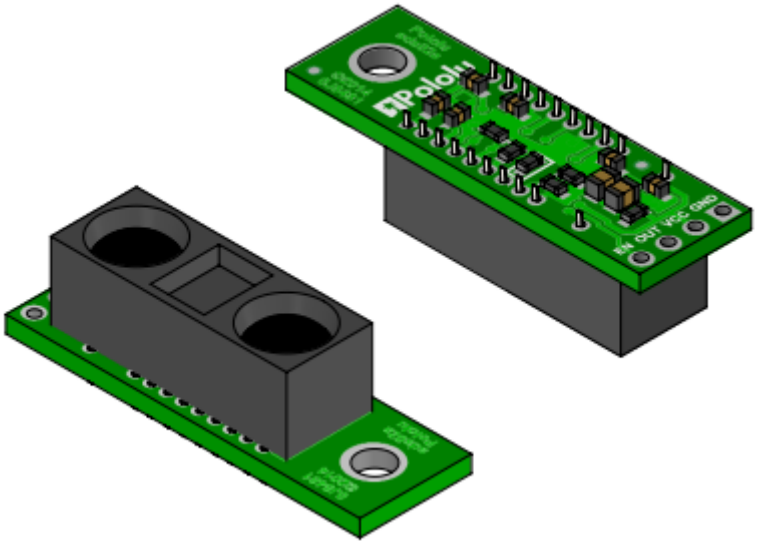


Abb. 8.2: IR Sensor on Carrier Board

Mounted IR Modul: Sharp GP2Y0A60SZLF

1. Distance measuring sensor is united with PSD, infrared LED and signal processing circuit.
2. Distance measuring range : 10 to 150 cm
3. Compact size (22.0 × 8.0 × 7.2mm)
4. Long distance measuring type (No external control signal required)
5. Analog output type
6. Update time: 16.5ms ± 3.7ms

8.3 GPS

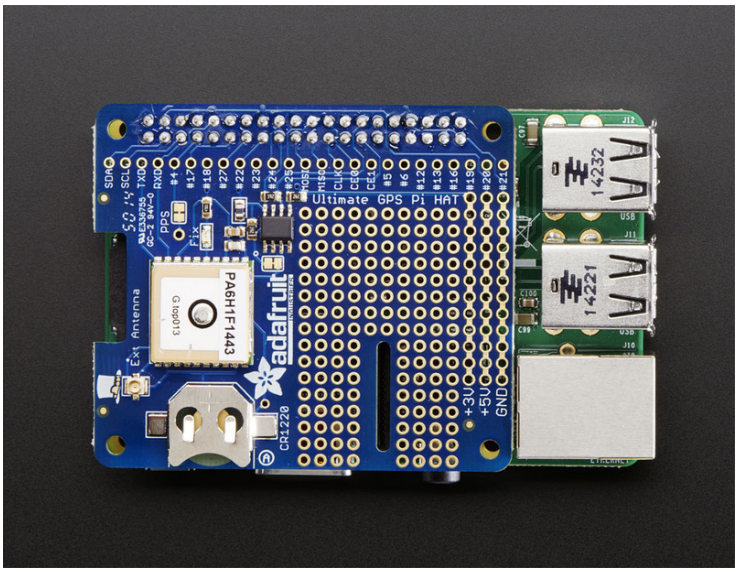


Abb. 8.3: Adafruit GPS Hat

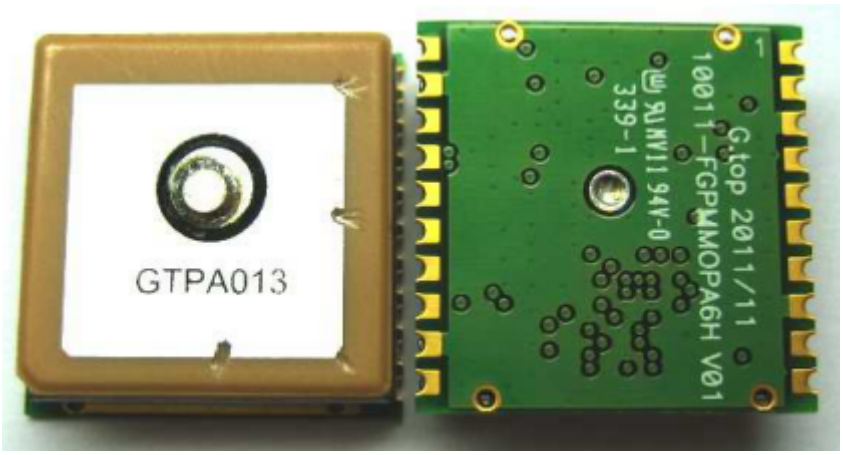


Abb. 8.4: GPS modul

- Built-in 15x15x2.5mm ceramic patch antenna on the top of module
- Ultra-High Sensitivity: -165dBm (w/o patch antenna), up to 45dB C/N of SVs in open sky reception.
- High Update Rate: up to 10Hz (note1)
- 12 multi-tone active interference canceller (note2)
- High accuracy 1-PPS timing support for Timing Applications (10ns jitter)
- AGPS Support for Fast TTFF („EPO“ Enable 7 days/14 days)
- Self-Generated Orbit Prediction for instant positioning fix

- „AlwaysLocate“ (note2)
- Intelligent Algorithm (Advance Power Periodic Mode) for power saving
- Logger function Embedded (note2)
- Automatic antenna switching function
- Antenna Advisor function
- Gtop Firmware Customization Services
- Consumption current(@3.3V):
Acquisition: 25mA Typical
Tracking: 20mA Typical
- E911, RoHS, REACH compliant

note 1: SBAS can only be enabled when update rate is less than or equal to 5Hz.
note2: Some features need special firmware or command programmed by customer, please refer to G-top „GPS command List“

8.4 Inertia Measurement Unit (IMU)

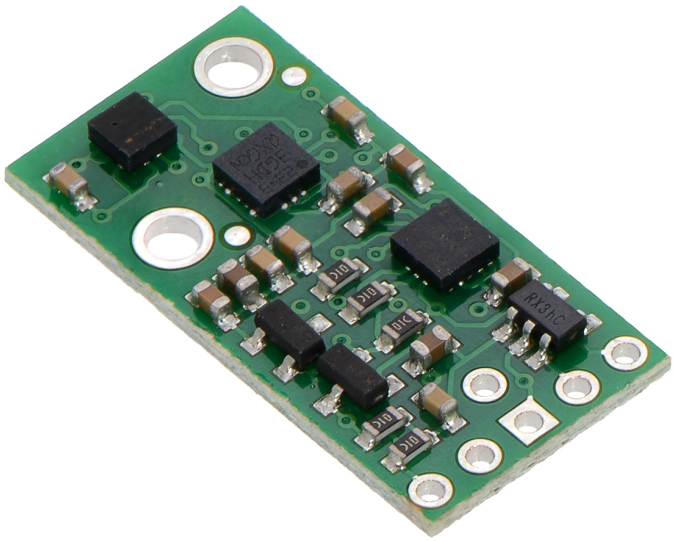


Abb. 8.5: Inertia Measurement Unit

Model: Pololu AltIMU

8.4.1 3D accelerometer and 3D magnetometer module

Features
- 3 magnetic field channels and 3 acceleration channels
- ±2/±4/±8/±12 gauss dynamically selectable magnetic full-scale
- ±2/±4/±6/±8/±16 g dynamically selectable linear acceleration full-scale
- 16-bit data output
- SPI / I2C serial interfaces
- Analog supply voltage 2.16 V to 3.6 V
- Power-down mode / low-power mode
- Programmable interrupt generators for free- fall, motion detection and magnetic field detection
- Embedded temperature sensor
- Embedded FIFO

Working of an accelerometer: The accelerometer is a small MEMS, that works on the principle of force acting on the MEMS. The micro springs attached to a piezoelectric component induces electricity in a capacitor. The output voltage is the measure of how strong the microsprings were pushed or pulled. One can think of an elevator where the mass reduces when the elevator is going down and hence the person inside the elevator can feel that the elevator is accelerating downwards. Hence, the basic principle of operation behind the MEMS accelerometer is the displacement of a small proof mass etched into the silicon surface of the integrated circuit and suspended by small beams.

8.4.2 MEMS pressure sensor

Features
- 260 to 1260 mbar absolute pressure range
- High-resolution mode: 0.020 mbar RMS
- Low power consumption:
- Low resolution mode: 5.5 uA
- High resolution mode: 30 uA
- High overpressure capability: 20x full scale
- Embedded temperature compensation
- Embedded 24-bit ADC
- Selectable ODR from 1 Hz to 25 Hz
- SPI and I2C interfaces
- Supply voltage: 1.71 to 3.6 V
- High shock survivability: 10,000 g
- Small and thin package
lead-free compliant

8.4.3 MEMS motion sensor: three-axis digital output gyroscope

Features
- Wide supply voltage, 2.2 V to 3.6 V
- Wide extended operating temperature range
(from -40 °C to 85 °C)
- Low voltage compatible IOs, 1.8 V
- Low power consumption
- Embedded power-down
- Sleep mode
- Fast turn-on and wake-up
- Three selectable full scales up to 2000 dps
- 16 bit rate value data output
- 8 bit temperature data output
- I2C/SPI digital output interface
- 2 dedicated lines (1 interrupt, 1 data ready)
- User enable integrated high-pass filters
- Embedded temperature sensor
- Embedded 32 levels of 16 bit data output FIFO
- High shock survivability

8.5 LIDAR-Lite Laser Ranging Module

Abb. 8.6: PULSEDLIGHT LIDAR Laser Sensor

Model LL-905-PIN-01

Performance
Range: 0-20m LED Emitter
Range: 0-60m Laser Emitterv (at full sunlight: 40m)
Accuracy: +/- 0.025m
Power: 5vdc, <100ma
Acquisition Time: < 0.02 sec
Rep Rate: 1-100Hz

Spread
At very close distances (less than a meter) the beam is about the size of the aperture (lens), at distances longer than that you can estimate it using this equation:

Distance/100 = beam size at that distance (in whatever units you measured distance in).

The actual spread is 8 milli-radians or 1/2 degree.

Configurations
- LED/PIN Diode, No Optics
- LED/PIN Diode, 12mm Optics
- Laser/PIN Diode 14mm Optics
(Class 1 Laser Product)

Interface
- I2C
- PWM

General Technical Specifications

Power 4.75 - 5.5V DC Nominal, Maximum 6V DC
Weight PCB 4.5 grams, Module 22 grams with optics and housing
Size PCB 44.5 X 16.5mm (1.75"by .65")
Housing 20 X 48 X 40mm (.8"X 1.9"X 1.6")
Current Consumption <2mA @ 1Hz (shutdown between measurements), <100mA (continuous operation)
Max Operating Temp. 70° C
External Trigger 3.3V logic, high-low edge triggered
PWM Range Output PWM (Pulse Width Modulation) signal proportional to range, 1msecmeter, 10µsec step size
I2C Machine Interface 100Kb - Fixed, 0xC4 slave address. Internal register access & control.
Supported I2C Commands Single distance measurement, velocity, signal strength
Mode Control Busy status using I2C, External Trigger input PWM outputs
Max Range under typical conditions aprox. 40m
Accuracy +/- 2.5cm, or +/- 1"
Default Rep Rate aprox. 50 Hz.

Laser Parameters

Wavelength: 905 nm (nominal)
Total Laser Power - Peak: 1.3 Watts
Mode of operation: Pulse (max pulse train 256 pulses)
Pulse Width: 0.5 uSec (50% duty Cycle)
Pulse Repetition Frequency: 10-20 KHz nominal
Energy per Pulse: <280 nJ
Beam Diameter at laser aperture: 12 mm x 2 mm
Divergence: 4 m Radian x 2 m Radian (Approx)

Innovation Summary

- The use of a signature matching technique (known as signal correlation) that estimates time delay by electronically sliding a stored transmit reference over the received signal in order to find the best match.
- Operation of the infrared LED or laser in short bursts allowing a 100:1 advantage in peak output power over measurement systems using a continuous beam.
- Decreased measurement times down to a millisecond or less allows significant power consumption advantages and high repetition rates for scanning applications.
- We have developed novel current driver technology with nanosecond signal transition times at high peak currents to produce high power transmit burst sequences.
- Our signal processing approach is implementable in a single programmable logic chip or System-on-Chip (SoC) to allow deployment without the costly development of custom processing chips.
- Detector switching technology allows multiple detectors to be processed by a single signal-processing channel. Enabling compact multichannel systems deployable in under one square inch of board space.
- Multiple digital processing cores implementable in a single cost effective programmable logic chip.
- Optical scanner technology to multiply low-resolution electronic scanning to higher resolutions.

