

# **Entwicklung eines Schwingungsmesssystems für ein Radar**

**Bachelorarbeit**

vorgelegt von Sebastian Beyer

Institut für Geophysik  
Universität Hamburg

September 2012



## **Kurzfassung**



# Inhaltsverzeichnis

<b>1 Motivation</b>	<b>1</b>
<b>2 Beschleunigungssensoren</b>	<b>4</b>
2.1 Piezoresistive Sensoren . . . . .	6
2.2 Kapazitative Sensoren . . . . .	6
<b>3 Berechnung der Wegwerte aus den Beschleunigungsdaten</b>	<b>9</b>
3.1 Numerische Integration . . . . .	9
3.2 Problematik der Integration Noisebehafteter Daten . . . . .	9
<b>4 Entwicklung des Schwingungsmesssystems: Prototyp</b>	<b>11</b>
4.1 Bosch BMA180 . . . . .	11
4.2 Arduino . . . . .	11
4.3 Schnittstellen . . . . .	12
4.3.1 I <sup>2</sup> C . . . . .	12
4.3.2 Serieller Port . . . . .	13
4.4 Aufbau und Schaltung . . . . .	14
4.4.1 Testaufbau . . . . .	14
4.4.2 Fester Aufbau . . . . .	15
4.5 Software . . . . .	18
4.5.1 Arduino . . . . .	19
4.5.2 Restitution . . . . .	25
4.5.3 PC Logger . . . . .	25
<b>5 Experimente in Waakirchen, erster Einsatz des Prototyp</b>	<b>26</b>
5.1 Ergebnisse der Messungen in Waakirchen . . . . .	27
5.1.1 Messungen an der Radarantenne . . . . .	27
5.1.2 Messungen am Autoklavenaufbau . . . . .	29
<b>6 Entwicklung des Schwingungsmesssystems: ASC 5511LN-002</b>	<b>33</b>
6.1 ASC 5511LN-002 . . . . .	33
6.2 Diamond-MM-16-AT PC/104 Analog I/O Module . . . . .	35
6.2.1 Interrupts . . . . .	36
6.2.2 FIFO Speicher . . . . .	37
6.3 Software . . . . .	38
<b>7 Experimente am Geomatikum</b>	<b>47</b>
<b>8 Zusammenfassung und Ausblick</b>	<b>56</b>



## 1 Motivation

Mit einem Dopplerradar lassen sich vulkanische Eruptionen quantitativ erfassen und auswerten. In vielen Studien haben diese Messungen bereits zu neuen Erkenntnissen über die dahinter liegenden Prozesse geführt.

Ursprünglich in der Meteorologie zur Regenmessung eingesetzt, ermöglichen Dopplerradargeräte die Messung der Geschwindigkeiten von Partikeln in der Eruptions säule. Ein solches Radar der Firma Metek ist in Abbildung 1 zu sehen.

Über die Satellitenantenne wird ein Radarstrahl mit der Wellenlänge von 1.25cm gebündelt und auf die Eruptionssäule gerichtet. Aus den Eigenschaften des zurück gestreuten Signals ermittelt das Radar dann eine Geschwindigkeitsverteilung der Streukörper.

Ein Radar kann entweder klassisch mit gepulsten Signalen oder mit einer zeitlich konstanten Sendeleistung arbeiten. Im ersten Fall ergibt sich die Entfernung eines Objektes durch die Laufzeit eines abgestrahlten Pulses. Wir arbeiten jedoch mit einem konstanten Signal und bestimmen die Entfernung über eine Frequenzmodulation. Dieses Verfahren nennt man *Frequency Modulated Continuous Wave*; es hat den Vorteil, dass der Energieverbrauch deutlich geringer ist.

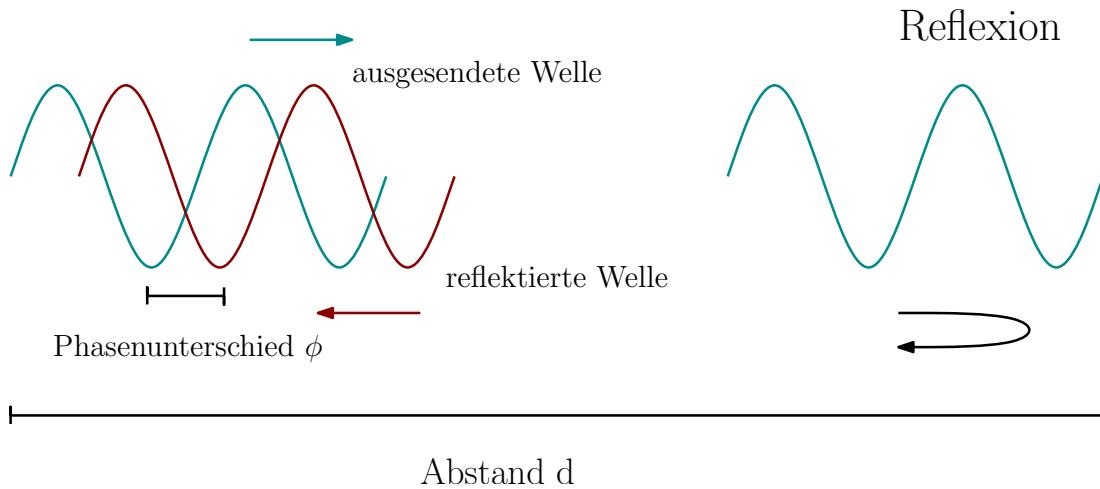


**Abb. 1:** Testaufbau des Metek Dopplerradars auf Stromboli mit Martin Vögele

Das Funktionsprinzip des Doppler Radars beruht auf dem Frequenzunterschied zwischen gesendetem und empfangenen Signal. Dieser Unterschied ist direkt pro

portional zur Geschwindigkeit eines Objekts, welches sich entlang des Radarstrahls bewegt. Eine genaue Beschreibung des Prinzips ist zum Beispiel in [Hort et al., 2003] zu finden.

Sehr langsame Bewegungen rufen einen sehr kleinen Frequenzunterschied hervor, der unter Umständen nicht mehr messbar ist. Betrachtet man allerdings die Phasenverschiebung des Signals, so lässt sich aus diesen Informationen auch die langsame Änderung der Entfernung eines Objekts sehr genau bestimmen. Mit dieser sogenannten *Phasenmessung* können zum Beispiel relativ langsam ablaufende Deformationsprozesse von Vulkanen im Millimeterbereich untersucht werden.



**Abb. 2:** Prinzip der Phasenmessung. Zum besseren Verständnis dargestellt für ein gepulstes Radar.

Das Funktionsprinzip ist in Abbildung 2 zu sehen. Aus Gründen der Darstellbarkeit und des besseren Verständnisses verwende ich zur Erklärung ein gepulstes Radar. Das empfangene Signal wird mit dem ausgehenden verglichen. Dabei tritt ein Phasenunterschied  $\phi$  auf, der vom Abstand  $d$  zwischen Radar und Reflektor abhängt. Jede Änderung im Phasenunterschied bedeutet eine Änderung des Abstandes. Zu beachten ist, dass dieses Verfahren keinen absoluten Abstand liefert, sondern immer nur eine relative Änderung zu einem gewählten Referenzpunkt.

Mit dieser Vorgehensweise konnten am Santiaguito Vulkan in Guatemala 2007 bereits Hebungen und Senkungen von mehreren Zentimetern Höhe wenige Sekunden vor und nach einer Eruption festgestellt werden [Scharff et al., 2007].

Bei der Interpretation dieser Deformationsdaten wird stets davon ausgegangen, dass das Radar fest auf einem Stativ fixiert ist und sich nicht bewegt. Sämtliche gemessene Bewegung wird als Bewegung der beobachteten Objekte gewertet. In der Praxis kann es allerdings vorkommen, dass sich das Radar aufgrund von Wind, Erschütterungen oder Deformationen des Untergrunds sehr wohl bewegt und in

Schwingung gerät. Mit einer alleinigen Abstandsmessung lassen sich diese Bewegungen des Messgerätes nicht von denen des Messobjektes unterscheiden.

Aus diesem Grund entwickle ich in dieser Arbeit ein Schwingungsmesssystem mit einem dreiachsigem Beschleunigungssensor und teste dieses anhand einiger Experimente.

Das Ziel ist es, zusätzlich zu den Radardaten Beschleunigungen aufzuzeichnen und aus diesen die Bewegung des Radarsystems zu berechnen. Damit ließen sich die Radardaten korrigieren und man könnte mit größerer Sicherheit feststellen, ob eine beobachtete Bewegung in den Radaraufzeichnungen auf eine Bewegung des Messobjektes oder auf eine Bewegung des Radars selber zurückzuführen ist.

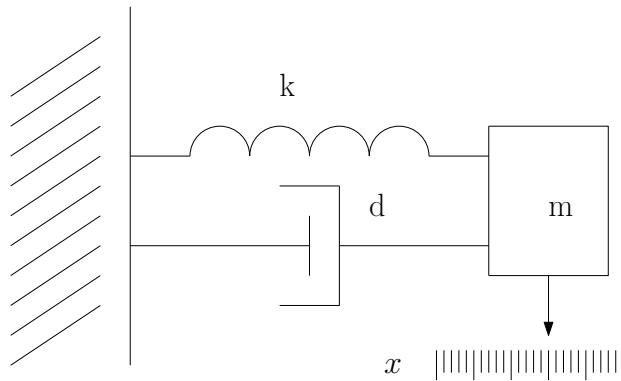
(Vorgehen und Aufbau der Arbeit beschreiben??)

## 2 Beschleunigungssensoren

Beschleunigungssensoren messen die Beschleunigung, die auf ein System wirkt, indem sie die auf eine Testmasse wirkenden Kräfte bestimmen. Die auf einen Körper wirkende Beschleunigung setzt sich nach Newton aus der Gravitation und Trägheitskräften zusammen. Isoliert man die Trägheitskräfte, so lässt sich daraus die Translationsbeschleunigung bestimmen. Mithilfe dieser und dem zweiten newtonischen Gesetz lässt sich die Positionsänderung des Systems berechnen:

$$F = m \cdot a \quad (1)$$

Es gibt verschiedene Prinzipien, mit denen die Beschleunigung gemessen werden kann. Am gebräuchlichsten ist das der Federwaage.



**Abb. 3:** Masse-Feder-System zur Beschleunigungsmessung.  $k$ ,  $d$ ,  $m$  und  $x$  bezeichnen Federkonstante, Dämpfung, Masse und Auslenkung (nach [Klingbeil, 2006])

Eine Masse  $m$  (auch seismische Masse genannt) ist über eine Feder der Federkonstanten  $k$  mit einem festen Bezugspunkt verbunden. Die Auslenkung  $x$  ist proportional zur auf die Masse wirkenden Beschleunigung  $a$ .

$$a = \frac{k}{m} \cdot x \quad (2)$$

Zu beachten ist die Ausrichtung des Sensors:  $a$  entspricht immer der Projektion der Beschleunigung auf die Auslenkungsrichtung von  $m$ .

Das System führt eine gedämpfte harmonische Schwingung aus. Einmal angeregt beginnt es mit konstanter Frequenz zu schwingen, die Amplitude nimmt kontinuierlich ab. Die Bewegungsgleichung lautet (vergleiche auch [Meschede, 2001]):

$$m\ddot{x} + d\dot{x} + kx = 0 \quad (3)$$

Wobei  $d$  die Dämpfung ist. Eine Lösung dieser Differentialgleichung lautet:

$$x(t) = x_0 e^{-\delta t} \sin(\omega_d t + \varphi_0) \quad \text{mit } \delta = \frac{d}{2m} \quad (4)$$

Die Lösung ist aus zwei elementaren Funktionen zusammengesetzt, wobei die eine den periodischen Anteil ausmacht und die andere den dämpfenden Anteil. Der periodische Anteil lässt sich auch als  $x_0 e^{i\omega_d t}$  schreiben, wobei

$$\omega_d = \sqrt{\omega_0^2 - \delta^2} \quad (5)$$

die gedämpfte Eigenfrequenz genannt wird und einen entscheidenden Einfluss auf das Schwingverhalten hat.

Bei  $\delta < \omega_0$  ist die Dämpfung gering und es gilt

$$\omega_d \approx \omega_0 \quad (6)$$

Dies nennt man den *Schwingungsfall*.

Wird  $\delta$  sehr groß gegenüber  $\omega$ , so wird der Term unter der Wurzel negativ und  $\omega_d$  damit imaginär. Aus

$$e^{i\omega_d t} \quad (7)$$

wird

$$e^{-kt} \quad (8)$$

und bewirkt eine zusätzliche Dämpfung. Eine Schwingung existiert nicht mehr und das System erreicht seine Ruhelage nach sehr langer Zeit. Dieses als *Kriechfall* bezeichnete Verhalten ist in den meisten Fällen unerwünscht.

Wenn die Dämpfung jedoch genau so groß wird, dass

$$\delta = \omega_0 \quad (9)$$

gilt, wird  $\omega_d = 0$  und das System kommt in kürzestmöglicher Zeit zur Ruhe. Man spricht vom *aperiodischen Grenzfall*. Dieses Verhalten ist bei Beschleunigungsauf-

nehmern erwünscht, weil man so die höchste Wiederholfrequenz von Messungen erreicht, ohne dass sich die Anregungen gegenseitig überlagern.