Source: Data sheet

8.5.1 Technology

PulsedLight‘s "Time-of-flight"distance measurement technology is based on the precise measurement of the time delay between the transmission of an optical signal and its reception. The patented, high accuracy measurement technique enables distance measurement accuracy down to 1 cm by the digitization and averaging of two signals; a reference signal fed from the transmitter prior to the distance measurement and a received signal reflected from the target. The time delay between these two stored signals is estimated through a signal processing approach known as correlation, which effectively provides a signature match between these two closely related signals. The correlation algorithm accurately calculates the time delay, which is translated into distance based on the known speed-of-light. A benefit of PulsedLight‘s approach is the efficient averaging of low-level signals enabling the use of relatively low power optical sources, such as LEDs or VCSEL (Vertical-Cavity Surface-Emitting) lasers, for shorter-range applications and increased range capability when using high power optical sources such as pulsed laser diodes.

Source: LidarLite Operating Manual

8.5.2 possible measurement problems

When the LIDAR-Lite unit sometime unexpected results, first think about these points:

There are several variables to consider if your LIDAR-Lite fails to return a valid measurement or seems not to recognize an object at all. These can be categorized into the following areas:

A. Reflectivity of the object
B. Distance of the object from the sensor
C. Size of the object relative to the transmitted infrared beam
D. Direct or reflected sunlight finding its way into the receiver
E. Atmospheric conditions
F. Obstruction of the receiver lens
F. Failure of the LIDAR-Lite unit

We'll consider reflectivity here:

Reflectivity

Reflective characteristics of an object's surface can be divided into three categories (in the real world, a combination of characteristics is typically present):
A. Diffuse Reflective
B. Specular, and
C. Retro-reflective

Diffuse Reflective

In the case of purely diffuse surfaces, we are talking about materials that have a textured quality that causes reflected energy to disperse uniformly. This tendency results in a relatively predictable percentage of the dispersed laser energy finding its way back to the LIDAR-Lite receiver. As a result, these materials tend to read very well. Materials that fall into this category are paper, matte walls, and granite. It is important to note that materials that fit into this category due to observed reflection at visible light wavelengths may exhibit unexpected results in other wavelengths. The near infrared range used by the LIDAR-Lite transmitter may detect them as nearly identical. A case in point is a black sheet of paper may reflect a nearly identical percentage of the infrared signal back to the

receiver as a white sheet.

Specular

Specular surfaces, on the other hand, are difficult or impossible for the LIDAR-Lite to recognize because radiated energy is not dispersed. Reflections off of specular surfaces tend to reflect with little dispersion which causes the reflected beam to remain small and, if not reflected directly back to the receiver, to miss the receiver altogether. The LIDAR-Lite may fail to detect a specular object in front of it unless viewed from the normal. Examples of specular surfaces are mirrors and glass viewed off-axis.

Retro-reflective

Retro-reflective surfaces return a very high percentage of radiated energy to the receiver due to their reflective properties. Light hitting a retro-reflective surface will return to the receiver without much signal loss so retro-reflective surfaces are typically very good targets for the LIDAR-Lite. Paint used to mark roadways, animals' eyes, license plates and road signs are examples of retro-reflective surfaces. Some bicycle reflectors are retro-reflective in the visible spectrum but are not easily detected by LIDAR-Lite due, in part, to their failure to reflect infrared wavelengths as efficiently as they do light in the visible spectrum.

9 Infrared Sensor

To achieve our main goal, the autonomous landing, we tried several different sensor. In this Chapter we introduce them, including the problems we were faced and the configurations we did.

We tried to establish a distance measurement with GP2Y0A60SZLF Infrared Sensor on a Pololu Carrier Board. The Measuring distance of that sensor is 10 to 150 cm. For detailed technical values, please see the sensor chapter or the datasheet.

9.1 First steps

Our first action was to check, weather the sensor is working or not. So we need to active the I²C Bus on the Raspberry Pi and set up the Analog Distance Converter (ADS1011), because the Infrared Sensor provides us only a analog output.
We did a fast test of the sensor using python, just for checking the sensor is not broken.

To gather a bunch of data, we developed I²C and ADC Driver ourselves in C.

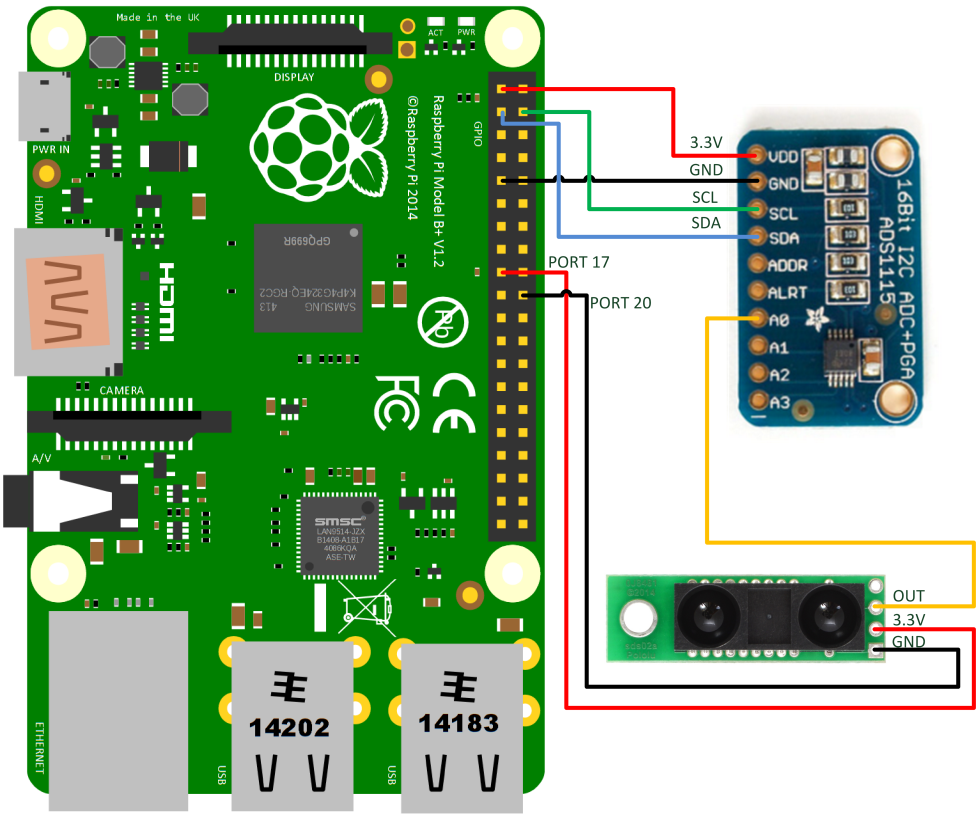


Abb. 9.1: Wiring ADC and IR

9.1.1 ADC Configuration

First Hex depends on Starting Conversion + the Input, which Pin to read A0-3,
Second Value is PGA (001)=+-4,099V and continuous Mode (0).

These three bytes are written to the ADS1015 to set the config register and start the conversion.

l_writeBuf_rg24[0] = 1;
This sets the pointer register to write two bytes to the config register

l_writeBuf_rg24[1] = l_mux_ui8;
This sets the 8 MSBs of the config register (bits 15-8) to 11000011

l_writeBuf_rg24[2] = 0x23;
This sets the 8 LSBs of the config register (bits 7-0) to 00100011

The following table shows the Hex values in the direction from top to bottom what is needed for reading a conversion value on the specific inputs. At the empty field, there is no change compared to Input A0.

	Input A0	Input A1	Input A2	Input A3
Slave adress+RW	0x49			
PTR register	0x01			
MSB Config	0xC2	0xD2	0xE2	0xF2
LSB Config	0x23			
Slave adress+RW	0x49			
PTR register	0x00			
Data from Slave	0xXX	0xXX	0xXX	0xXX
Data from Slave	0xXX	0xXX	0xXX	0xXX

Tab. 9.1: ADC Conversion Read

MSB:
The first hexadecimal value is to start the conversion and depends on the Input, which Pin to read A0-3.
The second hexadecimal value is PGA (001)= +-4,099V and continuous Mode (0).

LSB:
The first hexadecimal value is the sample Rate. (001) sets it to 250SPS + Comp Mode (0).
The second hexadecimal value is the Comp. config. (0011) disables the comparator.

9.2 Measured Values

With our own logic getting the data from the sensor, we could store as much data we want. Now we were able to get a impression on the jitter and could thing about a good algorithm to get reliable values.

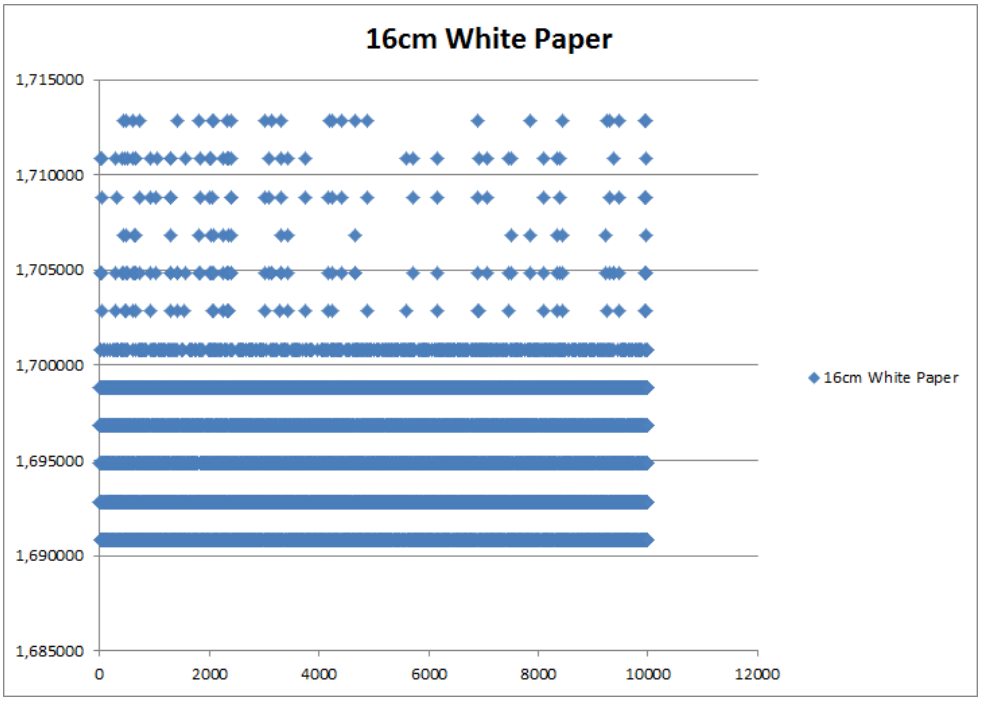


Abb. 9.2: IR: 16 cm to white paper

Gathering 10 000 values while measuring the distance of 16cm on a white paper. We used the white paper to compare the values with them in the datasheet.

As we changed the distance and the surface, we discover a astonishing fact.

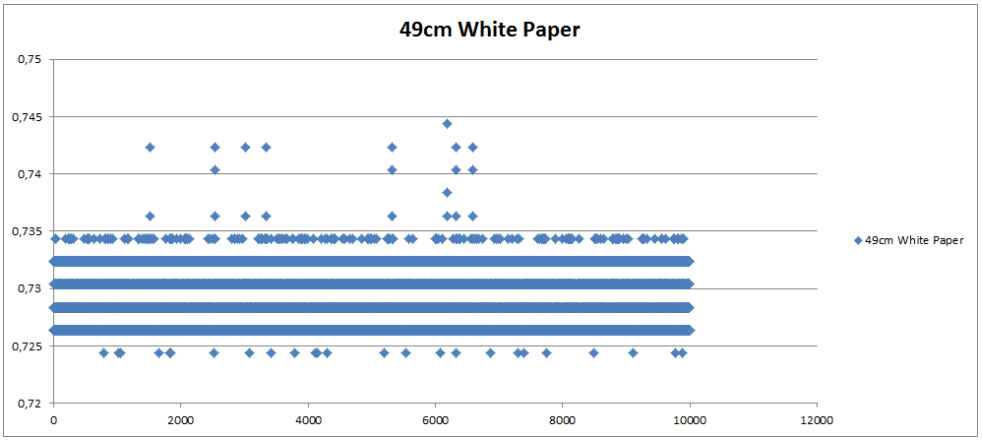


Abb. 9.3: IR: 49 cm to white paper

Same measurement on the surface of the table:

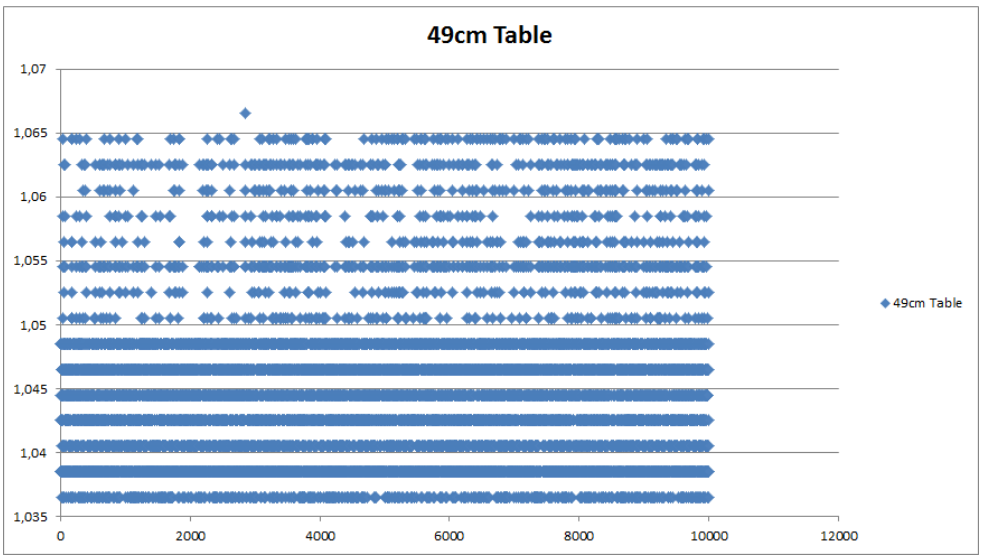


Abb. 9.4: IR: 49 cm to table

As you see, the measurements on different surfaces are highly varying. At 49cm we got the following rough values:

White Paper: 0,73 V
Desk Surface: 1,42 V

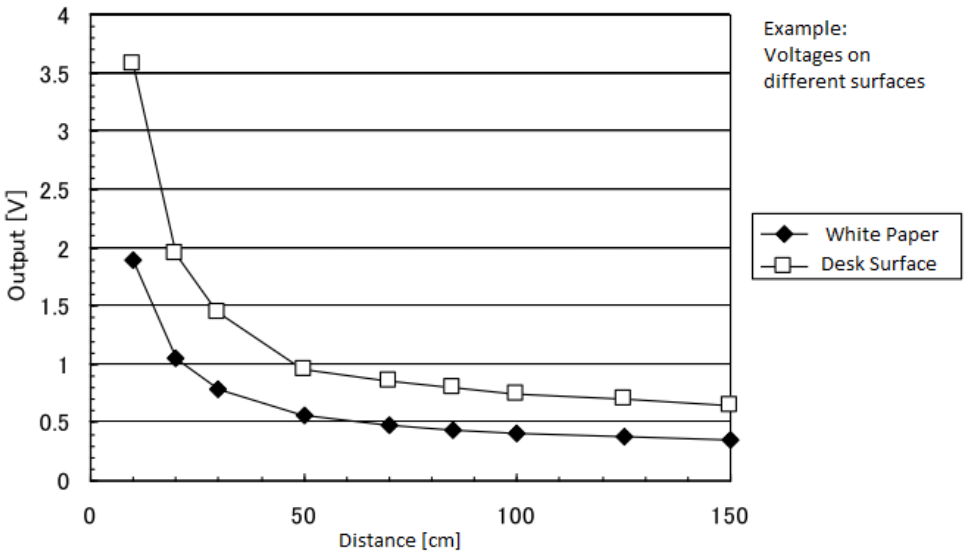


Abb. 9.5: IR: Sensor values (voltages) on different surfaces

2 Surfaces and 2 Distances:

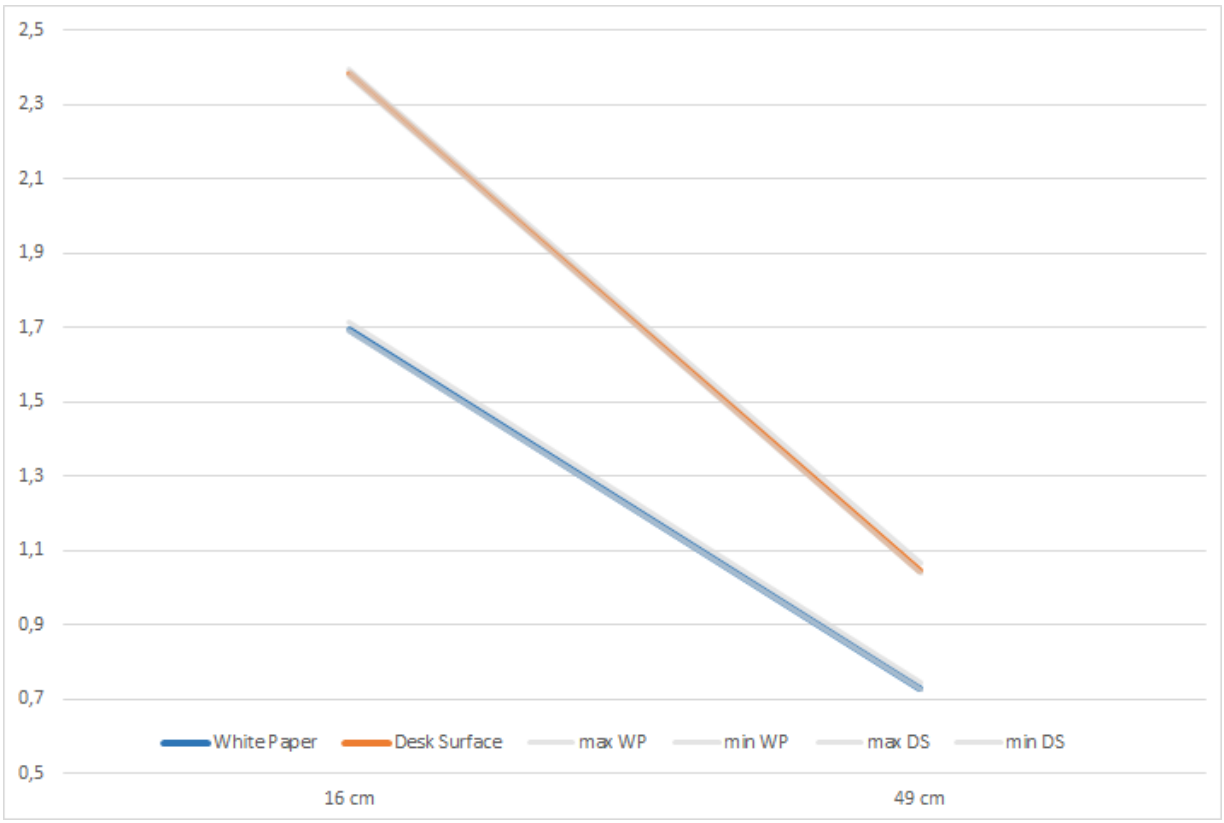


Abb. 9.6: IR: Compared Voltages on two surfaces

We only measured 2 different distances in order to check the functionality of the sensor. The grey lines shows the jitter which we measured.

9.3 Conclusion

As we could not guarantee a good flying with changing surfaces, we decided to stop chasing a solution with the IR Sensor.

10 Ultrasonic Distance Sensor

Our next approach was, measuring the distance with a ultrasonic distance sensor. In a meeting we agreed on using 3 ultrasonic sensor to measure the distance to ground, because a single sensor is imprecise. We hoped that a sensor fusion of 3 ultrasonics will bring us better values.

While searching for theses sensors, we found a relatively cheap Laser Sensor. After a short consultation, we decided to go with that one. So Ultrasonic was discarded, before start.

11 LIDAR Laser Sensor

LIDAR-Lite Laser Distance Sensor
Model LL-905-PIN-01

Performance
Range: 0-20m LED Emitter
Range: 0-60m Laser Emitter

Interfaces
- I^2C
- PWM

For detailed technical values, please see the sensor chapter or the datasheet.

11.1 Connecting the sensor with I^2C

Because there is no proper input for a PWM signal, we used the I^2C Bus to connect the laser sensor. As the sensor does not support fast I^2C mode, which we need, we decided to use the second I^2C on the Raspberry Pi. This bus is orginally located on the camera port with an FPC (flexible printed circuit) Connector. We managed it to redirect it to the normal Pins on the board. See Chapter xxxxxx.

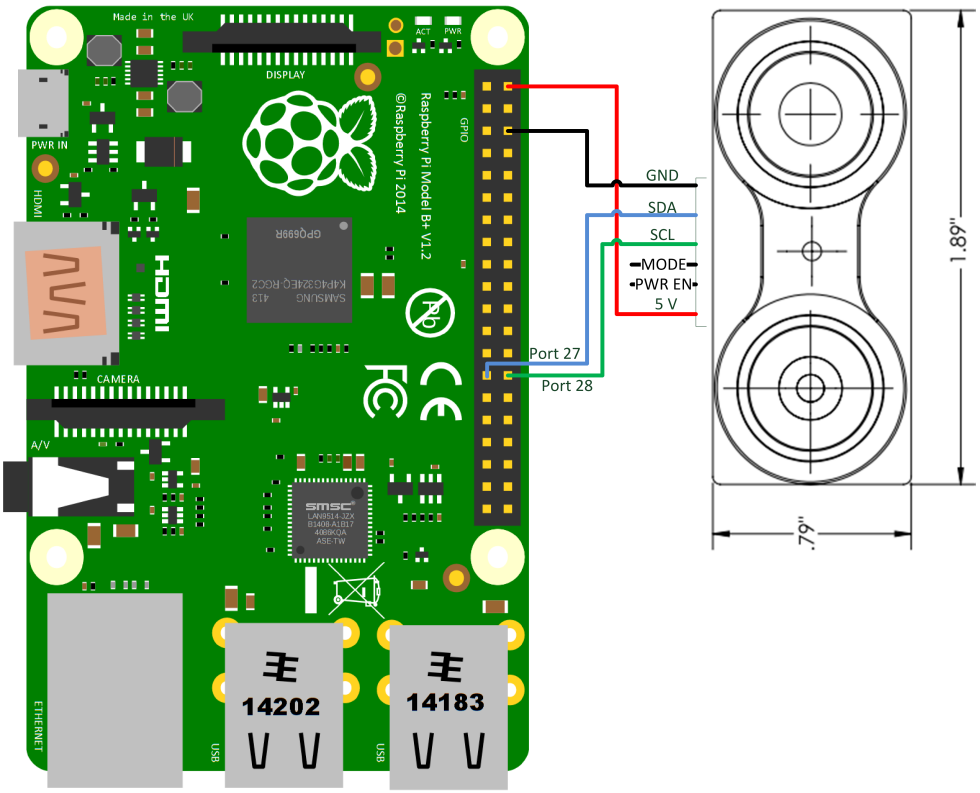


Abb. 11.1: Wiring LIDAR

11.2 First steps

Quick Start Guide

1. Make Power and I2C Data Connections as per J1 connector pin out diagram. Pins 2 & 3 are optional connections and not required.
2. Initialization: Apply Power to the Module. The sensor operates at 4.75-5.5V DC Nominal, Maximum 6V DC.
3. Measurement: Write register 0x00 with value 0x04 (This performs a DC stabilization cycle, Signal Acquisition, Data processing). Refer to the section „I2C Protocol Summary“ in this manual for more information about I2C Communications.
4. Periodically poll the unit and wait until an ACK is received. The unit responds to read or write requests with a NACK when the sensor is busy processing a command or performing a measurement. (Optionally, wait approx. 20 milliseconds after acquisition and then proceed to read of high and low bytes)
5. Read: register 0x0f, returns the upper 8 bits of distance in cm, register 0x10, returns the lower 8 bits of distance in cm. (Optionally a 2-Byte read starting at 0x8f can be done)

11.3 Measured Values

To test and verify functionality and accuracy of the sensor a series of tests was done. The following figure shows the deviation of the sensor of different surfaces. Each bar represents the mean value of at least ten measurements. As you can see at a distance of app. 150 cm the deviation is only 2 cm. With this tests the given accuracy of +/- 2.5 cm (see datasheet) gets confirmed.

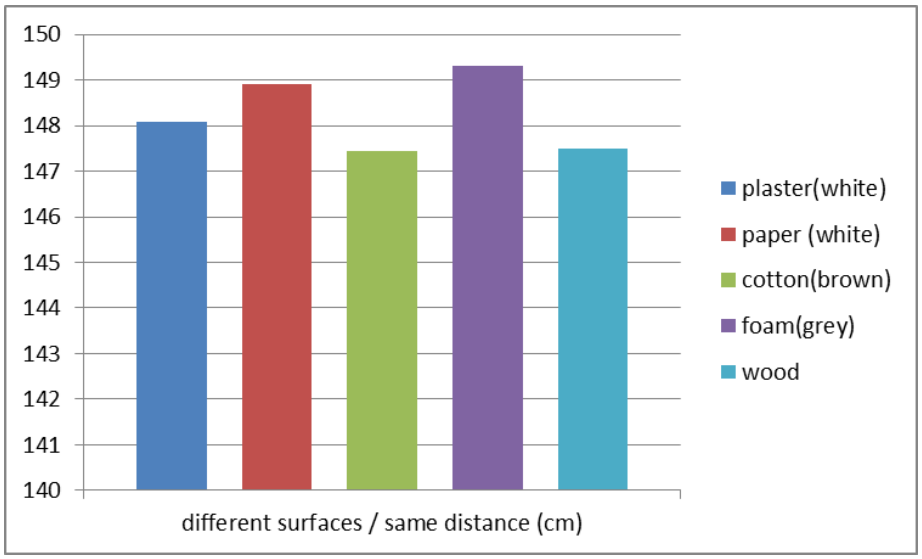


Abb. 11.2: Laser measured values

To verify a correct distance measurement while flying also different angles to ground were tested. In a set of tests with an angle between 90°(vertically) and 10°and a distance range of app. 200cm the sensor returned valid and correct values.

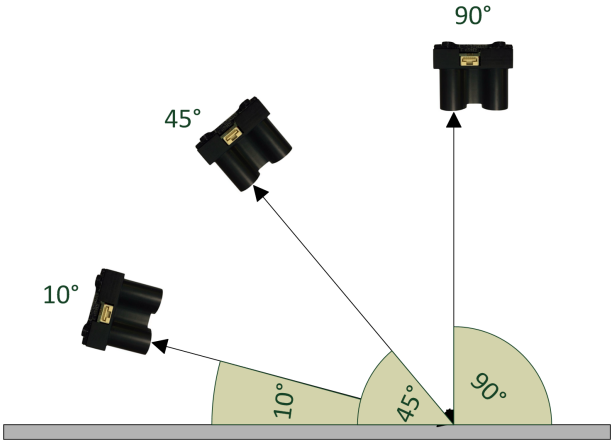


Abb. 11.3: different measured angles

11.4 conclusion

The Laser sensor provides us reliable distance values, independent from the surface or material we measure at. The measurement get's a little bit worse below 20 cm, but with a estimated build in height of 10 cm at the quadrocopter, we can along with that.

We recommend to pursuit future autonomous landing approaches, with that sensor.

Regarding the Class 1 Laser Specification, there is no danger in using the sensor, even when looking straight in the laser.

12 Functions in C

These chapter is about the C-Code we developed to work via Raspberry Pi with the sensors.

12.1 I²C

```
1 unsigned int g_lldI2c_WriteI2c_bl(unsigned char, const unsigned
    char*, unsigned int);
```

List. 12.1: Write on I2C-1

Parameter:	unsigned char	slave address of device
	const unsigned char*	buffer with data to write
	unsigned int	number of bytes to write
Return:	unsigned int	error detection, 0=OK 1=failure
Description:	function to write several data on the I2C-1 Bus of the Raspberry PI	

```
1 unsigned int g_lldI2c_ReadI2c_bl(unsigned char, const unsigned char
    *, unsigned int);
```

List. 12.2: Read from I2C-1

Parameter:	unsigned char	slave address of device
	const unsigned char*	array to store read data
	unsigned int	number of bytes to read
Return:	unsigned int	error detection, 0=OK 1=failure
Description:	function to read several data on the I2C-1 Bus of the Raspberry PI	

```
1 unsigned int g_lldI2c_WriteI2c0_bl(unsigned char, const unsigned
    char*, unsigned int);
```

List. 12.3: Write on I2C-0

Parameter:	unsigned char	slave address of device
	const unsigned char*	buffer with data to write
	unsigned int	number of bytes to write
Return:	unsigned int	error detection, 0=OK 1=failure
Description:	function to write several data on the I2C-0 Bus of the Raspberry PI	

```
1 unsigned int g_lldI2c_ReadI2c0_bl(unsigned char, const unsigned
    char*, unsigned int);
```

List. 12.4: Read from I2C-0

Parameter:	unsigned char	slave address of device
	const unsigned char*	array to store read data
	unsigned int	number of bytes to read
Return:	unsigned int	error detection, 0=OK 1=failure
Description:	function to read several data on the I2C-0 Bus of the Raspberry PI	

12.1.1 Configuration

$\#include < linux/i2c - dev.h >$ is doing the (local) device handling. We only use the slave address to communicate with the connected devices. Read or write commands are provided by fcntl.h.

12.1.2 I²C Write

Writes the number of stated bytes from the write buffer with the "write"command to the chosen I²c-device, depending on called function.

12.1.3 I²C Read

Reads the number of stated bytes from the read buffer with the "read"command from the chosen I²c-device, depending on called function.

12.2 Analog-Digital-Converter (ADC)

```
1 float g_halADC_get_ui16(unsigned char );
```

List. 12.5: Read from ADC

Parameter:	unsigned char	A0-A3 input selection
Return:	float	converted analog values
Description:	Interface to read ADS1015 (ADC)	

12.2.1 Configuration

Sensor Board Name: Pololu ADS1015
Sensor Name: GP2Y0A60SZLF

l_mux_ui8 = 0xC2;
"C "$_{16}$:
The first Hex-Value depends on Starting Conversion + the Input, which Pin to read A0-3

"2 "$_{16}$:
The second Value is PGA (001)=+-4,099V and continuous Mode (0)

These three bytes are written to the ADS1015 to set the config register and start the conversion

l_writeBuf_rg24[0] = 1;
This sets the pointer register to write the following two bytes to the config register

l_writeBuf_rg24[1] = l_mux_ui8;
This sets the 8 MSBs of the config register (bits 15-8) to 11000011

l_writeBuf_rg24[2] = 0x23;
This sets the 8 LSBs of the config register (bits 7-0) to 00100011

"2 "$_{16}$:
// First Hex is sample Rate. (001) sets to 250SPS + Comp Mode (0)

"3 "$_{16}$:
// Second Hex is Comp. config. (0011) disable the comparator

The l_writeBuf_rg24 is written to the configuration register of the ADC. After that, we can read the converted analog values from the chosen input.

12.2.2 ADC Read

To read a converted analog value, you need to set the pointer register to 0.

When the pointer register is set to 0 this signals that the converted analog value should be provided. You will get the these values when performing a i^2c-read command the next time. This value is 16 Bit large and will be calculated as a float-value, depending on the resolution which is adjusted.

12.3 Infrared Sensor

```
1 float g_halADC_get_ui16(unsigned char );
```

List. 12.6: Read Infrared

Parameter:	char	select Input on which IR is connected
Return:	float	Voltage from Sensor
Description:	Since our IR Sensor only provides analog output, we need to use the ADC	

12.3.1 Configuration

There is no configuration of the Infrared sensor needed.

12.3.2 Read Sensor Values

We get the analog values with the ADC-Function.

12.4 LIDAR-Lite Laser Sensor

```
1 double g_LIDAR_getDistance_f64(void);
```

List. 12.7: get Laser Distance

Parameter:	void	
Return:	double	distance in meter
Description:	returns calculated distance value in meters	

```
1 int g_LIDAR_readDistanceFromI2C_i32(void);
```

List. 12.8: trigger Laser measurement

Parameter:	void
Return:	int error detection, 0=OK -1=failure
Description:	triggers a measurement and stores the result

12.4.1 Configuration

Trigger Measurement of Distance (DC stabilization cycle, Signal Acquisition, DataProcessing)

First Config Byte: 0x00;
Representation of configuration register 0x00 of the laser sensor.
Second Config Byte: 0x04;
Take acquisition and correlation processing with DC correction

Set Reg 0x8f as Output-Register to read a two-byte value which gives the distance in cm.

12.4.2 Read Sensor Values

Read two-byte distance in cm from register 0x8f. This value is stored as a decimal value in cm.
Alternative you can read the high-byte of the measured value from register 0x0f and the low-byte from register 0x10.

13 I²C Configuration

In normal configuration the second I²C-Bus of the Raspberry Pi is set up as two of the output pins of the DSI Display Connector resp. the CSI Camera Connector.
To make the setup of the quadrocopter as easy as possible and with respect to the weight and soldering/cabling these output pins were redirecteed to two of the 40 pins of the GPIO Header.

This gets done by useage of a Python-script (see below) which gets excecuted while booting the system. To get this configuration running two additional files need to be edited.

In "/boot/cmdline.txt"
`bcm2708.vc_i2c_override=1`
has to be added
in "/etc/modprobe.d/i2c_o_enable.conf"
`blacklist snd_soc_tas5713`
has to be added.

After this the GPIO port 27 is configured as SDA0 and the GPIO port 28 as SCL0.

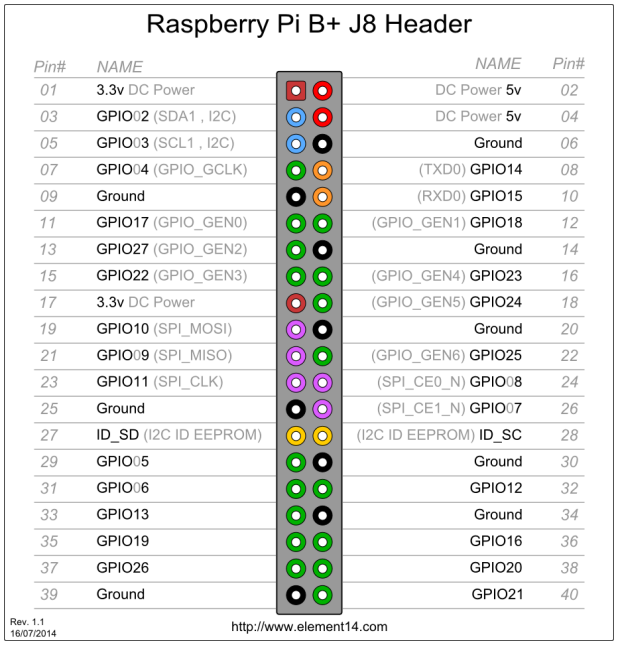


Abb. 13.1: GPIOs [2]

2 http://www.element14.com/community/servlet/JiveServlet/previewBody/68203-102-6-294412/GPIO.png

```
#!/usr/bin/python
#!/usr/bin/env python
#
#

# #######
# For I2C configuration test
import os
import mmap
bplus=0
BCM2708_PERI_BASE=0x20000000
GPIO_BASE=(BCM2708_PERI_BASE + 0x00200000)
BLOCK_SIZE=4096

def _strto32bit_(str):
    return ((ord(str[3])<<24) + (ord(str[2])<<16) + (ord(str[1])<<8) + ord(str[0]))

def _32bittostr_(val):
    return chr(val&0xff) + chr((val>>8)&0xff) + chr((val>>16)&0xff) + chr((val>>24)&0xff)

def get_revision():
    with open('/proc/cpuinfo') as lines:
        for line in lines:
            if line.startswith('Revision'):
                return int(line.strip()[-4:],16)
    raise RuntimeError('No revision found.')

def i2cConfig():
    if get_revision() >= 10:
       print "B+ or CM detected."
       s0 = 0b00000000000000000000000100100100
       s2 = 0b00000000000000000000000000000000
    if get_revision() <=9:
       s0 = 0b00000000000000000000000100100000
       s2 = 0b00100100000000000000000000000000
    if get_revision() <= 3:
        print "Rev 2 or greater Raspberry Pi required."
        return
    # Use /dev/mem to gain access to peripheral registers
    mf=os.open("/dev/mem", os.O_RDWR|os.O_SYNC)
    m = mmap.mmap(mf,BLOCK_SIZE, mmap.MAP_SHARED,
            mmap.PROT_READ|mmap.PROT_WRITE, offset=GPIO_BASE)
    # can close the file after we have mmap
    os.close(mf)
    # Read function select registers
    # GPFSEL0 -- GPIO 0,1 I2C0   GPIO 2,3 I2C1
    m.seek(0)
    reg0=_strto32bit_(m.read(4))
    # GPFSEL2 -- GPIO 28,29 I2C0
    m.seek(8)
    reg2=_strto32bit_(m.read(4))
    # print bin(reg0)[2:].zfill(32)[2:]
    # print bin(reg2)[2:].zfill(32)[2:]

    # GPFSEL0 bits --> x[26] SCL0[3] SDA0[3]
    #                          GPIO    GPIO
    m0 = 0b00000000000000000000111111111111
    #s0 = 0b00000000000000000000100100100100
    b0 = reg0 & m0
    if b0 <> s0:
        #print "reg0 I2C configuration not correct. Updating."
        reg0 = (reg0 & ~m0) | s0
        m.seek(0)
        m.write(_32bittostr_(reg0))

    # GPFSEL2 bits --> x[2] SCL0[3] SDA0[3] x[24]
    m2 = 0b00111111100000000000000000000000
    b2 = reg2 & m2
    if b2 <> s2:
        #print "reg2 I2C configuration not correct. Updating."
        reg2 = (reg2 & ~m2) | s2
        m.seek(8)
        m.write(_32bittostr_(reg2))

    # No longer need the mmap
    m.close()


if __name__ == '__main__':
    i2cConfig()
```

List. 13.1: I²C0 Port-Configuration

14 Conclusion and outlook

14.1 Achieved project goals and results

First of all a project plan for the two groups was set up and the needed hardware was chosen. Following this a plan of the mounting and wiring was drawn and two prototypes were build up.
To increase the flexibility of using various operating systems on the development computer an Ubuntu system on a virtual machine is used. Programming is done by Eclipse which also is installed in the virtual machine. Additionally cross compiling and the ability to debug is enabled. Due to the fact that the compiling runs on the development computer, the speed of compiling is dramatically increased. The remote debugging feature helps to track down and fix errors during the development process. The already set up development environment can be downloaded from the SVN repository.
To ensure real-time capability, a real time patch is applied to the Raspbian distribution of the Raspberry Pi.
By introducing of hierarchies in the software, the hardware dependency is separated to just one layer. This makes the software better portable. In case of hardware changes, only in the affected layer software has to be changed. With this in mind, it was split up in Low level driver, the hardware abstraction, signal processing and application layer.
Beginning with the programming an interface abstraction to the Low level drivers of the UART was developed and proofed. The also needed interface abstraction of the I2C driver was implemented by the second project group. Just some improvements were done by the authors of this document. The Graupner PPM decoding is enabled via a Kernel module which can be loaded directly to the Kernel of the system. This enables interrupting with a very low jitter. This is needed to provide accurate measuring of the PPM pulses of the remote control. Additionally an experimental time trigger is provided via a second Kernel module driver. This is used to provide accurate timing for the control loop. The patched and pre-configured operating system of the Raspberry Pi can also be downloaded from the SVN repository.
After successful testing of the low level drivers in conjunction with the interface abstraction, the development of the hardware abstraction layer was started. The implementation of the GPS driver and the Inertial measurement unit sensors was started and successfully finished. All of these modules provide an data interface for the next hierarchy. All the drivers are tested as single units to ensure proper working.
The last step of the implementation was the signal processing layer. On this hierarchy an reduced IMU-Filter and the Orientation fusion was implemented.