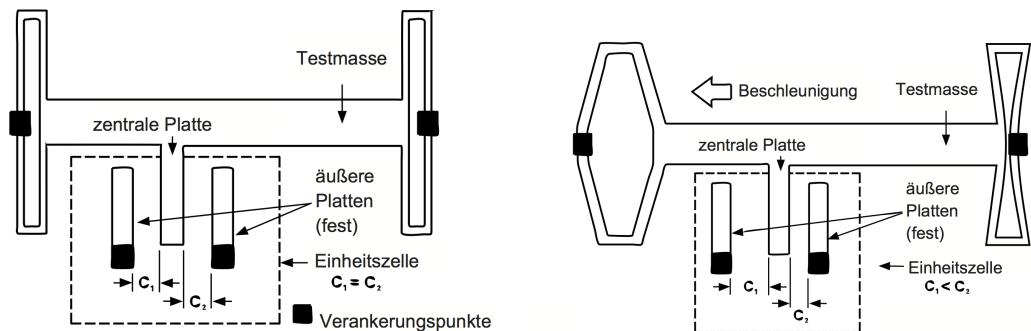
Mittlerweile sind Beschleunigungssensoren in den meisten Fällen als MEMS realisiert (**Micro Electro Mechanical Systems**). Sehr kleine mechanische Elemente (1-100 Mikrometer) werden zusammen mit elektronischen Schaltungen auf einen Siliziumwafer aufgebracht. Dabei werden Techniken aus der Fabrikation von integrierten Schaltkreisen (ICs) verwendet. So können komplizierte elektromechanische Systeme in winziger Größe und hoher Stückzahl hergestellt werden. Der geringe Preis ist maßgeblich dafür verantwortlich, dass die Sensoren in immer mehr Anwendungen integriert werden (Autos, Smartphones, Quadroopter...).

Die Beschleunigungsmessung erfolgt wie oben beschrieben über die Messung der Auslenkung einer Testmasse. Dazu haben sich zwei Verfahren durchgesetzt, die ich im Folgenden kurz erläutern möchte.

## 2.1 Piezoresistive Sensoren

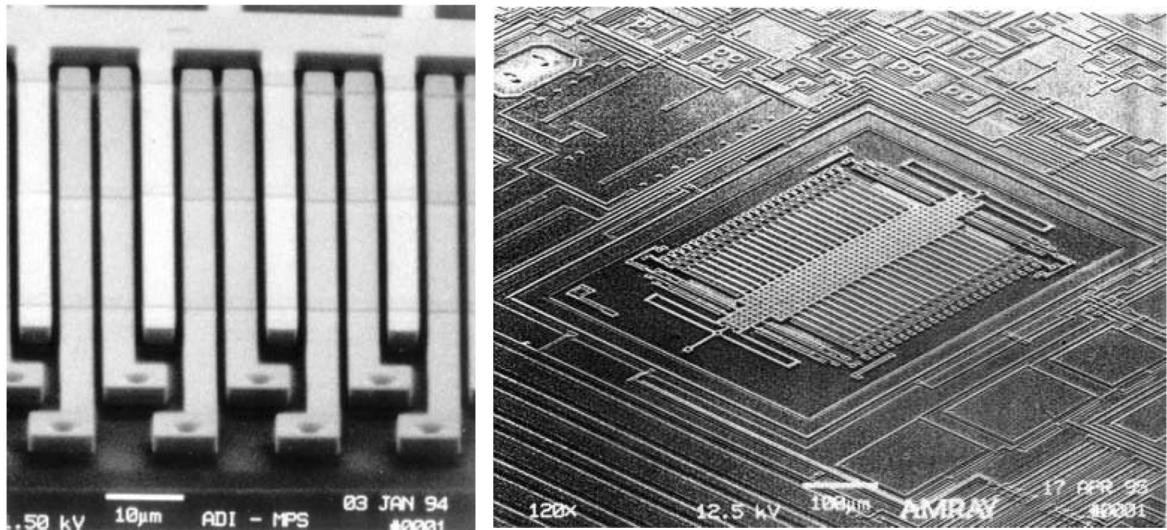
Piezoresistive Sensoren machen sich den piezoelektrischen Effekt zunutze. In der Feder der Testmasse befinden sich Piezoelemente, welche sich bei Auslenkung verformen und damit ihren Widerstand ändern. Silizium ist ein geeignetes Material, da es sehr empfindlich ist, linear reagiert und gleichzeitig gut mit der MEMS Technik kombinierbar ist [Kanda, 1991].

## 2.2 Kapazitative Sensoren



(a) Sensor in Ruhe, Abstand der Kondensatorplatten ist gleich,  $C_1 = C_2$       (b) Sensor während einer externen Beschleunigung,  $C_1 < C_2$

**Abb. 4:** Vereinfachtes Diagramm des ADXL05 [ADXL05, 1996]



**Abb. 5:** ADXL05 unter dem Elektronenmikroskop, Beispiel für einen kapazitiven MEMS Beschleunigungssensor mit 46 Einzelzellen [Klingbeil, 2006].

Die Auslenkung lässt sich auch über eine Kapazitätsmessung bestimmen, wenn man je eine Elektrode an der Testmasse und am fixen Referenzpunkt anbringt [Sherman et al., 1992]. Eine mögliche Realisierung ist in Abbildung 4 a,b zu sehen: Es handelt sich um einen sogenannten Differentialkondensatorsensor [Schmidt, 2007].

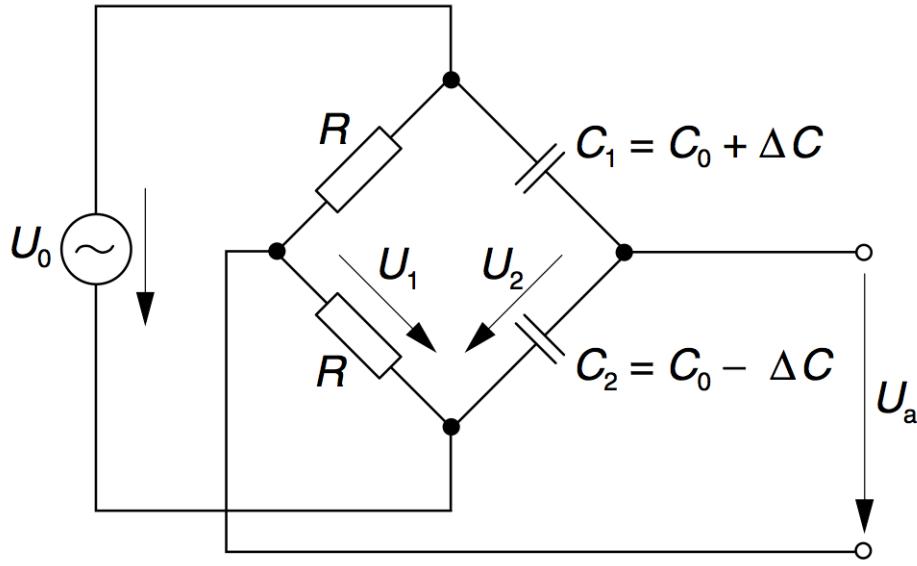
Die seismische Masse befindet sich zwischen zwei fest verankerten Platten, die zusammen zwei Kondensatoren  $C_1$  und  $C_2$  bilden (Abb. 4(a)). Eine auftretende Beschleunigung führt also zu einer Auslenkung der Mittelelektrode des Kondensatorpaars (seismische Masse) um die Länge  $\pm x$  und damit zu einer symmetrischen Kapazitätsänderung um  $\pm\Delta C$  (Abb. 4(b)). Dieser Aufbau ist eine sogenannte Einzelzelle. In einem Sensor befinden sich viele Einzelzellen, um die Kapazitätsvariation und damit die Sensibilität zu erhöhen (Abb. 5).

Um diese Variation in ein elektrisches Ausgangssignal umzusetzen, wird eine sogenannte Brückenschaltung verwendet (Abb. 6). Die folgende Herleitung ist aus [Schmidt, 2007] entnommen:

Wird eine Wechselspannung an die Brücke angelegt, so ergibt sich nach der Spannungsteilerregel für den Ausgang  $U_a$ :

$$U_a = U_1 - U_2 = U_0 \frac{R}{2R} - U_0 \frac{\frac{1}{j\omega C_2}}{\frac{1}{j\omega C_2} + \frac{1}{j\omega C_1}} \quad (10)$$

Durch Kürzen ergibt sich:



**Abb. 6:** Brückenschaltung mit Differentialkondensator (Die mit  $U_a$  verbundenen Elektroden von  $C_1$  und  $C_2$  bilden eine gemeinsame Platte) [Schmidt, 2007]

$$U_a = \frac{U_0}{2} - U_0 \frac{\frac{1}{C_2}}{\frac{1}{C_2} + \frac{1}{C_1}} = U_0 \left( \frac{1}{2} - \frac{C_1}{C_2 + C_1} \right) = \frac{U_0}{2} \left( \frac{C_2 - C_1}{C_2 + C_1} \right) \quad (11)$$

Mit  $C_1 = C_0 + \Delta C = \varepsilon_0 \cdot \varepsilon_r \cdot A / (d_0 - x)$  und  $C_2 = C_0 - \Delta C = \varepsilon_0 \cdot \varepsilon_r \cdot A / (d_0 + x)$  erhält man eine lineare Abhängigkeit der Ausgangsspannung  $U_a$  von der Auslenkung  $x$ :

$$U_a = -U_0 \frac{x}{2d_0} \quad (12)$$

Diese Spannung lässt sich nun digitalisieren und auslesen. In vielen MEMS Bausteinen ist bereits ein integrierter Analog Digital Wandler eingebaut, sodass die Messwerte direkt digital abrufbar sind.

### 3 Berechnung der Wegwerte aus den Beschleunigungsdaten

#### 3.1 Numerische Integration

Um aus den gemessenen Beschleunigungen die gesuchten Wegwerte (Ausschläge) zu ermitteln, müssen die Messwerte zweifach über die Zeit integriert werden. Da die zu integrierenden Funktionen als begrenzte Anzahl von Abtastpunkten in diskreten Zeitabständen vorliegen, bietet es sich an, numerische Integrationsverfahren anzuwenden. Die dazu möglichen Verfahren, wie z.B. Rechteckverfahren, Trapezformel, Rombergverfahren, unterscheiden sich durch die Art der Interpolation zwischen den Abtastpunkten [Bronstein et al., 1995, S. 760ff]. Die Wahl des Integrationsverfahrens beeinflusst dabei die Genauigkeit des Ergebnisses.

In dieser Arbeit verwende ich die Trapezformel und erhalte damit für den Weg folgende Rekursionsformel:

$$y_{i+1} = \Delta t^2 \cdot a_i - y_{i-1} + 2 \cdot y_i \quad (13)$$

Mit dieser Formel kann aus den vorherigen Wegwerten  $y_{i-1}$  und  $y_i$  sowie den gemessenen Beschleunigungen  $a_i$  und dem Zeitintervall  $\Delta t$  der nächste Wegwert bestimmt werden.

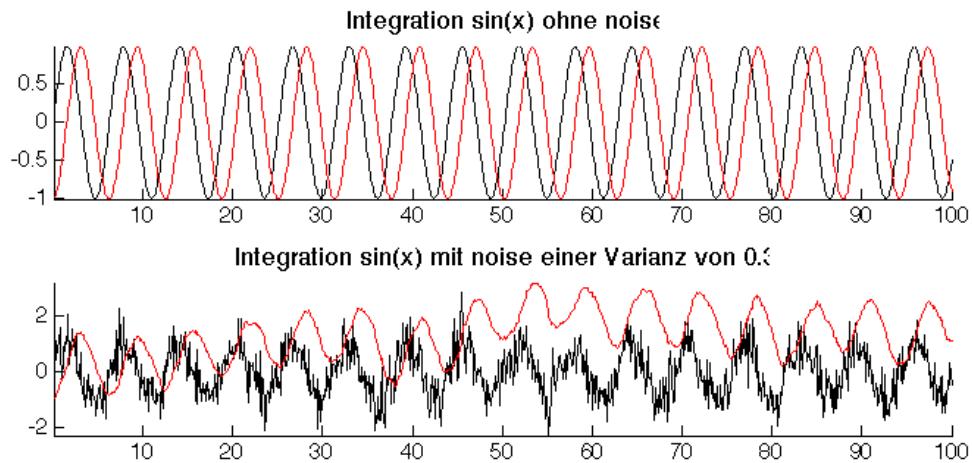
Die Startwerte für  $i = 1$  sind vorerst nicht bekannt und können z.B.  $y_0 = y_1 = 0$  gewählt werden. Dies führt allerdings zu systematischen Fehlern, die anschließend behoben werden müssen.

#### 3.2 Problematik der Integration Noisebehafteter Daten

Wie alle nicht synthetischen Daten enthalten die aufgenommen Beschleunigungsmessungen ein gewisses Rauschen. Schwankungen in der Spannungsversorgung, elektromagnetische Einstreuungen oder thermisches Rauschen sind einige Beispiele für mögliche Ursachen. Solange das Signal deutlich größer ist als das Rauschen, ist das kein Problem. Man spricht von einem guten Signal- zu Rauschverhältnis.

Bei der Integration der Beschleunigungswerte zu Wegwerten handelt es sich jedoch gewissermaßen um eine Addition aller Messwerte und damit einer Addition allen Rauschens. Je länger die Zeit ist, über die integriert wird, desto größer wird der Fehler in den Wegdaten.

Das Resultat ist in Abbildung 7 zu erkennen. Ich habe in Matlab eine Sinusfunktion mit 1000 Stützstellen numerisch integriert. Dabei habe ich zunächst keinen Noise hinzugefügt. Bei der unteren Abbildung jedoch liegt auf dem Sinus ein Rauschen



**Abb. 7:** Numerische Integration von  $\sin(x)$ . Oben: ohne Noise Unten: Mit Noise einer Varianz von 0.3

mit einer Varianz von 0.3. Es ist eine deutliche Drift und eine langwellige Störung der resultierenden Daten zu sehen. Bei der zweifachen Integration, die man benötigt um aus den Beschleunigungsdaten Wegdaten zu berechnen, tritt dieser Effekt sogar noch verstärkt auf. Die Fehler der Messungen überlagern die eigentlichen Messergebnisse und führen damit zu völlig falschen Wegstrecken.

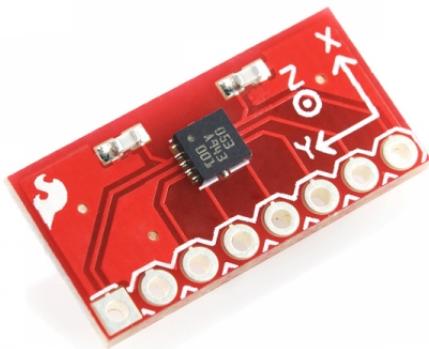
Um diesen Effekt zu verhindern, müssen die Daten gefiltert werden. Die sich dafür bietenden Möglichkeiten möchte ich in der jeweiligen Auswertung besprechen.

An dieser Stelle sei auch auf das Nyquist Theorem verwiesen. Es besagt, dass ein Signal immer mindestens mit der doppelten Frequenz des hochfrequentesten Anteils im Signal abgetastet werden muss, damit es komplett rekonstruiert werden kann. [vgl. Shannon, 1949]

## 4 Entwicklung des Schwingungsmesssystems: Prototyp

Um abschätzen zu können wie groß die auftretenden Beschleunigungen an der Radarantenne sind, habe ich damit begonnen, einen günstigen und einfach zu verwendenden Prototypen zu entwickeln. Letzteres ist vor Allem wichtig, um entscheiden zu können, für welchen Messbereich und welche Frequenzen der eigentliche Sensor ausgelegt sein muss.

### 4.1 Bosch BMA180



**Abb. 8:** BMA180 Breakoutboard von Sparkfun. Der eigentliche Sensor ist in dem kleinen schwarzen Kasten auf dem Board verborgen.

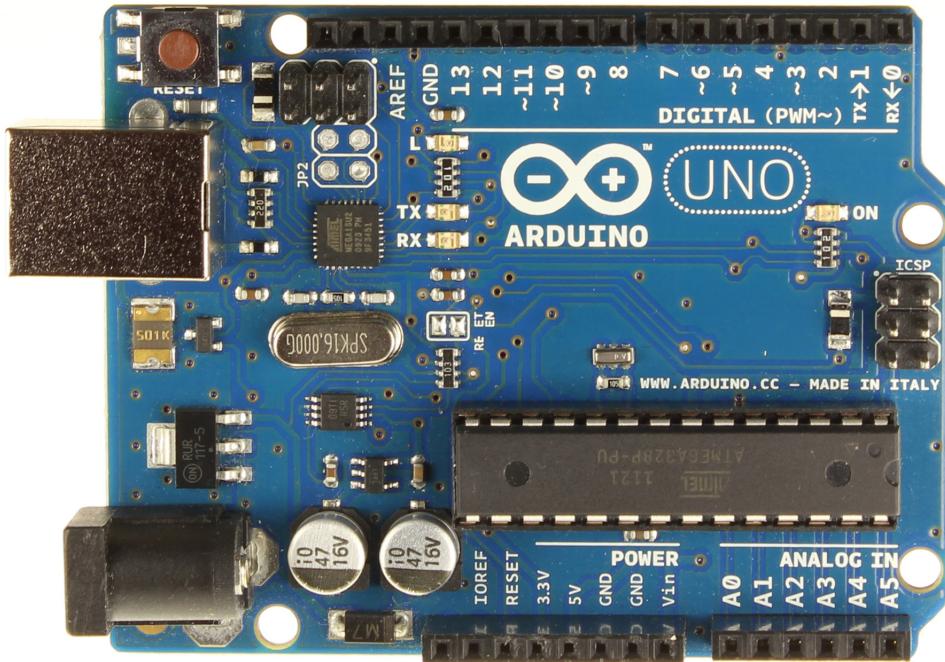
Bild: <https://www.sparkfun.com/products/9723>

Für den Prototyp fiel meine Wahl auf den digitalen, dreiachsigen MEMS Beschleunigungssensor *Bosch BMA180*. Er verfügt über einen eingebauten 14-bit Analog-Digital Wandler und sieben per Software verstellbare Messbereiche von  $\pm 1$  bis  $\pm 16g$ . Die Kommunikation kann über SPI (Serial Peripheral Interface) oder I<sup>2</sup>C (Inter-Integrated Circuit) erfolgen.

Ich nutze ein Breakoutboard von Sparkfun (Abb. 8), welches die IC Pins (SMD Technik) mit praktischen Lötösen verbindet. Außerdem sind bereits zwei spannungsstabilisierende Kondensatoren mit auf der Platine verbaut.

### 4.2 Arduino

Um den Sensor zu steuern und die Daten zur weiteren Bearbeitung an einen Rechner zu senden, benutze ich einen Arduino, eine auf ATmega Mikroprozessoren basierende Open-Source Entwicklungsplattform zur Verarbeitung von analogen und



**Abb. 9:** Arduino UNO

<http://arduino.cc/en/Main/ArduinoBoardUno>

digitalen Signalen. Die Programmierung kann über eine eigene Entwicklungsumgebung in einer an *Processing*<sup>1</sup> angelehnten Sprache erfolgen, die im Prinzip ein vereinfachtes C/C++ darstellt.

Die Plattform ist auf Prototyping und Experimente ausgelegt. Es ist bereits ein Bootloader vorinstalliert, so kann die Programmierung direkt über die serielle Schnittstelle erfolgen. Die Boards machen die meisten Pins des ATmegas für eigene Schaltungen verfügbar, in den gängigen Boards sind das 14 Pins, die frei als Ein- oder Ausgänge genutzt werden können. Die Stromversorgung kann über USB oder eine externe 5V Quelle erfolgen. Als Kommunikationsinterfaces werden SPI, ICSP (In-Circuit Serial Programming) und I<sup>2</sup>C angeboten.

Der ATmega arbeitet mit 16 MHz und hat einen geringen Energieverbrauch.

### 4.3 Schnittstellen

#### 4.3.1 I<sup>2</sup>C

Zur Kommunikation zwischen Beschleunigungssensor und Arduino habe ich den I<sup>2</sup>C Bus [NXP, 2012] gewählt. Dabei handelt es sich um einen von Philips entwickelten seriellen Datenbus, der ursprünglich gebaut wurde, um Chips in Fernsehgeräten

---

<sup>1</sup>[processing.org](http://processing.org)

steuern zu können. Inzwischen ist das Patent ausgelaufen und er wird in vielen Hardwareprojekten verwendet, da er sehr einfach zu verstehen und zu verwenden ist.

Ich benutze die Arduino Wire Library [Arduino, 2012], welche die gesamte Kommunikation über I<sup>2</sup>C steuert und einfache Funktionsaufrufe, wie zum Beispiel Senden und Empfangen, zur Verfügung stellt.

#### **4.3.2 Serieller Port**

Zum einfachen Anschluss des Arduinos an einen PC oder Datenlogger wird eine serielle Schnittstelle nach RS232 verwendet. Auf dem Arduino befindet sich ein ATmega16U2, welcher die seriellen Signale in USB umwandelt und dafür sorgt, dass der Arduino am PC als virtueller COM-Port erscheint.

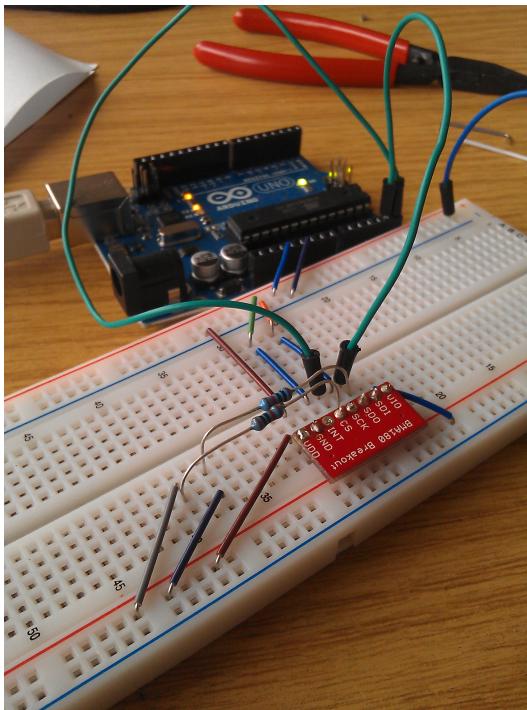
## 4.4 Aufbau und Schaltung

### 4.4.1 Testaufbau

Um die korrekte Verschaltung zu überprüfen und die Software für das Auslesen der Daten zu entwickeln, habe ich zunächst auf dem Breadboard gearbeitet (Abb. 10(a)). Der dazu verwendete Schaltplan ist in Abbildung 11 zu sehen.

Da der Sensor sowohl im I<sup>2</sup>C- sowie im SPI-Modus betrieben werden kann, ist es notwendig, den Modus bereits bei der Verschaltung einzustellen. Über den CS Pin (Pin 4) ist dies möglich: Schaltet man ihn auf High (3.3V), so benutzt der Chip I<sup>2</sup>C, verbindet man ihn mit GND, so wird ISP verwendet. Je nach Modus haben die Pins unterschiedliche Funktionen. In Tabelle 1 sind die verschiedenen Konfigurationen aufgeführt.

Pin 6 legt im von mir benutzten I<sup>2</sup>C Modus die Adresse des Chips fest. Ist dieser auf Low (GND), so ist die Adresse 0x40.



(a) Testaufbau auf dem Breadboard

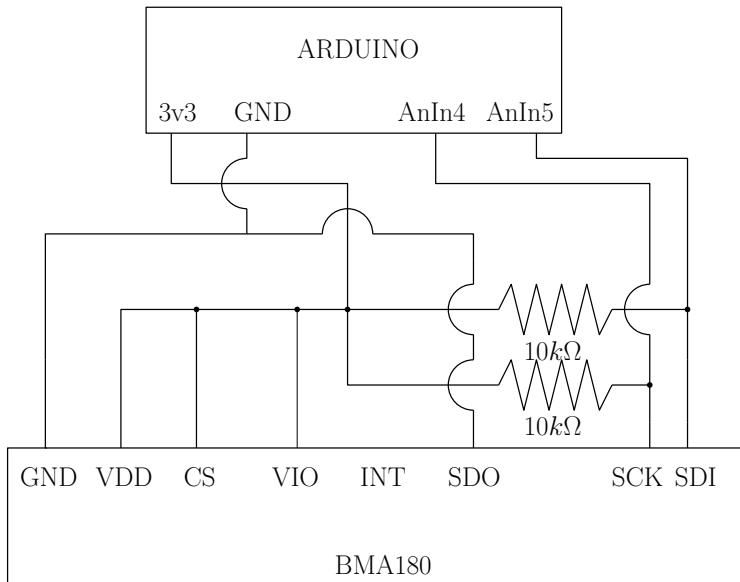


(b) Testplatine am Radar

**Abb. 10:** Testaufbau des Sensors

Pin	SPI mode	I2C mode
7	SDI input	SDA birectional (!)
6	SDO output	ADDR adress bit, input
5	SCLK input	SCL input
4	CSB chip select, input	I2C mode select, input

**Tab. 1:** BMA180 Pinbelegung für SPI und I<sup>2</sup>C Modes [BMA180, 2009]



**Abb. 11:** Schaltplan Testaufbau. Der Interrupt ist nicht verbunden, VDD und CS sind auf 3.3V geschaltet, GND und SDO auf die Masse des Arduinos gezogen, die I<sup>2</sup>C Datenleitungen SCK und SDI sind mit den Arduinopins A4 und A5 verbunden, wobei zusätzlich 10kΩ Pull-Up-Widerstände eingebaut sind.

#### 4.4.2 Fester Aufbau

Um das System praktisch nutzen zu können, muss es natürlich fest aufgebaut werden und mit einem Gehäuse versehen werden, welches es erlaubt, ihn fest an einem Testobjekt anzubringen und ihn gleichzeitig vor Schäden durch mechanische oder witterungsbedingte Einflüsse schützt.

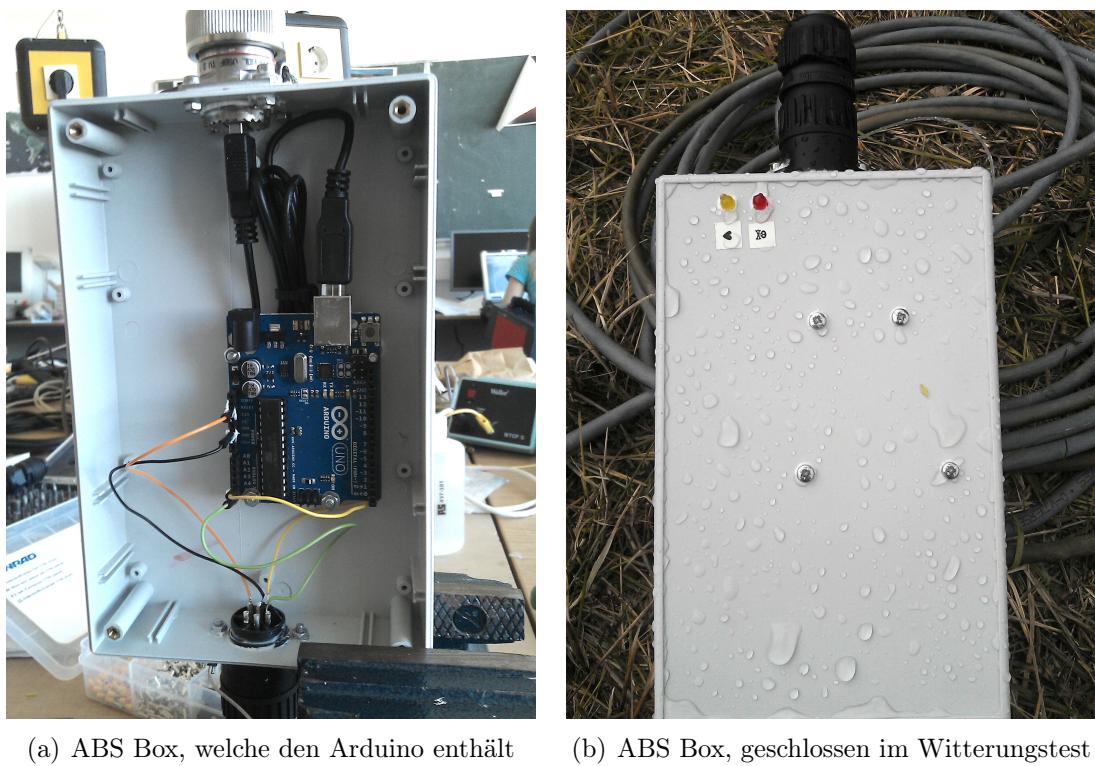
Um die eigentliche Sensoreinheit möglichst kompakt zu halten, habe ich mich entschieden, den Beschleunigungssensor vom Arduino zu trennen und auf eine kleine Lochrasterplatine zu löten. Diese wird in einen festen Block aus Polyurethanharz<sup>2</sup> eingegossen, womit sie gleichzeitig gut geschützt und leicht anzubringen ist (Abb. 12).