The sensor fusion covers the successful implementation of the complementary Filter and the Kalman Filter. Additionally a generic matrix library was programmed to enable Matrix operations for the Kalman Filter. This library can be used with matrices of various sizes, written in pure C-code. The sensor fusion provides the absolute orientation of the system. The orientation angles of roll, pitch and yaw are with the successful fusion independent of the restrictions of the sensors and provide accurate angles.
A Matlab model was produced which gets data from the Raspberry Pi via UDP network connection. Just a network cable needs to be connected between the host computer and the Raspberry Pi. On the Raspberry Pi is a C library used which enables the communication. With the help of this part the implementation can be tested and the configuration of the filter parameters can be improved.
Finally a doxygen file was written for automatic code documentation.

14.2 Remaining project goals and outlook

The basis of the PPM measurement is provided via a Kernel module. The analysis of the measured time differences needs to be done to get the separated control signals.
The orientation fusion delivers perfect representation of all three angles around the X,Y and Z axis of the system. Because of the usage of Euler angles there can be a gimbal lock when due to rotations two of the three axis fall together. Then one degree of freedom is lost. To solve this problem the system should not use Euler angles, instead Quaternion needs to be used.
Autonomous flying with just the orientation is not possible. Also the position of the Quadrocopter needs to be known. For this, additional sensor fusion focusing the velocity and position in all directions has to be calculated.
Last step is the implementation of the controller for the orientation and the position which finally controls the complete system. To switch between flying with the remote control and the autonomous flying a switch of the remote control needs to be used.

15 Grundlagen

Motoren

Bei den Motoren handelt es sich um über Treiber "BL-Ctrl V1.2"gesteuerte Brushless DC Motoren "Robbe ROXXY BL-Outrunner 2824-34". Die Ansteuerung des Treibers erfolgt über den I2C Bus. An den Treiber werden 2 Byte Daten gesendet:

- die I2C Adresse des gewünschten Motors
- den PWM-Wert (kleiner Wert langsam drehen, hoher Wert schnelles drehen)

Der Treiber behält die Werte 10 msec lang und verliert diese anschließend. Die Werte müssen also zyklisch gesendet werden.

Die Anzahl und Ordnung der Motoren ist je nach Typ des Helikopters unterschiedlich. Es gibt vier- oder acht Motorvarianten welche verschiedene Grundaufbauten haben können (+ oder x).

Der Code soll so allgemein gehalten werden, dass er für jeden dieser Typen verwendet werden kann. In der main.h wird mittels Definition angegeben für welchen Helikopter das Programm definiert werden soll. In der */hal/MOTOR/MOTOR.h* sind helikopterspezifische Daten abgelegt, darunter die I2C-Adressen jedes Motors, die Anzahl der Motoren, die min-Werte und max-Werte des PWM-Signales.

Adafriut Ultimate GPS PI HAT[3]

Dieses Modul wird direkt auf das PI aufgesteckt und leitet alle Pins bis auf die UART Pins weiter. Folgende Pins werden für das Bord benötigt und können nicht mehr verwendet werden:

Pin	Verwendung	Fest
UART TXD ; UART RXD	Die einzige serielle Schnittstelle muss verwendet werden um mit dem GPS-Modul zu kommunizieren	
GPIO #4	Kann bei Bedarf verwendet werden, falls die Zeitsynchronisation mit dem Raspberry nicht benötigt wird	
EEDATA ; EECLK	Werden für die Verbindung mit dem EEPROM benötigt, derzeit noch nicht vom Raspberry verwendet	

Tab. 15.1: Ultimate GPS Pins

Dieses Modul liefert das GPS-Modul *FGPMMOPA6H* (Datasheet). Das Modul bietet Platz für eine Batteriezelle, für eine Real-Time-Clock und Prototypen Platz, um weitere Bauteile mittels Löten hinzuzufügen.

ADS1015 12-Bit ADC[4]

Hierbei handelt es sich um einen hochauflösenden 12bit Analog-Digital-Konverter. Der Konverter kann bis zu 3300 Umwandlungen in der Sekunde berechnen und besitzt eine I2C-Schnittstelle. Die Standardadresse des Bausteines ist die 7bit-Adresse 0x47. Diese Adresse kann durch die Verbindung folgender Pins mit dem *ADDR-Pin* bei Bedarf angepasst werden:

Adresse	Pin 1	ADDR
0x48	GND	ADDR
0x49	VDD	ADDR
0x4A	SDA	ADDR
0x4B	SCL	ADDR

Tab. 15.2: ADC 12 Bit I2C Adress Manipulation

Dieser ADC liefert folgende zwei verschiedene Betriebsmodi, welche über die Anschlüsse *A0* bis *A3* verwendet werden:

- *Single Ended*: berechnet den ADC der Spannung zwischen jedem der *Ax-Pins* und Ground. Hiermit ist nur das Messen von positiven Spannungen möglich. Effektiv verliert man dadurch ein Bit der Auflösung. Der Modus bietet doppelt so viele Inputs.
- *Differential*: berechnet den ADC der Spannung zwischen *A0* & *A1* und *A2* & *A3*. Der Modus liefert weniger rauschanfällige Signale.

Eine Spannung über 5V darf nicht angelegt werden, da diese das Modul zerstören würde.

Pololu AltIMU-10 v4[5]

Dieses Modul besteht aus drei IC´s. Sie beinhalten die folgenden Funktionen:

Funktion	IC	Default I2C Adresse	Manipulate I2C
Beschleunigungsmesser Magnetometer	LSM303D	0x1D	0x1E
Gyrometer	L3GD20H	0x6B	0x6A
Barometer	LPS25H	0x5D	0x5C

Tab. 15.3: IMU ICs

Auch dieser Baustein verwendet zur Konfiguration und Kommunikation den I2C-Bus (7bit Adresse). Jeder dieser ICs besitzt eine eigene Adresse, die sich ebenfalls manipulieren lässt, indem man den Pin *SA0* auf *GND* zieht.

LIDAR-Lite v2[6]

Dieser Lasersensor besitzt eine Reichweite bis zu $40m$ mit einer Genauigkeit von $\pm$0,025 , eine Verzögerung von 0,02 Sekunden und kann bis zu 500 Messwerte pro Sekunde liefern. Der I2C-Bus kann mit $100kbits/s$ oder $400kbits/s$ betrieben werden. Eine eigene Adressierung ist möglich. Als Standard-I2C Adresse ist die 0x62 festgelegt. Falls mehrere LIDAR Light v2 Sensoren an einem I2C-Bus hängen ist dies die Broadcast-Adresse.

ï»¿

16 Realisierung

Der Remoteempfi¿½nger GR-16 hat drei Verbindungen zum Raspberry Pi: Eine 3,6V bis 8,4V Spannungsversorgung, Ground und eine Datenverbindung.

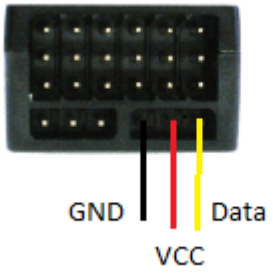


Abb. 16.1: GR-16 Verkabelung[2]

Die Spannung und Ground kann direkt i¿½ber das Raspberry Pi bezogen werden. Fi¿½r die Daten muss einer der GPIO Pins als Input deklariert werden.

Um die Fernbedienung mx-20 mit dem Empfi¿½nger GR-16 zu verbinden mi¿½ssen beide Geri¿½te eingeschaltet sein (Empfi¿½nger LED blinkt Rot). Beim Anschalten der Fernbedienung wird gefragt ob *HF* EIN oder AUS ist. Stellen Sie auf AUS und besti¿½tigen Sie mit Dri¿½cken der Set-Taste. Dri¿½cken sie erneut Set um in die Einstellungen zu gelangen. Scrollen Sie sich mit den Pfeiltasten durch das Meni¿½ bis Sie das Meni¿½ *Grundeinstellungen Mod.* sehen. i¿½ffnen Sie mit Set das Meni¿½ und scrollen Sie durch bis zu Punkt *Modul.* Besti¿½tigen Sie noch NICHT mit der Set-Taste. Halten Sie nun am Empfi¿½nger solange die Set Taste gedri¿½ckt, bis sich zur der roten LED noch eine gri¿½ne LED einschaltet. Beti¿½tigen Sie jetzt den Set-Taster auf der Fernbedienung. Es sollte der Info-Text *Binden ...* angezeigt werden. Wenn die Verbindung erfolgreich durchgefi¿½hrt wurde leuchtet die LED dauerhaft gri¿½n. [1]

2 http://www.graupner.de/mediaroot/files/33508_Kurzanleitung_de.pdf

16.1 Umstellung auf eine automatische dynamische Testumgebung

Stattdessen soll die neue Main-Funktion ï¿½ber eine Dauerschleife verfiï¿½gen, in dieser ein Textfile ausgelesen wird. Dieses File beinhaltet die zu startenden Testcases des Programmes.

In diesem Textfile werden alle Testfiï¿½lle festgehalten, mit einem Status ob dieser laufen soll oder nicht. Der Status soll wï¿½hrend der Laufzeit des Programmes ï¿½nderbar sein und dieses auch nicht unterbrechen.

Dieses Textfile sieht folgendermaïï¿½en aus:

...
testmotorpwm=1
testmotorisr=1
testmotortxt=0
....

Dies bedeutet: die Testfiï¿½lle testmotorpwm und testmotorisr sollen gestartet werden, testmotortxt soll nicht gestartet werden. Der Testfall testmotorpwm wird zuerst ausgelesen und von der Main-Funktion auf Null zurï¿½ckgesetzt. Anschlieïï¿½end lï¿½uft der Testfall ab.

Wï¿½hrenddessen wird der Testfall testmotortxt aktiviert.

...
testmotorpwm=0
testmotorisr=1
testmotortxt=1
....

Ist der Testfall beendet wird das Textfile erneut eingelesen und der nï¿½chste gesetzte Testfall wird gestartet.

Umstrukturierung der main.c: Es wurde eine Endlosschleife eingefi¿½gt. Statt auf eine Tastatureingabe i¿½ber eine Konsole zu warten, wird in dieser alle zwei Sekunden das *_Testfile* ausgelesen und i¿½berpri¿½ft, ob ein Testfall ausgefi¿½hrt werden soll.

In allen Testfi¿½llen darf keine Endlosschleife vorhanden sein, bzw. muss in weiteren Endlosschleifen ein Abbruchkriterium festgelegt sein. In den meisten Fi¿½llen kann der Testfall mit Dri¿½cken der Taste *p* abgebrochen werden. In Testfi¿½llen, in denen ein Testfile ausgelesen wird, wird bei Erreichen des Dateiendes der Testfall beendet und gelangt so wieder in die Hauptschleife zuri¿½ck.

Script zum Setzen der Testfi¿½lle: Zum Setzen oder Li¿½schen eines Testfiles wurde ein Script geschrieben, welches auf das textfile *_Testfile* zugreift und die Eintri¿½ge je nach Eingabe i¿½berarbeitet.

Verwendet wird das Script folgendermai¿½en:

```
1 .\testcase NameTestfall [Modus]
```

Als ersten Parameter ist der zu i¿½berarbeitende Testfall anzugeben.
Der Modus ist ein optionaler Parameter. Ist dieser Parameter nicht vorhanden, wird automatisch *set* angenommen. *set* markiert den Testfall als abzuarbeiten, *clear* li¿½scht diese Markierung wiederrum. Das Script zeigt keinen Fehler oder Warnung auf, wenn der testcase nicht gelistet ist.

Der Quellcode ist im Kapitel **?? ??** auf S.**??** einzusehen.

16.2 Software Motortreiber

Es soll Software fi¿½r den Hardware-Treiber, der die Motoren ansteuert, geschrieben werden. Zuni¿½chst muss der Raspberry Pi mit dem I2C-Bus verbunden werden. Die I2C Adressen der Motoren sind in der Motor.h hinterlegt. Fi¿½r die Quadrocopter wurden die Werte definiert. Die Werte des Octocopter mi¿½ssen zuni¿½chst i¿½berpri¿½ft werden.