<sup>2</sup>OPTICALLY CLEAR POLYURETHANE, RS COMPONENTS, RS 195-984A

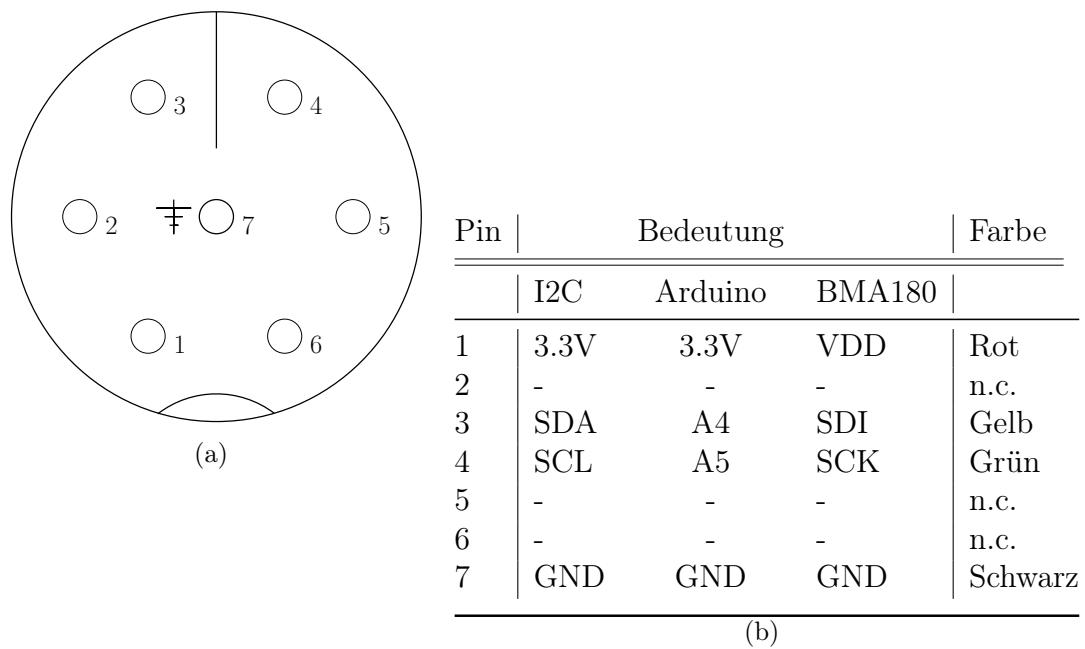


**Abb. 12:** BMA180 in Polyurethanharz eingegossen, die Kunststoffform wurde von den feinmechanischen Werkstätten des Fachbereichs Geowissenschaften gefertigt.

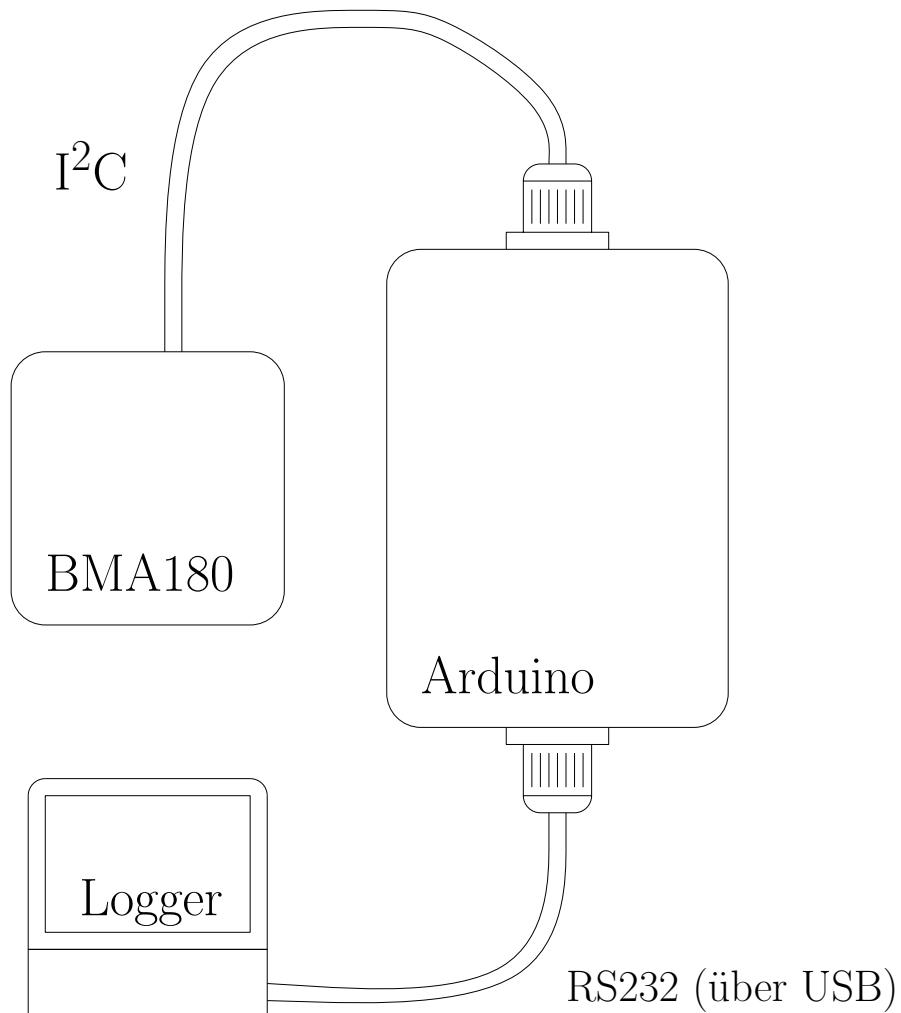
Der Arduino ist in ein ABS Gehäuse (zum Beispiel von BOSS Enclosures) eingebaut (Abb. 13(a) und 13(b)). Die I<sup>2</sup>C Verbindung mit dem Sensor erfolgt über einen Hirschmannstecker<sup>TM</sup> mit 7 Polen, von denen 4 beschaltet sind (siehe Abb. 15). Mit dem Logger ist der Arduino über ein USB-Kabel verbunden, über das die Kommunikation per RS232 geführt wird.



**Abb. 13:** ABS Box mit Arduino



**Abb. 15:** I<sup>2</sup>C Beschaltung des Hirschmannsteckers<sup>TM</sup> (n.c. = not connected)



**Abb. 14:** Verbindungen zwischen BMA180, Arduino und Logger per I<sup>2</sup>C bzw. RS232

#### 4.5 Software

Die Software besteht aus zwei Teilen: Der Code auf dem Arduino fragt die Daten vom Sensor ab und stellt sie per RS232 zur Verfügung. Der zweite Teil auf dem Logger sammelt die Messungen über den Seriellen Port und schreibt diese gemeinsam mit der jeweils aktuellen Uhrzeit in eine Datei.

Bei der Erläuterung des Codes möchte ich mich auf die wesentlichen Punkte beschränken. Die vollständigen Quelltexte befinden sich auf der beiliegenden CD-ROM.

#### 4.5.1 Arduino

Der Arduino ist für die Kommunikation mit dem BMA180 über I<sup>2</sup>C zuständig. Für diese Aufgabe nutze ich die Arduino Bibliothek bma180<sup>3</sup>. Sie enthält eine Sammlung von Lese- und Schreibroutinen, die ich für meine Zwecke angepasst habe. Hier möchte ich die einzelnen Schritte beschreiben, die notwendig sind, um eine Messung zu erhalten und diese an den Logger weiterzuleiten.

Die Steuerung des BMA180 erfolgt über direkte Zugriffe auf die Register. Entweder schreibt man einen neuen Wert in ein Register, oder man liest ein Register aus. Es gibt fünf Arten von Registern: **test**, **control**, **image**, **status** und **data**. Davon interessieren uns nur die **control** und die **data** Register. Weitere Informationen über die zusätzlichen Register lassen sich im Datenblatt [BMA180, 2009] nachschlagen. Die **control** Register dienen dazu, den Beschleunigungssensor zu steuern, vornehmlich also den Messbereich festzulegen und eventuell einen zusätzlichen digitalen Bandfilter einzustellen. Dazu müssen Werte in Register hineingeschrieben werden. Dazu geht man folgendermaßen vor:

#### In Register schreiben

1. Übertragung initiieren (mit der Adresse des Sensors)
2. Adresse des Registers senden, in welches wir schreiben möchten
3. Daten senden, die wir in das Register schreiben möchten
4. Übertragung beenden

Da ein Register immer ein Byte (8 Bit) groß ist, finden sich oftmals verschiedene Einstellungen in einem Register um Platz zu sparen. Einen korrekten Schreibvorgang auszuführen wird dadurch komplexer. Daher möchte ich den Prozess am Beispiel der Messbereichseinstellung demonstrieren.

---

<sup>3</sup>Selbst nach ausführlicher Recherche ist es mir nicht gelungen, den Originalautor der Bibliothek zu ermitteln. Ich habe sie von John Mc Combs, welcher allerdings nicht der Ursprungsautor ist.

<https://bitbucket.org/johnmccombs/arduino-libraries/src/058c7101c8da/bma180> (August 2012)

offset_y	30h	offset_x<11:0> (msb)	80h
offset_x	38h	offset_x<11:4> (msb)	80h
offset_t	37h	offset_x<4:0> (msb)	80h
offset_lsb2	36h	offset_z<3:0> (lsb)	00h
offset_lsb1	35h	offset_x<3:0> (lsb)	00h
gain_z	34h	gain_z<6:0>	smp_skip
gain_y	33h	gain_y<6:0>	wake-up
gain_x	32h	gain_x<6:0>	shadow_dis
gain_t	31h	gain_t<4:0>	dis_reg
			80h

Abb. 16: Memory Map BMA180, Ausschnitt [BMA180, 2009]

Die relevanten Bits befinden sich im Register 0x35<sup>4</sup>, Bit eins, zwei und drei<sup>5</sup> (siehe Abb. 16). Hier die entsprechenden Zeilen aus dem Quellcode:

```

1 void BMA180::setGSensitivity(GSENSITIVITY maxg) //1, 1.5 2 3 4 8 16
2 {
3     setRegValue(0x35, maxg << 1, 0xF1);
4     gSense = maxg;
5 }
```

Listing 1: Funktion zum Einstellen des Messbereichs aus der bma180 Bibliothek

maxg ist eine dem Messbereich zugeordnete Binärzahl zwischen 000 und 110

Die aufgerufene Funktion *setRegValue* (Zeile 3) bekommt als Argumente das zu schreibende Register (0x35), den zu schreibenden Wert (*maxg* << 1) und zusätzlich eine Maske, die angibt, welche Bits des Registers nicht(!) verändert werden sollen.

Da das letzte Bit (Bit Null) eine andere Funktion hat, 'schieben' wir *maxg* mit Hilfe der *bitshift Operation* << um ein Bit nach links.

Bei *gSense* (Zeile 4) handelt es sich um eine Statusvariable (Membervariable der Klasse bma180), die später benutzt wird, um den genutzten Wertebereich mit zu dokumentieren.

```

1 void BMA180::setRegValue(int regAdr, int val, int maskPreserve)
2 {
3     int preserve=getRegValue(regAdr);
4     int orgval=preserve & maskPreserve;
5     Wire.beginTransmission(address);
6     Wire.write(regAdr);
7     Wire.write(orgval | val);
8     int result = Wire.endTransmission();
9     checkResult(result);
10 }
```

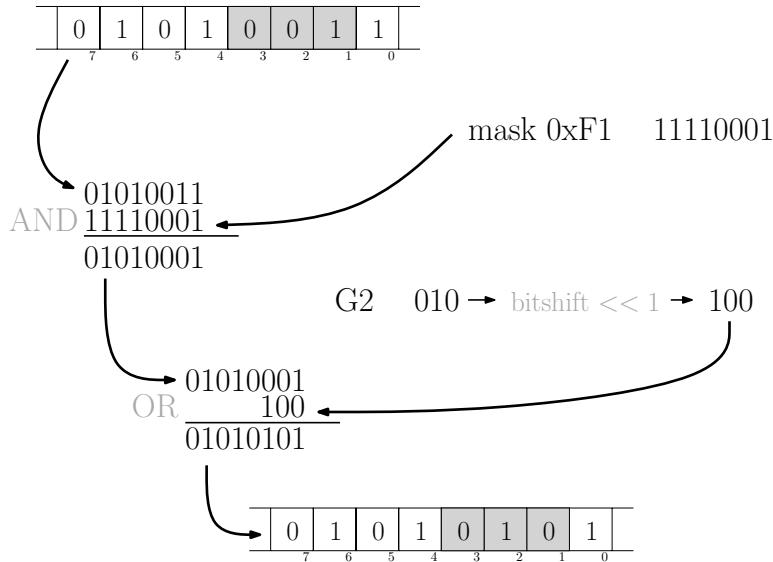
Listing 2: Funktion zum Schreiben eines Registers aus der BMA180 Bibliothek

<sup>4</sup>Ein vorangestelltes 0x bedeutet, dass es sich um eine Zahl im Hexadezimalsystem handelt

<sup>5</sup>Es wird von rechts nach links gezählt und mit 0 begonnen

Nun lesen wir zunächst den bisherigen Wert des Registers ein (Zeile 3) und maskieren ihn mit der übergebenen Maske, so dass wir alle Bits übernehmen, außer denen, welche wir verändern wollen (Zeile 4). Dann schreiben wir in das Register, wobei wir nach dem oben beschriebenen Schema vorgehen. Beim Senden der Daten (Zeile 7) verknüpfen wir jedoch *orgval* mittels eines bitweisen *OR* mit dem zu schreibenden Wert. So haben wir sichergestellt, dass wir nur die relevanten Bits verändern.

In Abbildung 17 ist das Vorgehen noch einmal dargestellt.



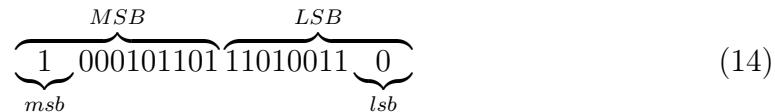
**Abb. 17:** Beispiel eines Schreibvorgangs in das Register 0x35. Nur die grau markierten Bits werden verändert.

### Daten aus Registern lesen

In die data Register werden die gemessenen Beschleunigungsdaten geschrieben. Da der A/D-Wandler des BMA180 eine Auflösung von 14 Bit hat, existieren pro Achse 2 Register mit jeweils 8 Bit. Hier benötigen wir das Konzept der **Bitwertigkeit**:

In einer Zahl haben die Stellen unterschiedlichen Wert. Verändert man zum Beispiel bei der Zahl 4650 die letzte Stelle, so ist der Unterschied maximal 9 (4650 verglichen zu 4659). Wird hingegen die dritte Stelle (von hinten) verändert, so kann der Unterschied bis zu 900 betragen. Analog verhält es sich mit den binären Zahlen. Dabei bezeichnen wir das höchstwertige Bit als **most-significant-bit (msb)** und das geringwertige Bit als **least-significant-bit (lsb)**.

Analog dazu spricht man vom höchswertigen Byte **most-significant-byte (MSB)** und geringwertigen Byte **least-significant-byte (LSB)**.



Die gemessene Beschleunigung ist also aufgeteilt und muss im Programm wieder zusammengesetzt werden.

Um die Beschleunigungsdaten vom Sensor zu laden, ist so vorzugehen:

1. Übertragung initiieren (mit der Adresse des Sensors)
2. Adresse des Registers senden, bei dem wir **anfangen** wollen zu lesen
3. Übertragung beenden
4. So viele Bytes abrufen, wie wir haben möchten
5. Übertragung beenden

```

1 void BMA180::readAccel()
2 {
3     unsigned int result;
4
5     Wire.beginTransmission(address);
6     Wire.write(0x02);
7     Wire.endTransmission();
8     Wire.requestFrom((int)address, 7);
9     if(Wire.available() == 7)
10    {
11         int lsb = Wire.read() >> 2;
12         int msb = Wire.read();
13         x = (msb << 6) + lsb;
14         if (x & 0x2000) x |= 0xc000; // set full 2 complement for neg values
15         lsb = Wire.read() >> 2;
16         msb = Wire.read();
17         y = (msb << 6) + lsb;
18         if (y & 0x2000) y |= 0xc000;
19         lsb = Wire.read() >> 2;
20         msb = Wire.read();
21         z = (msb << 6) + lsb;
22         if (z & 0x2000) z |= 0xc000;
23         temp = Wire.read();
24         if (temp & 0x80) temp |= 0xff00;

```

```

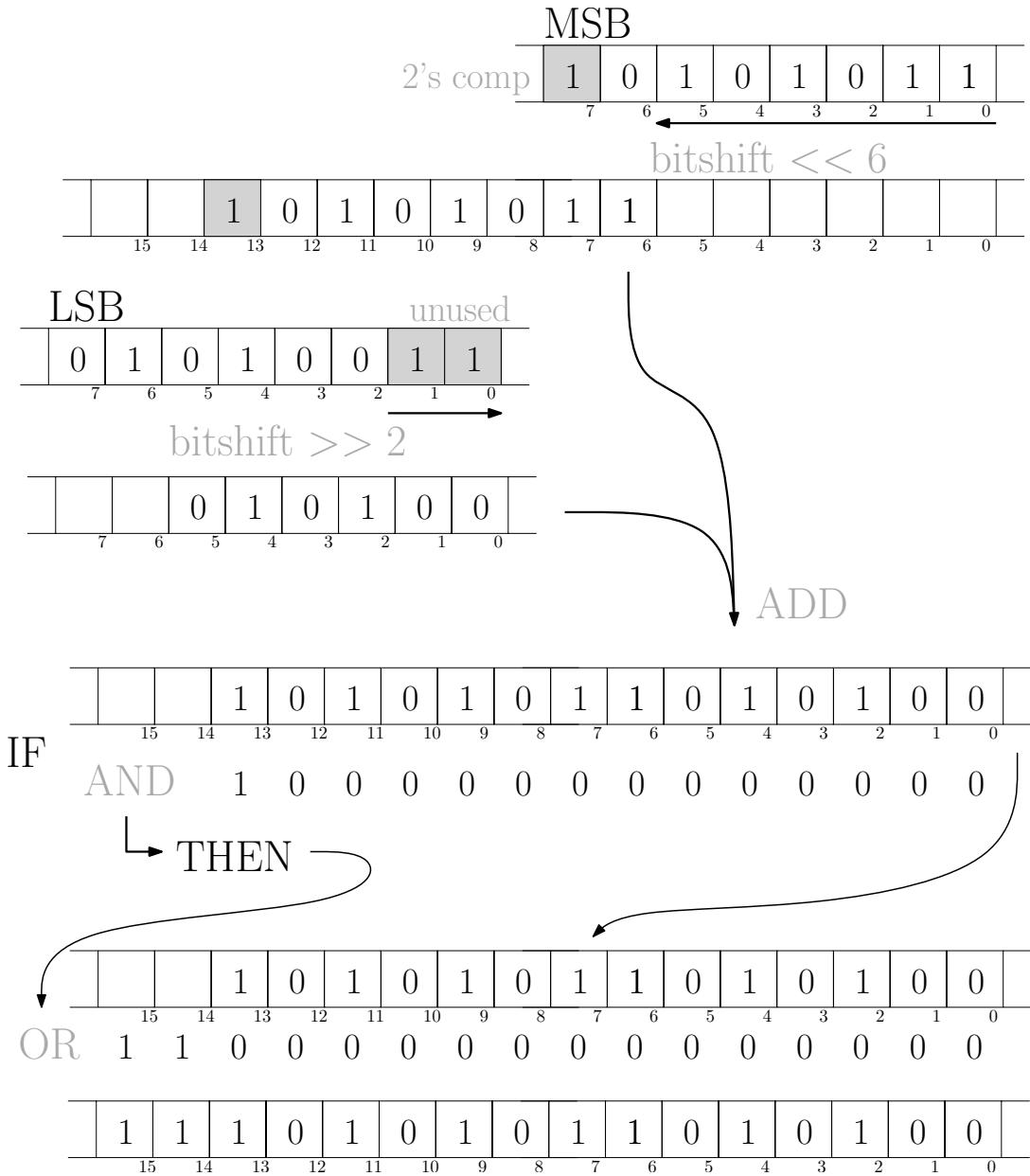
25 }
26   result = Wire.endTransmission();
27 }
```

**Listing 3:** Funktion zum Auslesen der Beschleunigungsdaten, x,y,z sind 2 Byte Integerzahlen

Das Register, bei dem wir beginnen wollen zu lesen, ist 0x02, (Zeile 6) und wir benötigen die nächsten 7 Bytes (Zeile 8). Jeweils zwei Bytes MSB und LSB für die drei Achsen und als letztes Byte lesen wir zusätzlich die aktuelle Temperatur aus. Diese wird jedoch im Weiteren nicht verwendet. Das Vorgehen ist noch einmal in Abbildung 18 gezeigt.

Mit *Wire.requestFrom(addr, n)* fordern wir einen Stapel von *n* Bytes vom Sensor (Zeile 8). *Wire.read()* entfernt immer das oberste Byte von diesem Stapel und gibt seinen Wert aus. Da das LSB an Stelle null und eins für den Messwert nicht relevante Bits enthält, entfernen wir diese per Bitshift (Zeile 11). Dann shiften wir das MSB im ganzen um sechs Stellen nach links, um es dann mit dem LSB zu addieren und damit den kompletten Messwert zu erhalten (Zeile 13).

Da die Daten im sogenannten Zweierkomplement (näheres in [Tietze and Schenk, 2002]) ausgegeben werden und es sich bei x (sowie bei y und z) um Integer Variablen handelt, müssen wir das Vorzeichenbit an vorderster Stelle setzen. Dies geschieht in Zeile 14 (bzw. Zeile 18 für y und Zeile 24 für z) mittels masking und OR Operator.



**Abb. 18:** Auslesen und zusammenfügen der Daten. MSB und LSB werden addiert. Mittels IF Abfrage und Masking wird überprüft, ob es sich um eine negative Zahl handelt (hochwertigstes bit = 1); Falls ja, so wird das korrekte Zweierkomplement gebildet und als Integer gespeichert.

### 4.5.2 Restitution

Jetzt haben wir eine Beschleunigungsmessung, jedoch ist diese noch in gerätespezifischen 'count' Einheiten. Diese müssen nun in die physikalisch bestimmte Einheit der Beschleunigung umgerechnet werden. Der Vorgang wird als Restitution bezeichnet und der entsprechende Code ist in Listing 4 zu sehen.

```

1 float BMA180::getXValFloat ()
2 {
3     // normalize ( if x is maximum (8191) and GSENSE=1.0 then 1.0
4     return (float)x/8191.0*getGSense ();
5 }
```

**Listing 4:** Restitution

der aufgenommenen Beschleunigungsdaten. Ausgabe als Vielfaches von der Erdbeschleunigung  $g = 9.81m/s^2$

Aus dem Datenblatt [BMA180, 2009] lässt sich entnehmen, dass 10 0000 0000 0000 den geringstmöglichen Wert in jedem Messbereich darstellt und 01 1111 1111 1111 den Größten. Das erste Bit gibt dabei das Vorzeichen an (Zweierkomplement). Teilt man den Wert durch 8191, was in Binärdarstellung 1 1111 1111 1111 entspricht, erhält man nach einer Multiplikation mit dem Messbereich (GSENSE) die Beschleunigung als Vielfaches der Erdbeschleunigung  $g = 9.81m/s^2$ .

### 4.5.3 PC Logger

Das Programm, welches auf dem PC läuft und die Daten aufzeichnet, ist ein Python Script, welches ich von Matthias Hort übernommen und angepasst habe. Sobald es gestartet ist und eine kurze Beschreibung für das aktuelle Experiment eingegeben wurde, nimmt es sämtliche Nachrichten auf dem eingestellten COM-Port entgegen und schreibt diese mit der jeweils aktuellen Systemzeit in eine Datei.

Die exakte Zeit ist sehr wichtig, um die Ergebnisse später mit anderen Messungen vergleichen zu können. Zu diesem Zweck ist der PC mit einem NTP (Network Time Protocol) Server verbunden, welcher stets die korrekte Uhrzeit von GPS-Satelliten empfängt und im Netzwerk zur Synchronisation bereitstellt. Ein Programm auf dem PC (Domain Time II) synchronisiert die computerinterne Uhr mit dieser GPS Zeit.

## **5 Experimente in Waakirchen, erster Einsatz des Prototyp**

Um das System in der Anwendung zu testen, habe ich im Mai 2012 an Eruptionsdynamikexperimenten in Waakirchen, in der Nähe von München, teilgenommen. Diese Experimente wurden gemeinsam mit der Ludwig-Maximilians-Universität München, der Universität Catania und dem Istituto Nazionale di Geofisica e Vulcanologia (Sezione di Catania and Dept. of Seismology and Tectonophysics, Rome) durchgeführt.