Mittels des Befehl:

Die Daten, die von dem Hardware Brushless-Controller erwartet werden, besitzen folgendes Format:

I2C Adresse [1 Byte]	PWM Value [1 Byte]

Tab. 16.1: I2C Frame Brushless Motoren Treiber

Mit Hilfe des Befehls

i2cset -y 1 0x29 0x55

wird an den Controller mit der I2C Adresse 0x29 (Motor Nr.1) der Wert 0x55 gesendet.

16.2.1 Verwendung des Software-Treibers

In der main.h muss zunï¿½chst definiert werden, auf welchem Typ von den HElicoptern das Programm geladen werden soll.

```
#include <time.h>
#define Quadro_Plus 1
//#define Quadro_X 1
//#define Okto_Plus 1
```

Anschlieï¿½end sollte baldmï¿½glichst die InitMotor() aufgerufen werden, die unter anderem einen Timer initialisiert und startet. Bei Ablauf des Timers wird ein Flag gesetzt. Dieses Flag muss im Quellcode mit der Funktion *GetFlagRunSendPwmToMotor()* abgefragt werden. Wenn dieses Flag gesetzt ist, muss die Funktion *sendPwmToMotor()* aufgerufen werden.

```
...
InitMotor();
...
while(1){
...
if(GetFlagRunSendPwmToMotor() == 1){
sendPwmToMotor();
}
...

}
```

Der aktuelle PWM-Wert eines Motors kann mittels der Funktion *GetPwmMotor(...)* zurï¿½ckgegeben werden.

```
value = GetPwmMotor(6);
value > 0? value--: (value=DEFMotorSetpointMIN);
SetPwmMotor(DEFMotorNo7_PWM, value ,0);
```

Wichtig: Alle ISR sollen knapp gehalten werden, da ansonsten die Motoren nicht mehr angesprochen werden kï¿½nnen.

Mit der Funktion *SetPwmMotor(...)* kï¿½nnen die PWM-Werte, die per I2C gesendet werden, ï¿½berschrieben werden. Optional kann ein Flag gesetzt werden. Ist dies der Fall wird anschlieï¿½end die Funktion *sendPwmToMotor()* aufgerufen.

16.2.2 Headerfile

Hier sind die verschiedenen HElicopter Varianten sowie deren definierte Eigenschaften (Anzahl Motoren, Drehrichtung der Motoren, Motorenreihenfogen) festgehalten.

Fï¿½r weitere Informationen (wie z.B. Namen der defines) siehe in Kapitel **??** in **??** auf S.**??** .

16.2.3 Funktionen

Beschreibung der Funktionen befinden sich in den jeweiligen darï¿½ber liegenden Kommentaren mit Parametern und Return Values.

Auf alle globalen Variablen/Flags werden mit Funktionen zugegriffen. Ein direkter Zugriff ist zu vermeiden.

16.2.4 Testfiï¿½lle

Um die Funktionalitiï¿½t des Treiber zu testen und kontrollieren sind die folgenden Testfiï¿½lle geschrieben worden.

TESTMOTORPWM Dieses Testprogramm sendet alle 10ms an die Motoren einen stets steigenden PWM-Wert bis zum Wert 0x50. Bei erreichen des Wertes wird der PWM-Wert auf den definiertes Minimum gesetzt. Dieser Testfall liï¿½uft ohne ISR ab. Die Daten werden direkt iï¿½ber I2C gesendet. Die Schrittweite und das Maximum des Testfalles sind per Konstanten definiert und kiï¿½nnen veriï¿½ndert werden.

TESTMOTORISR Mit diesem Testprogramm wird die ISR-Funktion getestet. Durch Eingabe in der Konsole wie z.B.

+0+0+0+0+7+7+7-8

wird der PWM-Wert des Motors 0 um vier erhiï¿½ht, der Motor 7 um drei erhiï¿½ht und der Motor 8 um eins verringert. Die Eingabe ist nicht blockierend. Zum Starten des Testfalls muss ein '+' eingetippt werden.

TESTMOTORTXT Der letzte Testfall liest aus einem Textfile, das sich auf dem Raspberry unter dem Pfad */home/pi/MotorTest.txt* befinden muss, Zeile fi¿½r Zeile aus und setzt die PWM-Werte so, wie sie in der Zeile angegeben sind.

Die Befehlszeile hat folgendes Format:

#MOTORNUMER[+][-][=][PWMWERT] DELAY

z.B. #0+100;10 ⌢D̂er PWM des Motors 0 erhï¿½ht sich um 100. Die nï¿½chste Zeile wird in 10s eingelesen.

16.3 Remotecontroller-Treiber

Um den autonomen Start bzw. Landevorgang des Quadrocopter einzuleiten muss eine Funkverbindung ï¿½ber eine Fernbedienung und einen Transmitter eingerichtet werden. Die Daten sind anschlieï¿½end auszuwerten und zu verarbeiten.

Als Fernbedienung von der Firma Graupner dient der *MX-20* und als Receiver der *GR-16*. Der Anschluss des Receivers erfolgt ï¿½ber drei Leitungen:

- Versorgungsspannung: 5Volt (vom Raspberry)
- Ground (Raspberry)
- PPM (Puls-Pause Modulation) (auf einen der GPIO Pins)

Verbinden mit Receiver Schalten Sie die Fernbedienung ein und schlieï¿½en den Receiver an den Raspberry Pi an [siehe Abschnitt 16 auf Seite 93]

PPM-Signal genannt Puls-Pausen-Modulation (oder auch Puls-Position-Modulation) ist ein fiï¿½r analoge Werte verwendetes Kodierungsverfahren und wird vor allem in Funkfernsteuerungen verwendet. Der zu kodierende Wert wird in der Liï¿½nge des Pausen/Low Signals zwischen zwei Peaks/High Signalen gesendet. Diese Peaks haben stets die gleiche Liï¿½nge als auch gleiche Amplitude.[2]

Meist besteht ein PPM-Signal aus mehreren Kanïï¿½len zu einem Frame zusammengefiï¿½gt und anschlieï¿½end versendet.

Fiï¿½r die Decodierung des PPM-Signals ist es empfehlen, ein weiteres Board zu integrieren, das die Decodierung iï¿½bernimmt und die Ergebnisse an das Raspberry Pi weiterleitet, da das Signal sehr genau aufgeliï¿½st werden muss und die Decodierung iï¿½ber einen Interrupt gesteuert werden soll. Bei zu hiï¿½ufigen Auftreten des Interrupts wiï¿½rde der Helicopter destabilisiert werden. Zur Auswertung miï¿½ssen nur die Raising Edges oder Falling Edges beachtet und der Offset des High-Pegels abgezogen werden.

Um auf ein weiteres Bauteil zu verzichten wurde stattdessen eine Real-Time-Linux-Version auf dem Raspberry Pi installiert, die die Signale im Nanosekundenbereich dekodieren kann.

PPM Signal Mitschnitte Der Receiver sendet die empfangenen Signale im Format der Pulse-Pausen-Modulation. Mitgeschnittene ï¿½bertragungen[3]mit Beschreibung sind beigefi¿½gt.

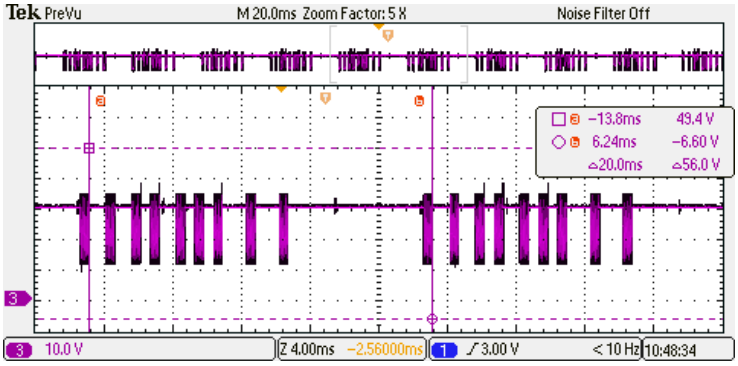


Abb. 16.2: 8 Channel Frame

Es gibt zwei Mï¿½glichkeiten: ein Frame mit acht Kanï¿½len, bestehend aus neun Peaks und acht Pausensignalen...

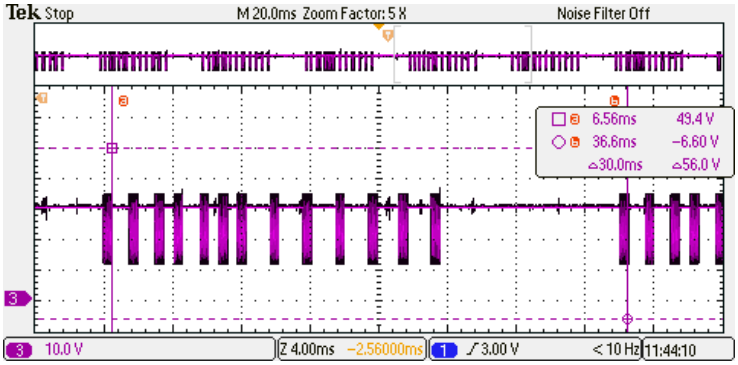


Abb. 16.3: 12 Channel Frame

... oder einen 12 Kanï¿½len Frame, bestehend aus 13 Peaks und 12 Pausensignalen.

3 Von Herrn Trybek bereitgestellt

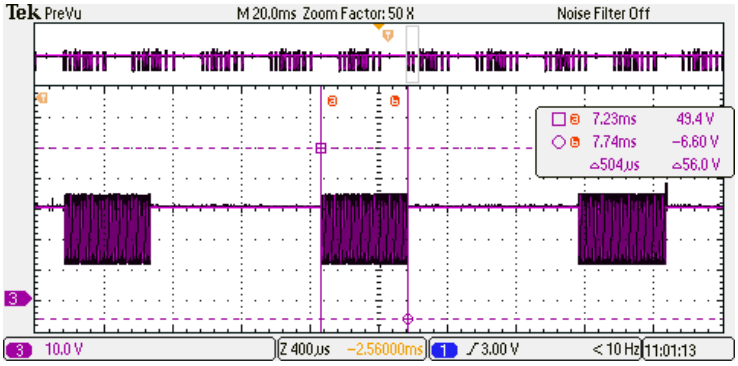


Abb. 16.4: Dauer 8 Chanel Frame

Folgend die Anschaung eines Frames mit acht Kanï¿½len mit der Dauer von 504us.

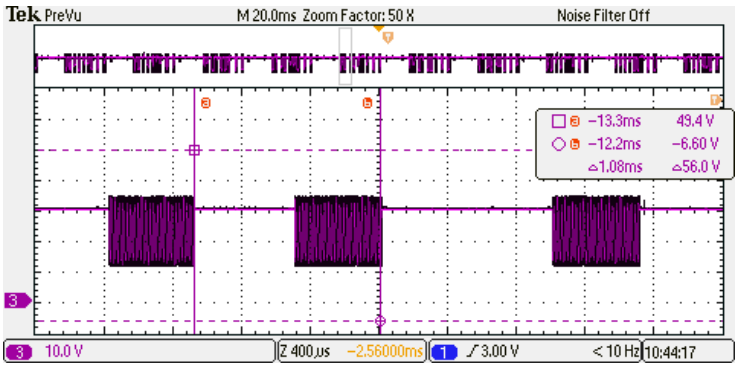


Abb. 16.5: Minimale Pause zwischen Frames

Zu sehen ist hier der kleinstmï¿½gliche zeitliche Abstand zwischen zwei Frames mit der Dauer von nur 504us.

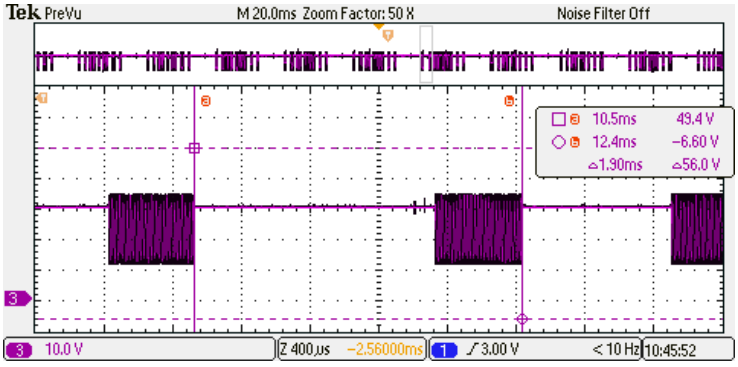


Abb. 16.6: Maximale Pause zwischen Frames

Hier ein Ausschnitt mit dem grï¿½ï¿½tmï¿½glichen Abstand zwischen zwei Frames mit der Dauer 1,396ms.

16.4 Hardware Layout

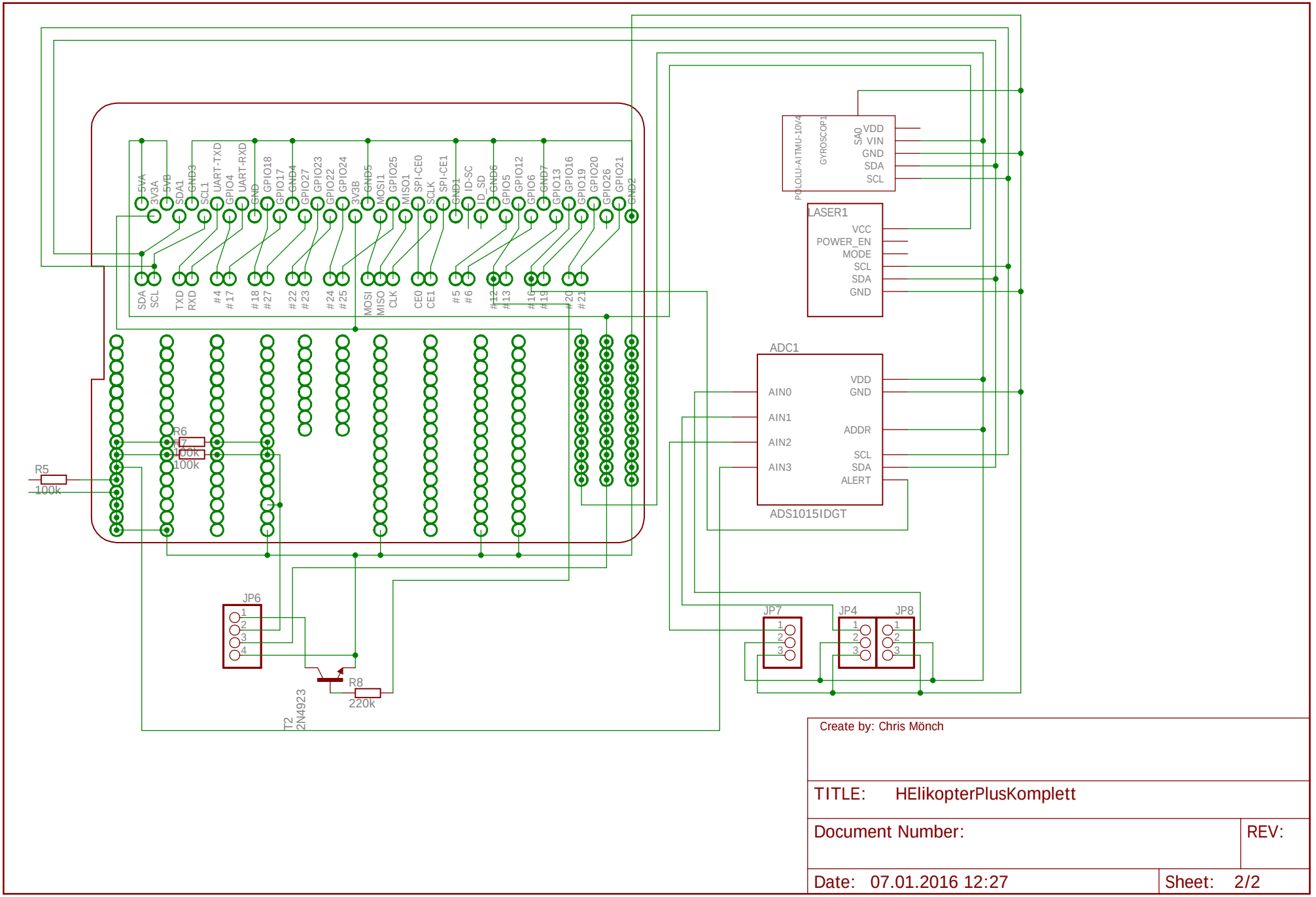
Fï¿½r die schnelle Durchfi¿½hrung von i¿½nderungen, Erweiterungen oder Anpassungen an der Hardware werden die Leiterplatinen und Schaltplï¿½ne mittels Tool elektronisch festgehalten. Die dazu mï¿½gliche Software sind: EAGLE oder Fritzing, die kostenlos oder als eingeschrï¿½nkte Freeversion angeboten werden, aber diversen Einschrï¿½nkungen unterliegen. Beide Tools haben eine Community, die viele Bauteile in Liberias bereitstellt.

Im direktem Vergleich macht EAGLE einen professionelleren Eindruck und bietet eine Sammlung von Tutorials und Libraries. Aus diesen Grï¿½nden fiel die Entscheidung auf die EAGLE SOFTWARE. Es sollte aber mit dem Platz mï¿½glichst sparend umgegangen werden, da bei dieser Freeeware die Grï¿½ï¿½e und Anzahl der Layouts begrenzt ist.

16.4.1 HElicopter Schematic Layout

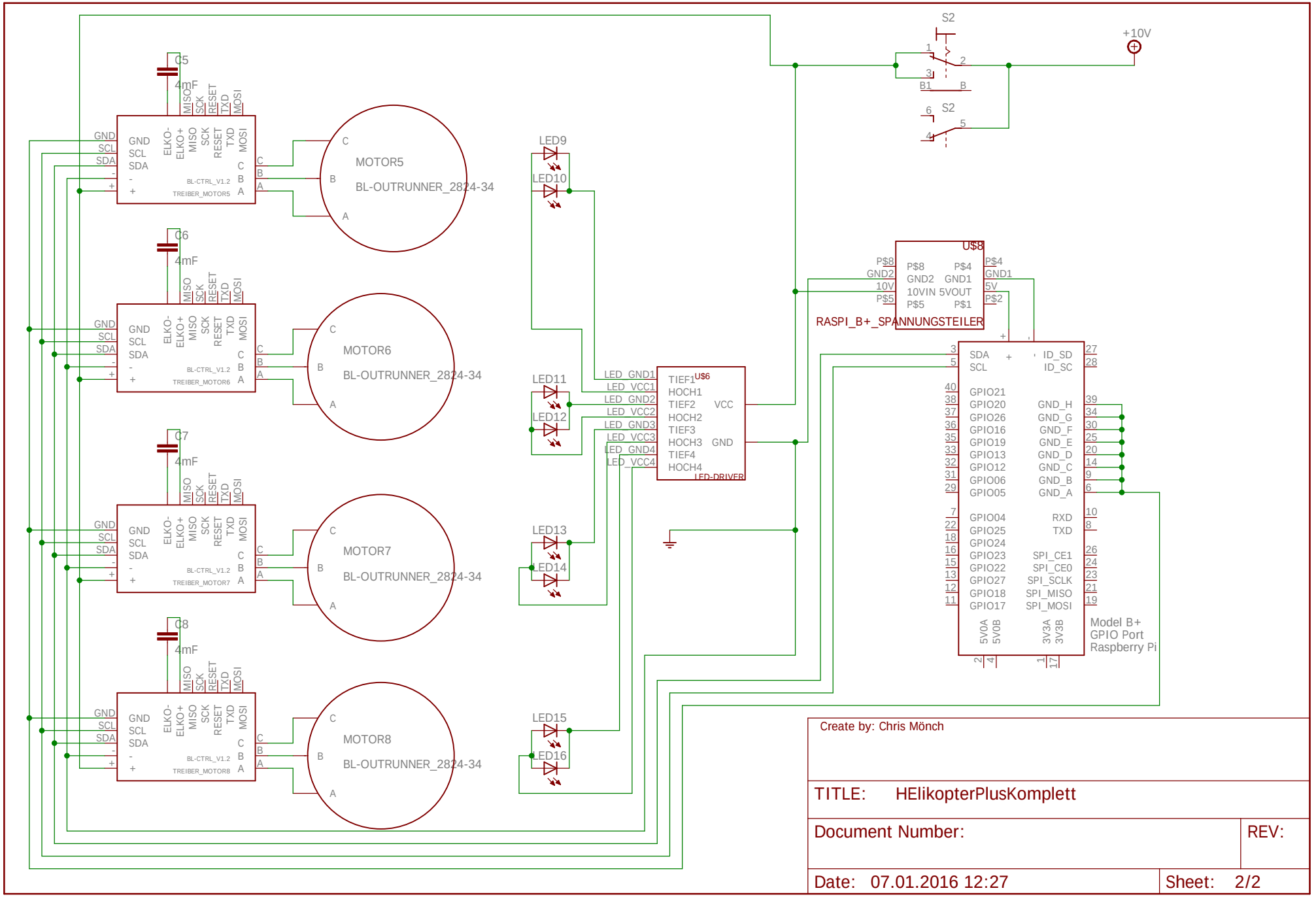
Es wurde fï¿½r die schematischen Zeichnungen ein EAGLE Projekt angelegt, dass die schematischen Zeichnungen aller selbst gebauten bzw. zusammengefï¿½gten Bauteile enthï¿½lt. Das Projekt File ist im Verzeichnis *trunk/hardware/* unter *HElicopterPlus* abgespeichert.

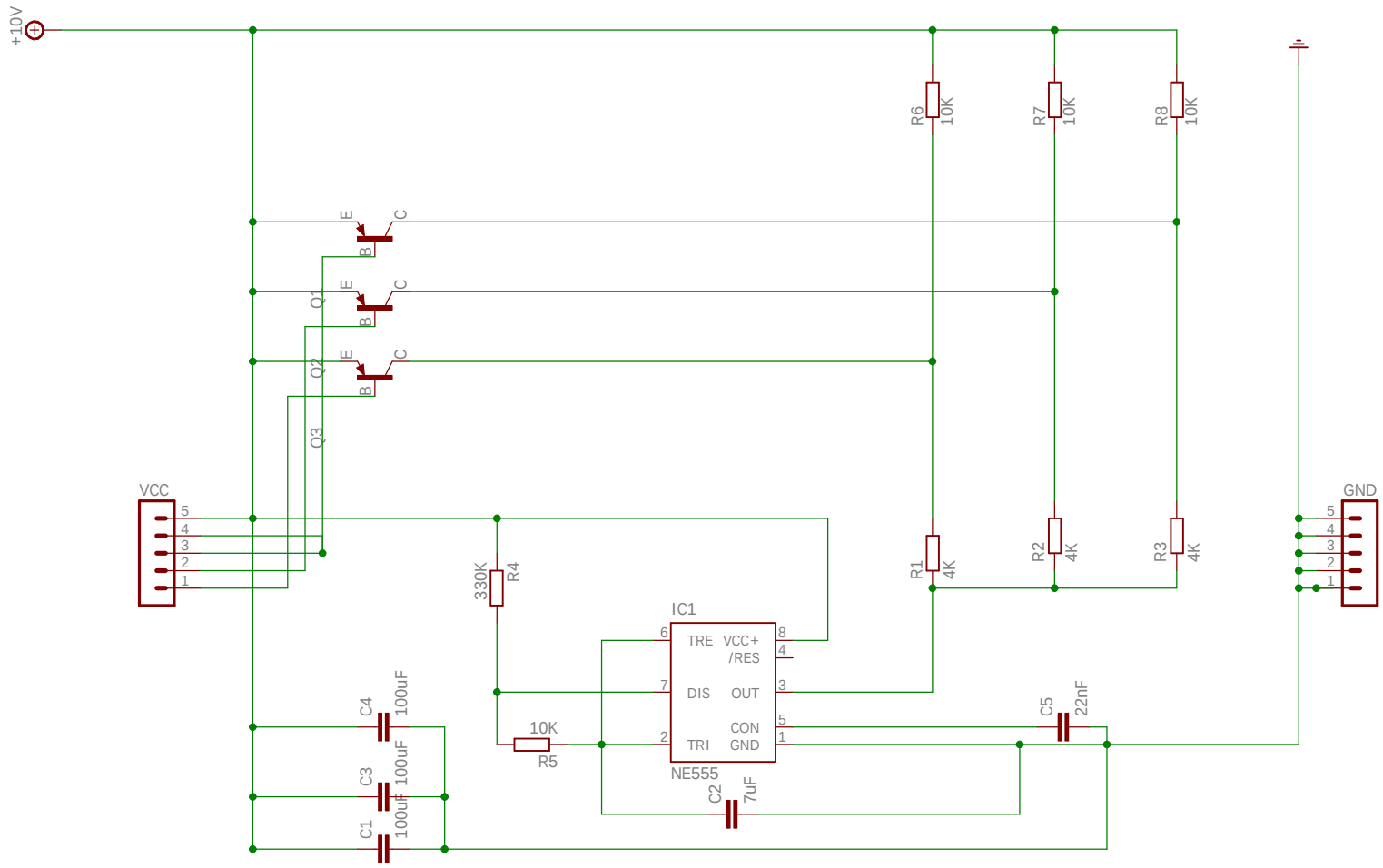
Derzeit befinden sich zwei verschiedene schematischen Zeichnungen im Projektverzeichnis, die maximal aus zwei sogenannten Sheets besteht (Begrenzung durch die Freeware-Version von EAGLE). Der erster Sheet beinhaltet immer, denn Komplettaufbau des jeweiligen Systems. Der zweite Sheet wird verwendet um druckbare Versionen das Schaltplans zu erstellen (DINA4 Frame). Hierbei ist es wichtig, eine ï¿½bersichtliche Abspaltung vorzunehmen.