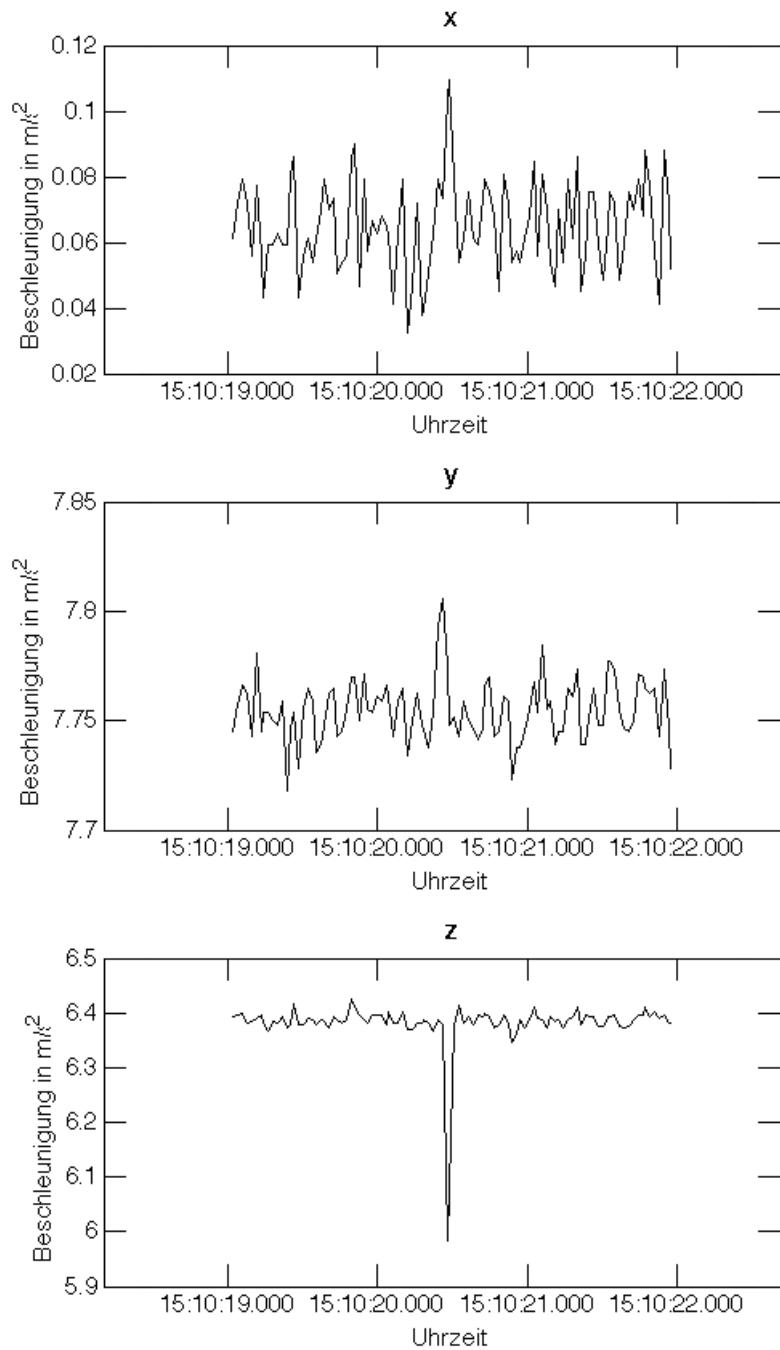
Bei diesen Experimenten wurden künstliche Explosionen in Autoklaven [Spieler et al., 2004] verschiedener Geometrie (von 16 cm bis 40 cm Länge) ausgelöst. Es kamen dabei zwei Aufbauten zum Einsatz, von denen einer das Erhitzen der Proben auf bis zu  $850^{\circ}C$  und damit der Wirklichkeit nähere Umgebungsbedingungen ermöglichte. Bei den verwendeten Samples handelte es sich um Proben von verschiedenen Vulkanen (sowohl Asche als auch festes Lavagestein) und Analogmaterialien. Die Experimente wurden wiederholt durchgeführt und potentielle Einflussgrößen wie Korngröße, Mischverhältnis, Druck und Temperatur variiert.

Untersucht wurde eine Vielzahl von Parametern durch den Einsatz von Hochgeschwindigkeits-, Thermo- und optischen Kameras, Piezo Sensoren, einem akustischen Array und unserem Doppler Radar.

Mein Ziel bei diesem Experiment war es insbesondere, den Prototypen im Feld zu testen und Informationen über die Größenordnung der an der Radarantenne auftretenden Geschwindigkeiten zu gewinnen. Zusätzlich wollte ich die Bewegung des Autoclavenaufbaus untersuchen, um herauszufinden, ob dessen Bewegung Einfluss auf die Messungen haben kann.

## 5.1 Ergebnisse der Messungen in Waakirchen

### 5.1.1 Messungen an der Radarantenne



**Abb. 19:** Restituierte Beschleunigungsdaten in 3 Komponenten in Experiment 7.  
Die X-Achse entspricht der Querachse (Nickachse) des Radars, die Y-Achse seiner Hochachse und die Z-Achse der Längsachse, welche in Radarstrahlrichtung ausgerichtet ist.

In Abbildung 19 sind exemplarisch die Beschleunigungsdaten aus Experiment 7 zu sehen. Die gesamten Experimente befinden sich auf der CD-ROM. Es wurde, technisch bedingt, nur mit 32 Hz aufgenommen und der Messbereich auf  $\pm 1.5g$  eingestellt. Auf der Z-Achse ist ein deutlicher Ausschlag um ca.  $0.4m/s^2$  zu sehen, welcher höchstwahrscheinlich durch die Druckwelle der Explosion zustande kommt.

Die Bewegung der Schüssel wird allerdings lediglich durch ein einziges Sample erfasst. Es ist nicht sichergestellt, ob hier wirklich das Maximum an Beschleunigung abgebildet ist und wie die Bewegung genau verläuft. Eine Bestimmung der Geschwindigkeit oder des Weges durch Integration ist damit praktisch nicht möglich.

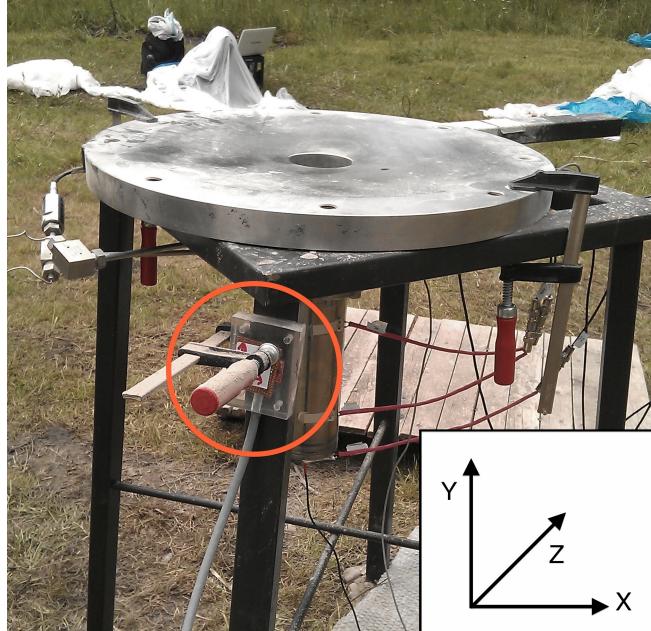
Betrachtet man lediglich das Sample, bei dem die Beschleunigung auftritt und interpoliert linear zu den nächsten beiden Samples, welche noch keine Beschleunigung anzeigen, so lässt sich dennoch eine Geschwindigkeit abschätzen. Diese ergibt sich demnach als:

$$v = \frac{1}{2} a \cdot t \quad (15)$$

Im betrachteten Experiment 7 ist  $v = 0.012m/s$ . Diese Geschwindigkeit ist sehr gering und kann vom Radar nicht aufgelöst werden. Aus den eben genannten Gründen ist dieses Ergebnis vorsichtig zu betrachten. Eine höhere Abtastrate ist notwendig, um zuverlässige Aussagen treffen zu können.

Vergleicht man den Zeitpunkt des Maximalausschlags (15:10:20.465 Uhr) mit dem vom Radar ermittelten Explosionszeitpunkt (15:10:20.274 Uhr), so ergibt sich ein Unterschied von 191 Millisekunden. Diese Differenz lässt sich nicht durch die langsamere Ausbreitungsgeschwindigkeit der Druckwelle erklären. Es gab offensichtlich ein Problem mit der Zeitsynchronisation, so dass das Radar oder der Logger nicht zu jedem Zeitpunkt über eine korrekte GPS Zeit verfügten.

### 5.1.2 Messungen am Autoklavenaufbau



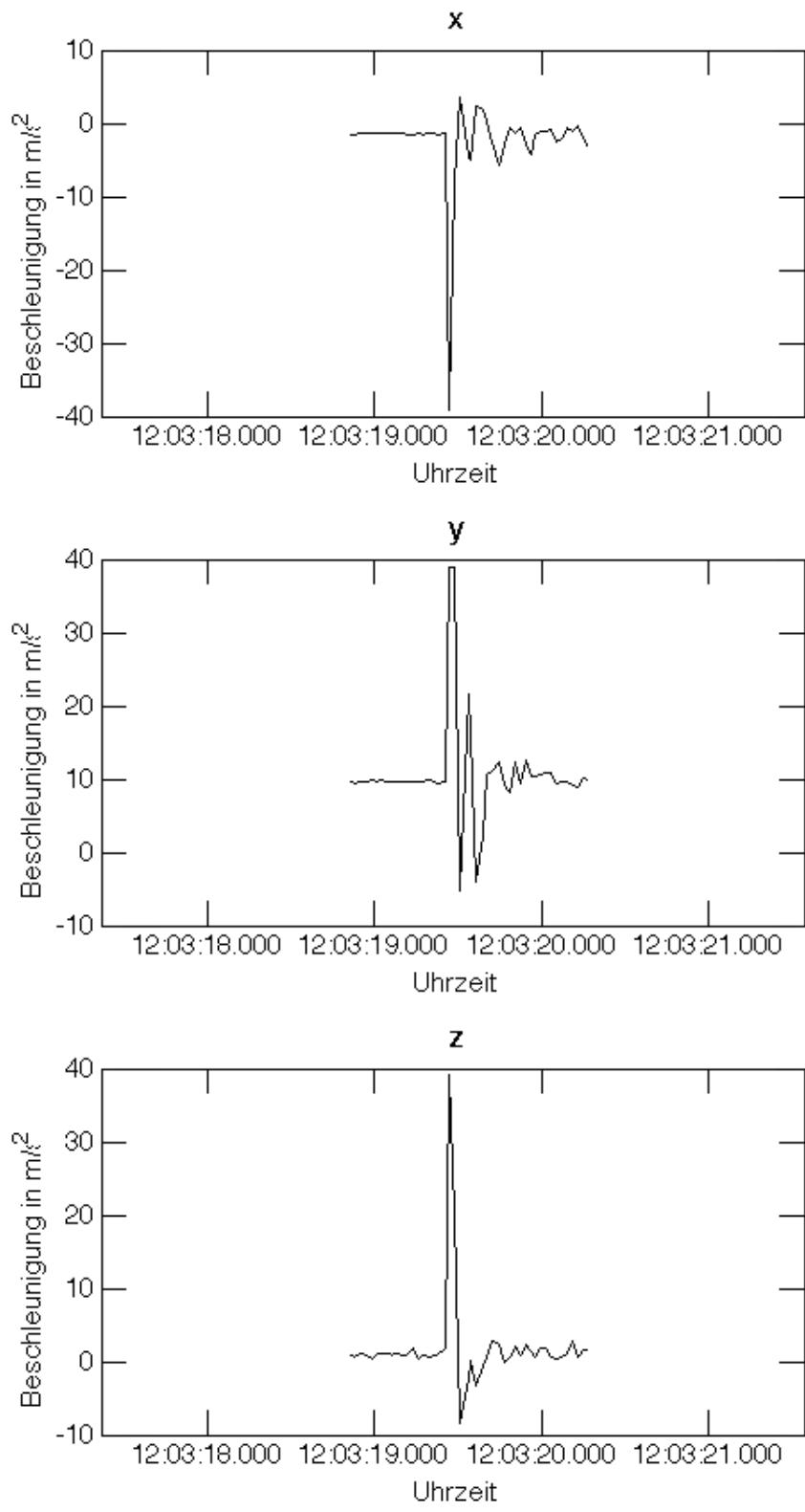
**Abb. 20:** Messaufbau: Bewegungsmessung des Autoklavenaufbaus. Der Beschleunigungssensor ist am Tischbein des Aufbaus befestigt (roter Kreis).

Während Experiment 16 habe ich den Beschleunigungsmesser am Bein des Autoklavenaufbaus befestigt. Die Ausrichtung der Achsen ist in Abbildung 20 verdeutlicht.

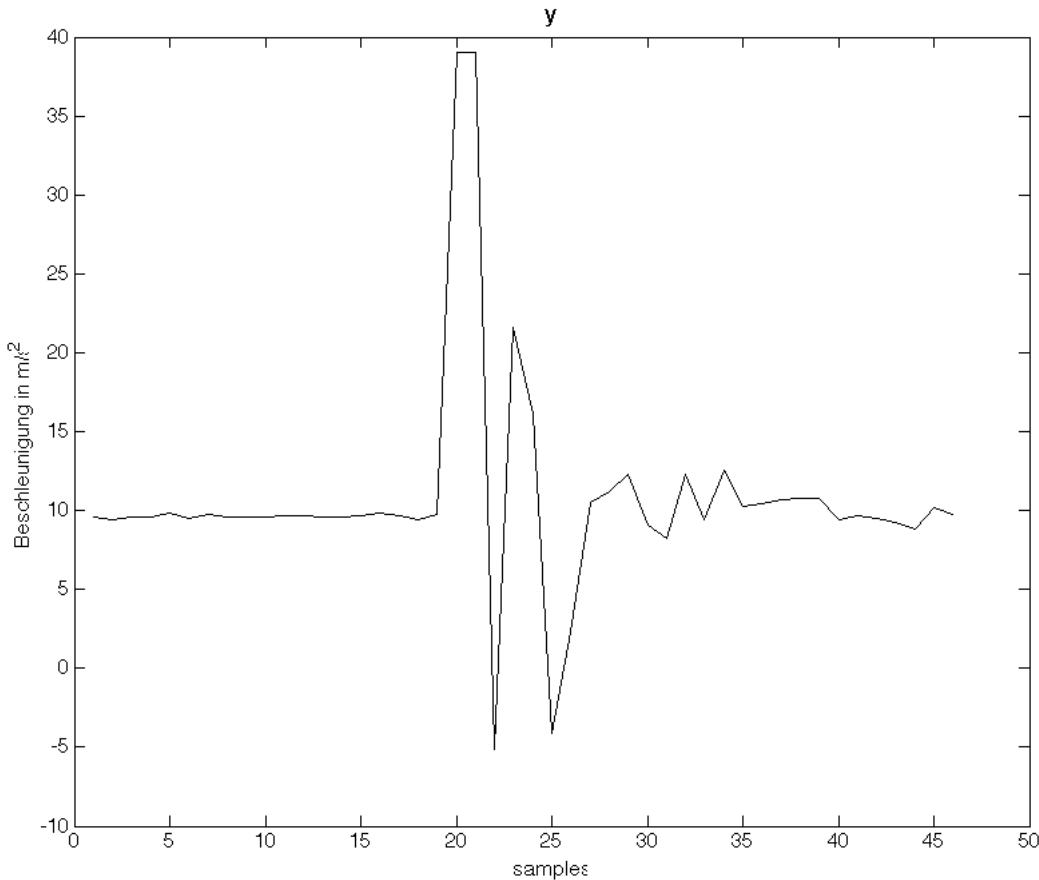
Es wurde wiederum mit einer Aufnahmefrequenz von 32Hz aufgezeichnet und der Messbereich war auf  $\pm 4g$  eingestellt. In Abbildung 21 sind die Ergebnisse zu sehen.