Create by: Chris Mönch

TITLE: HElikopterPlusKomplett	
Document Number:	REV:
Date: 07.01.2016 12:27	Sheet: 2/2


S2
+10V
C5
4mF
C6
4mF
C7
4mF
C8
4mF
MISO SCK RESET TXD MOSI
GND SCL SDA - +
ELKO- ELKO+ MISO SCK RESET TXD MOSI
C B A
BL-CTRL_V1.2
TREIBER_MOTOR5
TREIBER_MOTOR6
TREIBER_MOTOR7
TREIBER_MOTOR8
MOTOR5
MOTOR6
MOTOR7
MOTOR8
BL-OUTRUNNER_2824-34
LED9
LED10
LED11
LED12
LED13
LED14
LED15
LED16
LED_GND1
LED_VCC1
LED_GND2
LED_VCC2
LED_GND3
LED_VCC3
LED_GND4
LED_VCC4
TIEF1
HOCH1
TIEF2
HOCH2
TIEF3
HOCH3
TIEF4
HOCH4
VCC
GND
U$6
LED-DRIVER
U$8
P$8 GND2 10V P$5
P$4 GND1 5V P$2
10VIN 5VOUT
RASPI_B+_SPANNUNGSTEILER
SDA SCL
ID_SD ID_SC
GPIO21 GPIO20 GPIO26 GPIO16 GPIO19 GPIO13 GPIO12 GPIO06 GPIO05
GND_H GND_G GND_F GND_E GND_D GND_C GND_B GND_A
GPIO04 GPIO25 GPIO24 GPIO23 GPIO22 GPIO27 GPIO18 GPIO17
RXD TXD
SPI_CE1 SPI_CE0 SPI_SCLK SPI_MISO SPI_MOSI
5V0A 5V0B 3V3A 3V3B
Model B+
GPIO Port
Raspberry Pi
Create by: Chris Mönch
TITLE: HElikopterPlusKomplett
Document Number:
REV:
Date: 07.01.2016 12:27
Sheet: 2/2



R1,R2,R3 = 4K = gelb schwarz schwarz braun braun
R4 = 330K = orange orange schwarz orange braun
R5 = 10K = braun schwarz schwarz rot braun
R6,R7,R8 = 10K (SMD)

Created by: Chris Mönch	
TITLE: LED-DRIVER	
Document Number:	REV:
Date: 07.01.2016 12:07	Sheet: 1/1

Richtlinien[7]

- Platzierung von mindestens einem Frame in jedem Sheet
- Verwendung von "CommonSSymbolen
- Jedem Bauteil einen Wert zuordnen (wenn mï¿½glich)
- Fï¿½r Verbindungen ausschlieï¿½lich *Net* verwenden (nicht *Wire*)
- Verbindungen oder ï¿½berbrï¿½ckungen wie in Abb. 16.7 dargestellt
- Abkï¿½rzungen direkt in dem jeweiligen Sheet festhalten
- Name und Bauteile immer in eine Richtung flieï¿½en lassen (wenn mï¿½glich)
- ï¿½berbrï¿½cken von Netzen mï¿½glichst gering halten
- Fï¿½r groï¿½e schematische Skizzen immer auch druckbare Versionen in DINA4 Format erstellen
- Wichtigen Netzen immer einen Namen zuordnen (GND, +5V, SDA, SLC, ...)
- Namen und Bezeichnungen kurz halten

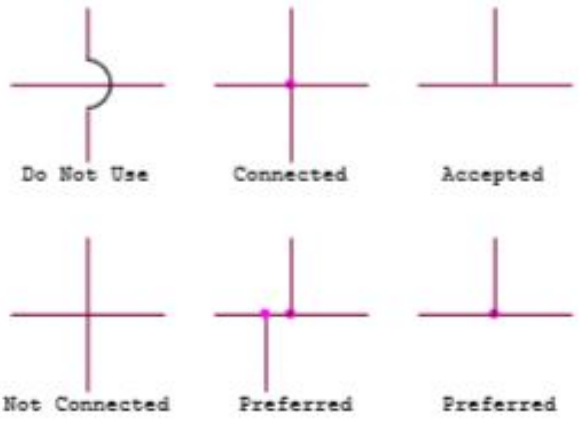


Abb. 16.7: Net verbinden/ï¿½berbrï¿½cken[4]

4 http://www.k-state.edu/ksuedl/publications/Technote%208%20-%20Guidelines%20for%20Drawing%20Schematics.pdf

16.4.2 HElicopter Libary

Auf der Bibliothek von Eagles kï¿½nnen unter der Rubrik *Downloads* Bauteile zu denn Schaltplï¿½nen gefunden werden. Die dort vorhandenen Bibliotheken sind von Usern bereitgestellt worden. Deshalb besteht die Mï¿½glichkeit, dass Bauteile fehlerhaft sind.

Wenn in der Bibliothek kein Element fï¿½r das Bauteil gefunden werden konnte, muss dieses erstellt und in der bereits vorhandene Libary *HElicopterLib.lbr* hinzugefï¿½gt werden. In dieser Libary sind alle Teile festgehalten, die fï¿½r die Quadrocopter benï¿½tigt werden.

Ein Bauteil oder auch ein Device besteht aus insgesamt drei eigenen Bestandteilen, die alle vorhanden sein mï¿½ssen, um das Bauteil zu verwenden:

Symbol	Darunter versteht man eine schematische Skizze eines Bauteils. Alle Schnittstellen des Bauteiles sind zu erkennen und ggf. zu kennzeichen. Der Orignalmaï¿½stab und Echtheit der Pinanordnung spielt keine Rolle. Diese soll mï¿½glichst so platziert werden, um im spï¿½teren Layout eine gewisse ï¿½bersichtlichkeit zu wahren. Die Namen der Pins sollen so benannt werden, wie diese vom Hersteller in den Datenblï¿½ttern festgelegt ist. Ansonsten kï¿½nnte es Spï¿½ter Probleme zur Pin-Zuordnung geben Die Schlï¿½sselwï¿½rter $>NAME$ und $>VALUE$ sollen sich (wenn mï¿½glich) im Symbol befinden.
Package	In dieser Skizze soll eine realitï¿½tsnahe Abbildung des Bauteiles erstellt werden, mit original Abmessungen wie z.B. Platinengrï¿½ï¿½e, Platinenform, Pins und Bohrlï¿½cher, die aus den jeweiligen Datenblï¿½ttern ersichtlich sind. Weitere Ergï¿½nzungen sind sonstige Bauteile oder SMD auf einem Device sowie deren Verbindungen auf Ober- bzw. Unterseite. Diese werden derzeit nicht benï¿½tigt und wurden deshalb weggelassen.
Device	In einem Device werden ein Symbol und ein Package miteinander zusammengefï¿½gt. Die Pins aus dem Symbol und Package Skizzen werden miteinander verknï¿½pft. Wenn jeder Pin zugeordnet wurde kann das Device verwendet werden.

Der Vorteil dieser Aufteilung ist der, dass einzelne Packages und Symbole fï¿½r mehrere Devices verwendet werden kï¿½nnen. Nur die Anzahl der Pins aus den Packages und Symbolen mï¿½ssen eindeutig ï¿½bereinstimmen und zugewiesen werden.

Um bereits existierende Libaries hinzuzufi¿½gen wird die gewï¿½nschte Libary in dem Ordner EAGLE-7.5.0/*lbrName* gespeichert. Anschlieï¿½end klickt man im Reiter *Bibliothek* auf *Benutzen* und wï¿½hlt die gewï¿½nschte Libary aus. Ein Schaltplan muss hierbei geï¿½ffnet sein. Im *Controll Panel* ist das Hinzufi¿½gen von libaries nicht mï¿½glich. Zuletzt mï¿½ssen die libaries noch aktualisiert werden(unter *Bibliothek* auf die Rubrik *Alle Aktualisieren*). Abschlieï¿½end auf *Add/Neues Bauteil hinzufi¿½gen*, es erscheint die Libary in der Liste.

Um ein vorhandenes Bauteil in die eigene Libary einzufi¿½gen muss die Ziel-Libary geï¿½ffnet sein. Im *Control Panel* von EAEGLE(in der linken Spalte das Menï¿½ *Bibliotheken* ï¿½ffnen). Hier sollten sich alle bereits hinzugefi¿½gten Libaries befinden. Ist dies nicht der Fall: rechtsklick auf die *Bibliotheken* und *alle Bibliotheken laden* auswï¿½hlen. Nun muss die Quell-Libary des Bauteils geï¿½ffnet werden.Das zu kopierende Bauteil rechts klicken und *In Bibliothek* auswï¿½hlen. Es ist auch mï¿½glich, einzelne Packages oder Symbole neben ganzen Bauteilen/Devices zu kopieren. Nun sollte das hinzugefi¿½gte Objekt in der Ziel-Libary geï¿½ffnet sein. Bestï¿½tigen sie das Hinzufi¿½gen mit Speichern der Libary.

Fï¿½r das Bearbeiten existierender Libaries ï¿½ffnen sie eine schematische Skizze. Klicken sie auf *Bibliothek*, dann *ï¿½ffnen..* und wï¿½hlen sie die zu ï¿½berarbeitende Libary aus. Nun kï¿½nnen alle existierende Bauteile, Packages und Symbole bearbeitet oder neue erstellt werden. Nach der ï¿½nderung speichern Sie die Libary und klicken sie auf wieder auf Bibliothek. Anschlieï¿½end aktualisieren Sie die Libaries. Die ï¿½nderungen sollten nun vorgenommen sein.

Achtung: ï¿½nderungen werden auf alle bestehenden und eingefi¿½gten Elementen vorgenommen. Dies kann sich z.B. unvorteilhaft auf die Lesbarkeit auswirken. Aus diesem Grund sollte zumindest eine Kopie des geï¿½nderten Devices oder eine neue Version erstellt werden.

Literaturverzeichnis

[BCM] BCM2835 ARM Peripherals; Broadcom Corporation, 2012 https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf

[BorEng] Alex, The quadcopter : control the orientation, 2012, http://theboredengineers.com/2012/05/the-quadcopter-basics/

[LMK69] Jürgen Quade, Eva-Katharina Kunst; Kern-Technik: Kernel- und Treiberprogrammierung mit dem Linkux-Kernel (Folge 69); Linux Magazin 08/2013 http://www.linux-magazin.de/Ausgaben/2013/08/Kern-Technik

[LMK70] Jürgen Quade, Eva-Katharina Kunst; Kern-Technik: Kernel- und Treiberprogrammierung mit dem Linkux-Kernel (Folge 70); Linux Magazin 10/2013 http://www.linux-magazin.de/Ausgaben/2013/10/Kern-Technik

[LMK81] Jürgen Quade, Eva-Katharina Kunst; Kern-Technik: Kernel- und Treiberprogrammierung mit dem Linkux-Kernel (Folge 81); Linux Magazin 07/2015 http://www.linux-magazin.de/Ausgaben/2015/07/Kern-Technik

[REP] Robotics/Electronics/Physical Computing; Wordpress Blog, https://trandi.wordpress.com/2011/04/12/graupner-r700-ppm-signal/

[RPL] Robot Platform; Wordpress Blog http://www.robotplatform.com/knowledge/servo/servo_control_tutorial.html

[STM] Jay Esfandyari, Roberto De Nuccio, Gang Xu; STMicroelectronics, http://uk.mouser.com/applications/sensor_solutions_mems/

Literaturverzeichnis

[1] Graupner/SJ Gmbh, Bedienungsanleitung Graupner HoTT 2.4, http://www.graupner.de/mediaroot/files/33508_Kurzanleitung_de.pdf; *Januar 2011 DE V1.3*

[2] Herbert Bernstein, Informations- und Kommunikationselektronik, Walter de Gruyter GmbH, 2015, *1. Auflage*

[3] Lady ada/Adafruit Industries, Adafruit Ultmate GPS HAT for Raspberry Pi, https://learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-hat-for-raspberry-pi.pdf, *2016-01-11*

[4] Bill Earl/Adafruit Industries, Adafruit 4-Channel ADC Breakouts, https://learn.adafruit.com/downloads/pdf/adafruit-4-channel-adc-breakouts.pdf, *2014-11-30*

[5] Pololu Robotics & Electronics, AAltIMU-10 v4 Gyro, Accelerometer, Compass, and Altimeter (L3GD20H, LSM303D, and LPS25H Carrier), https://www.pololu.com/product/2470, *2016-01-13*

[6] EXP Tech, LIDAR-Lite v2, http://www.exp-tech.de/lidar-lite-v2, *2016-01-13*

[7] Olin Lathrop, Rules and guidelines for drawing good schematics, http://electronics.stackexchange.com/posts/28255/revisions, *2016-01-14*

[Gun04] Karsten Günther, LaTeX2 — Das umfassende Handbuch, Galileo Computing, 2004, http://www.galileocomputing.de/katalog/buecher/titel/gp/titelID-768; *1. Auflage*