Zum Explosionszeitpunkt sind Beschleunigungen in allen drei Raumrichtungen zu erkennen. Der Messbereich war zu fein gewählt, die aufgetretenen Beschleunigungen sind teilweise größer als  $4g$ .

Zur genaueren Betrachtung der Vertikalbewegung ist in Abbildung 22 nur die Y-Komponente der Beschleunigung aufgetragen.



**Abb. 21:** Restituierte Beschleunigungsdaten am Autoklavenaufbau. Experiment 16. Die Ausrichtung der Achsen kann Abbildung 20 entnommen werden.



**Abb. 22:** Y-Komponente der Beschleunigung. Der Tisch führt eine gedämpfte Schwingung aus. Deutlich erkennbar ist der zu gering gewählte Messbereich, der Erstausschlag ist nach oben hin 'abgeschnitten'

Die Messung zeigt, dass der Tisch eine gedämpfte Schwingung ausführt. Der Erstausschlag ist positiv, also bewegt sich der Tisch zunächst nach oben. Dann fällt er wieder auf den Boden zurück und schwingt ca. eine Sekunde lang nach.

Mit der bei der Messung an der Antennenschüssel bereits angewandten Abschätzung der Geschwindigkeit erhalten wir zu Beginn eine Maximalgeschwindigkeit von ca.  $1.33m/s$ . Der wahre Wert ist größer, da das Maximum der Beschleunigung nicht korrekt erfasst wurde.

Es lässt sich festhalten, dass sich sowohl die Radarschüssel, als auch der Autoklavenaufbau bei einer Explosion bewegen. Die Geschwindigkeiten sind, verglichen mit den gemessenen Radargeschwindigkeiten, aber vernachlässigbar klein.

In beiden Versuchen ist festzustellen, dass die Abtastfrequenz deutlich zu gering ist. Aussagen über die in den Versuchen auftretenden Beschleunigungen und Geschwindigkeiten sind nur bedingt möglich. Die verwendete Hardware ermöglicht zwar eine höhere Abtastrate, doch bei höheren Datenraten wird die Verbindung zwischen Sensor und Logger zunehmend instabil.

Die Auswahl des Messbereichs ist kritisch für ein Gelingen der Messung. Sind die untersuchten Beschleunigungen größer als das Maximum des Messbereichs, übersteuert der Sensor und die Daten werden in den entsprechenden Momenten unbrauchbar. Wird der Bereich jedoch zu groß gewählt, führt dies zu einer wesentlich geringeren Auflösung und kleine Änderungen der Beschleunigung sind nicht mehr zu erkennen.

Um verlässlichere Aussagen über die am Radar auftretenden Bewegungen zu machen, ist es notwendig ein leistungsfähigeres System zu entwickeln.

## 6 Entwicklung des Schwingungsmesssystems: ASC 5511LN-002

Mit den bisher gewonnenen Erkenntnissen konnte ich das zweite Schwingungsmesssystem mit dem ASC Chip (ASC 5511LN-002) realisieren. Durch die höhere Auflösung und Samplerate und geringeres Rauschen, ergibt sich ein besser nutzbares Signal. Den Vergleich der relevanten technischen Daten enthält Tabelle 2.

Beschleunigungsaufnehmer	Auflösung	Noise	Temperatur Drift
BMA180	0.244 mg	$150\mu\text{g}/\sqrt{\text{Hz}}$	0.5 mg/k
ASC 5511LN-002	0.077 mg	$5\mu\text{g}/\sqrt{\text{Hz}}$	0.2 mg/k

**Tab. 2:** Vergleich der relevanten technischen Daten der beiden Sensoren bezogen auf einem Messbereich von  $\pm 2\text{g}$ .

Das Auflösungsvermögen des ASC 5511LN-002 bezieht sich hier auf die Verwendung mit dem *Diamond-MM-16-AT PC/104 Analog I/O Module*

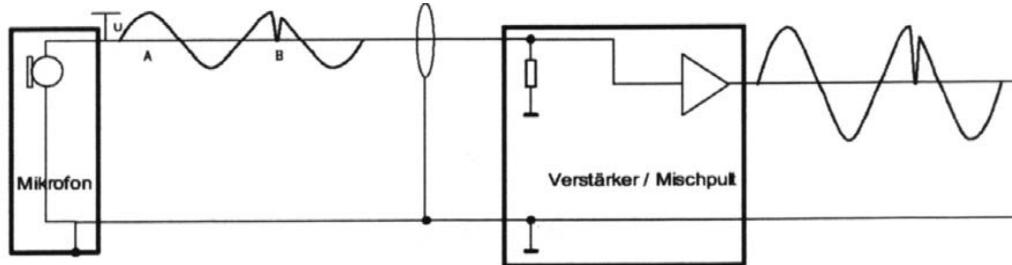
### 6.1 ASC 5511LN-002



**Abb. 23:** ASC 5511LN-010 von Advanced Sensors Calibration (die Zahl am Ende der Typenbezeichnung gibt den Messbereich in g an: im Bild 10g, der von mir verwendete 2g)  
<http://www.asc-sensors.de>

Bei dem ASC 5511LN-002 handelt es sich ebenfalls um einen kapazitativen MEMS Sensor. Er ist allerdings von hochwertigerer Qualität als der BMA180 und gibt das

Signal analog als Spannung aus. Aus diesem Grund wird ein externer AD-Wandler benötigt. Durch das Aluminiumgehäuse ist die Elektronik bereits gut geschützt und die symmetrische Ausführung der Signalkabel sorgt dafür, dass äußere elektromagnetische Einflüsse auf dem Weg zum AD-Wandler vermieden werden. Das Signal wird auf zwei Leitungen ausgegeben, wobei das Signal auf der einen Leitung invertiert ist. Störungen wirken auf beide Adern und somit kann man sie an der Empfangsstation per Differenzbildung leicht entfernen.

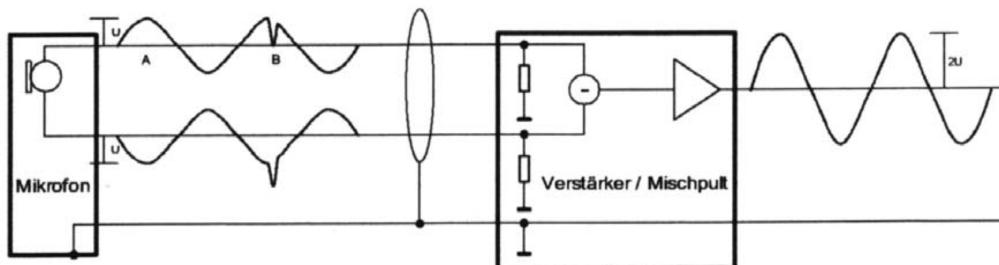


**Abb. 24:** Unsymmetrische Signalübertragung mit Störung 'Zacke'. [Sengpiel, 2001]

Bei unsymmetrischer Signalführung wird das Störsignal ebenso stark verstärkt, wie das Nutzsignal (Abb. 24). Mathematisch lässt sich dies so ausdrücken [Sengpiel, 2001]:

$$U_a = v \cdot (U_e + U_{stoer}) = v \cdot U_e + v \cdot U_{stoer} \quad (16)$$

wobei  $U_a$  dem Ausgangssignal,  $U_e$  dem Eingangssignal,  $U_{stoer}$  dem Störsignal und  $v$  dem Verstärkungsfaktor entspricht.



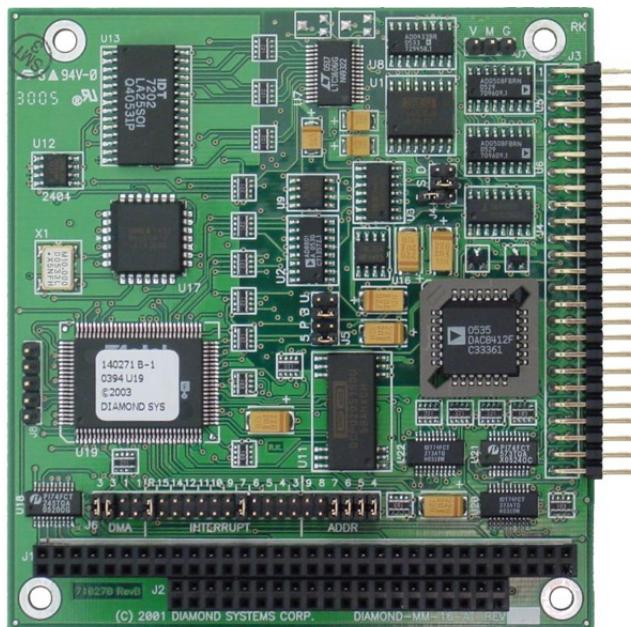
**Abb. 25:** Symmetrische Signalübertragung mit sich aufhebender Störung. [Sengpiel, 2001]

Symmetrische Signalführung (Abb. 25) hingegen sorgt dafür, dass sich der Störinfluss auslöscht:

$$U_a = U_e + U_{stoer} - (-U_e + U_{stoer}) = 2 \cdot U_e \quad (17)$$

Das Resultat ist ein besseres Signal zu Rausch Verhältnis.

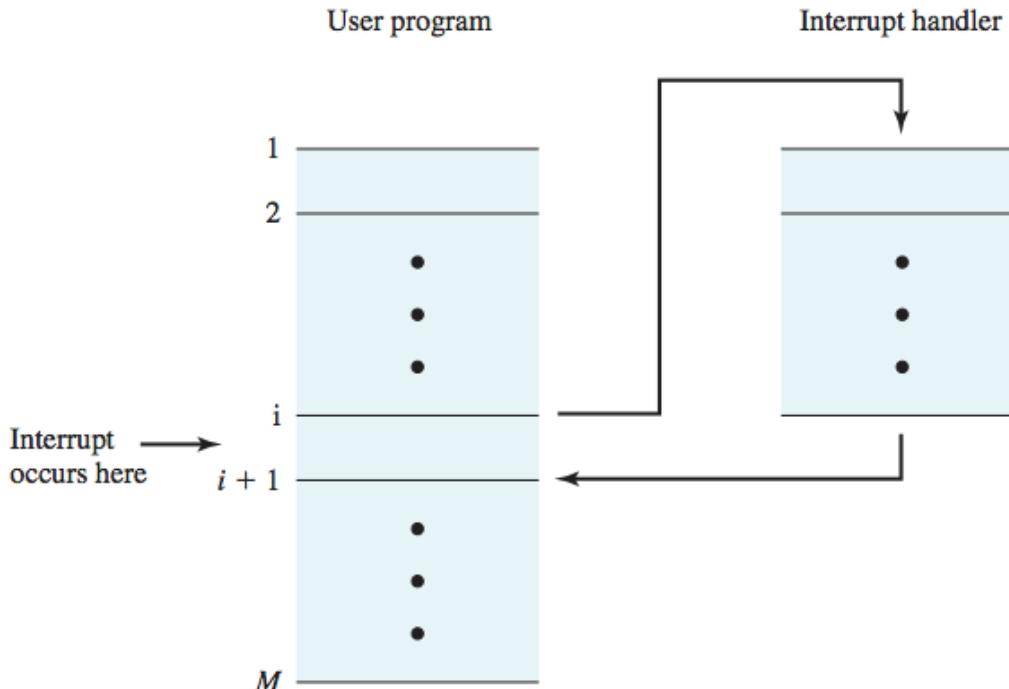
## 6.2 Diamond-MM-16-AT PC/104 Analog I/O Module



**Abb. 26:** Diamond-MM-16-AT PC/104 Analog I/O Module

Als AD-Wandler verwende ich ein *Diamond-MM-16-AT PC/104 Analog I/O Module*. Das Board verfügt über 8 Eingänge (16 wenn man sie nicht als Differentialeingänge verwendet) und eine maximale Samplingrate von 100kHz. Außerdem besitzt es einen 512-Sample *FIFO Speicher* und verwendet *Interrupts*, worauf ich in den beiden nächsten Punkten eingehen möchte. Mit dem Computer kommuniziert das Board über den *PC/104 Bus*.

### 6.2.1 Interrupts



**Abb. 27:** Ablauf eines Interrupts [Stallings, 2000]

Ein Interrupt beschreibt in der Informatik eine kurzzeitige Unterbrechung des normalen Programmflusses, um eine andere Operation auszuführen. Diese andere Operation ist zeitkritisch und ihr exaktes Auftreten nicht vorher bestimmbar. Sobald sie beendet ist, setzt das Programm seine Ausführung an der vorher unterbrochenen Stelle fort.

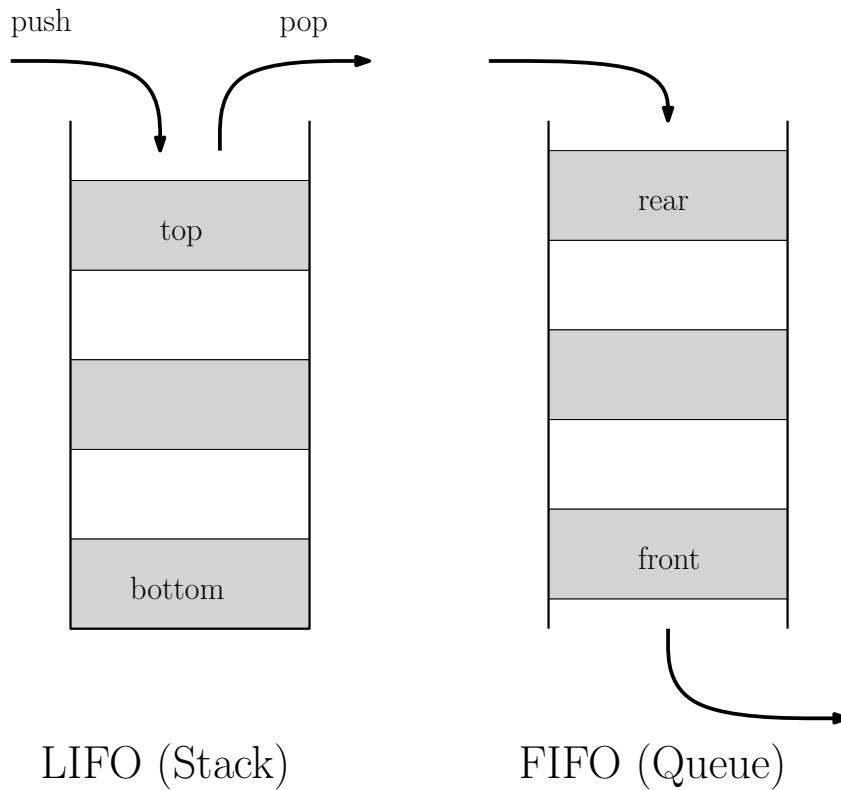
Genutzt werden Interrupts zum Beispiel von Ein- und Ausgabegeräten, wie Maus und Tastatur. Ohne Interrupts müssten alle Programme zyklisch nachfragen, ob es eine neue Eingabe gibt und diese dann entsprechend bearbeiten (sogenanntes Polling). Da ein Programm in dieser Zeit nichts anderes machen kann, ist diese Art der Abfrage höchst ineffizient.

Die am weitesten verbreitete Analogie zur Verdeutlichung des Prinzips ist eine Wohnungstür mit Klingel. Man kann den ganzen Tag in der Wohnung seinen Aufgaben nachgehen und zwischendurch klingeln Gäste, mit denen man sich dann kurz beschäftigt. Das Klingeln ist hier also der Interrupt. Dann kann man wieder mit der Arbeit fortfahren bis der Nächste klingelt. Hat man aber keine Klingel, so muss man ständig zur Tür rennen und nachsehen, ob eventuell jemand dort steht und

gerne hereinkommen möchte (Polling).

An dieser Analogie kann man auch erkennen, dass Interrupts zusätzliche Hardware (eine Klingel) erfordern.

### 6.2.2 FIFO Speicher



**Abb. 28:** Vergleich von FIFO und LIFO

Ein FIFO (First In First Out) ist ein Speicherverfahren, bei dem die Elemente, die als erstes gespeichert werden, auch als erstes wieder entnommen werden. Es wird auch als Warteschlangenprinzip (Queue) bezeichnet und in vielen Bereichen der Datenverarbeitung verwendet. Besonders bei Schnittstellen und Kommunikationsprotokollen ist es wichtig, dass die Daten in der Reihenfolge verarbeitet werden, in der sie angekommen sind, beziehungsweise abgesendet wurden. Das gegensätzliche Prinzip ist das LIFO (Last In First Out), auch Stapelprinzip (engl. Stack) genannt.

Im AD-Wandler sorgt der FIFO dafür, dass wirklich alle digitalisierten Samples vom Computer gespeichert werden, auch wenn dieser aufgrund starker Prozessorbelastung nicht in der Lage ist diese sofort zu verarbeiten. Die digitalisierten Daten

werden unabhängig vom PC immer sofort in den FIFO gespeichert und können dort angefordert werden. Ein Problem ergibt sich erst wenn der PC die Samples so langsam ausliest, dass der FIFO Speicher voll wird. In diesem Fall spricht man von einem Speicherüberlauf und die ältesten bereits aufgenommenen Daten werden überschrieben.

Zusammen mit Interrupts erlaubt der FIFO Speicher eine konstante Messwertdigitalisierung mit hoher Frequenz, ohne den Prozessor des PCs zu stark zu beanspruchen. Dazu werden die Daten kontinuierlich vom Board in den FIFO geschrieben und sobald ein festgelegter Grenzwert (threshold) erreicht ist, löst es einen Interrupt aus. Darauf reagiert nun der PC, indem er das komplette Datenpaket auf einmal (also so viele Samples, wie mit dem threshold Wert eingestellt sind) aus dem FIFO liest und auf seine Festplatte speichert. Währenddessen werden weiterhin Daten digitalisiert und es geht kein Sample verloren.

Würde bei jedem neuen Sample ein Interrupt ausgelöst werden und die Daten jedes Mal einzeln abgerufen und gespeichert, so würde dies bei hohen Abtastfrequenzen zu extremer Prozessorbelastung führen. Der FIFO dient also gleichzeitig auch zur Verringerung der Interrupt Rate:

$$\text{Interrupt Rate} = \frac{\text{A/D Frequenz} \cdot \text{Anzahl der aufgenommenen Kanäle}}{\text{FIFO threshold}} \quad (18)$$

Bei einem Kanal und einer Abtastfrequenz von  $100000\text{Hz}$  resultiert, bei einem threshold von 256, ein Unterschied von 100000 zu ca. 391 Interrupts pro Sekunde.

### 6.3 Software

Für das Diamond Board existiert die *Universal Driver Software*<sup>6</sup>, eine C-basierte Programmbibliothek, die eine große Zahl von Funktionen zur Datenaquisition zur Verfügung stellt. Die Low-Level-Programmierung, bei der auf die einzelnen Register zugegriffen wird, entfällt damit.

Bei der Entwicklung habe ich mich an der *DSCADScanInt Demo*<sup>7</sup> aus den Beispielen der Treibersoftware orientiert. Ich möchte wiederum nur die wichtigsten Teile des Quellcodes erläutern. Der komplette Code findet sich auf der beiliegenden CD-

---

<sup>6</sup><http://www.diamondsystems.com/products/dscud>

<sup>7</sup>[http://files.diamondsystems.com/cdrom/Software/Universal%20Driver/Demos/DMM16-AT\\_5.91\\_Demo.zip](http://files.diamondsystems.com/cdrom/Software/Universal%20Driver/Demos/DMM16-AT_5.91_Demo.zip) (September 2012)

ROM.

Das Programm wird mit zwei Parametern aufgerufen: Dem Namen des Logfiles, in das die Daten geschrieben werden und der Anzahl an Samples, die aufgenommen werden sollen.

Wenn im folgenden von einem Buffer gesprochen wird, so ist damit nicht der FIFO gemeint, sondern ein Speicher, in dem die Samples gespeichert werden, sobald sie aus dem FIFO herauskommen. Dieser Buffer wird nur von Interrupts beschrieben, daher wird zusätzliche Logik benötigt, um die Daten in der richtigen Reihenfolge aus diesem Buffer auszulesen.

Der Programmablauf ist folgendermaßen gegliedert:

1. Initialisierung des Treibers
2. Initialisierung des Boards (mit Bus-Adresse und Interrupt Level)
3. Konfigurationsvariablen für den AD-Wandler setzen
4. Konfigurieren des Interrupts und des FIFOs
5. Loop, der alle x Sekunden überprüft, ob neue Daten vorliegen
  - Daten in der richtigen Reihenfolge aus dem Buffer lesen und samt Zeitstempel in eine Datei schreiben

## 1. Initialisierung des Treibers

Zunächst wird der Treiber für das Board initialisiert (Listing 5), dabei wird überprüft, ob die korrekte Treiberversion installiert ist.

```
1 if( dscInit( DSC_VERSION ) != DE_NONE )  
2 {  
3     dscGetLastError(&errparams);  
4     fprintf( stderr , "dscInit error: %s %s\n" , dscGetErrorString(  
                  errparams.ErrCode) , errparams.errstring );  
5     return 0;  
6 }
```

**Listing 5:** Initialisierung der Treibersoftware

## 2. Initialisierung des Boards

Als nächstes wird das Board selbst initialisiert (Lst. 6). Dazu notwendig sind die Bus-Adresse (Zeile 3) und das Interrupt-Level (Zeile 4), die auf dem Board per Jumper eingestellt werden können. Es ist dadurch möglich mehrere Boards gleichzeitig in einem Computer zu betreiben.

```
1 printf( "\nDMMI16AT BOARD INITIALIZATION:\n" );
2
3 dscrb . base _ address = 0x300 ;
4 dscrb . int _ level = (BYTE) 7;
5
6 if( dscInitBoard(DSC_DMMI16AT, &dscrb , &dsrb )!= DE_NONE)
7 {
8     dscGetLastError(&errparams );
9     fprintf( stderr , "dscInitBoard error: %s %s\n" , dscGetString( errparams . ErrCode ) , errparams . errstring );
10    return 0;
11 }
```

**Listing 6:** Initialisierung des Boards

Der Ablauf der Konfiguration ist in allen Punkten einheitlich. Zunächst werden die entsprechenden Werte in eine Struktur geschrieben, dann wird diese als ganzes an das Board übergeben. Sollten dabei Fehler auftreten, so werden diese ausgegeben und das Programm bricht ab.

## 3. Setzen der Konfigurationsvariablen für den AD-Wandler

Der AD-Wandler kann in verschiedenen Messbereichen arbeiten und außerdem eine gewisse Signalverstärkung (gain) vornehmen. Die dazu notwendigen Einstellungen sind in Listing 7 aufgeführt. Der *ASC 5511LN-002* gibt eine maximale Spannung von 5V aus, daher ist RANGE\_5 der richtige Eintrag. In unserem Fall muss die Spannung auch nicht verstärkt werden.

Über eine längere Zeit und vor allem mit der Änderung der Temperatur neigen AD-Wandler dazu, eine gewisse Drift aufzuweisen. Dies liegt daran, dass sich der Ohmsche Widerstand, welcher als Referenz zur gemessenen Spannung dient, mit der Temperatur ändert. Das *Diamond-MM-16-AT* verfügt über einen Autokalibrationsmechanismus. Indem es vorher exakt bekannte Ohmsche Widerstände misst und die Messungen mit den bekannten Werten vergleicht, kann die genaue Drift bestimmt und herausgerechnet werden [siehe auch Miller, 2006].

Ob diese Autokalibration genutzt werden soll, legt `load_cal` in Zeile 4 fest.

Zeile 5 setzt den aktuellen Kanal, bei dem die Digitalisierung beginnen soll fest.

```

1 dscadsettings.range = RANGE_5;
2 dscadsettings.polarity = UNIPOLAR;
3 dscadsettings.gain = GAIN_1;
4 dscadsettings.load_cal = (BYTE)TRUE;
5 dscadsettings.current_channel = 0;
```

**Listing 7:** Konfigurieren des AD-Wandlers

#### 4. Konfigurieren des Interrupts und des FIFOs

An dieser Stelle (Lst. 8) wird festgelegt, dass der FIFO benutzt werden soll (Zeile 9) und wie groß der threshold ist, bei dem ein Interrupt ausgelöst wird (Zeile 11).

Die Größe des FIFOs (`fifo_depth`) ist durch die Hardware auf 256 Byte festgelegt.

`Low_channel` (Zeile 5) und `high_channel` (Zeile 6) geben an, welche der 8 Eingänge des Boards genutzt werden sollen. Obwohl der Beschleunigungssensor nur 3 Komponenten hat, verwende ich hier 4 Kanäle, da dies die Ausleselogik vereinfacht. In Zeile 2 wird die Samplingrate festgelegt. Sie ist immer auf alle Kanäle bezogen, so dass sich bei einer Samplingfrequenz von 1000 Hz und 4 Kanälen eine effektive Abtastrate von 250Hz pro Kanal ergibt.

Theoretisch lassen sich mit dem Board Samplingraten von bis zu 100 000Hz erreichen. Dazu ist jedoch eine genaue Kalibrierung aller Parameter und weitere Codeoptimierung notwendig, was nicht mehr Teil dieser Arbeit sein soll.

```

1 dscaioint.num_conversions = num_conversions;
2 dscaioint.conversion_rate = 1000;
3 dscaioint.cycle = (BYTE)TRUE;
4 dscaioint.internal_clock = (BYTE)TRUE;
5 dscaioint.low_channel = 0;
6 dscaioint.high_channel = 3;
7 dscaioint.external_gate_enable = (BYTE)FALSE;
8 dscaioint.internal_clock_gate = (BYTE)FALSE;
9 dscaioint fifo_enab = (BYTE)TRUE;
10 dscaioint fifo_depth = 256;
11 dscaioint.dump_threshold = 256;
```

**Listing 8:** Setzen der Variablen für den Interrupt

In Zeile 3 lässt sich zwischen *One-Shot-Mode* und *Recycle-Mode* wechseln. Im One-Shot-Mode wird ein Buffer ein Mal mit Daten gefüllt und danach beendet sich die

Routine. `Num_conversions` gibt dabei die Größe des Buffers an, und damit wie viele Samples aufgenommen werden sollen.

Im Recycle-Mode wird bei Erreichen des Bufferendes zum Anfang des Buffers gesprungen und dort die alten Daten überschrieben (Ringbuffer Prinzip). Der Parameter `num_conversions` gibt auch in diesem Fall die Größe des Buffers an, hat jedoch nichts mehr mit der Anzahl maximal aufzunehmender Samples zu tun. So mit enthält der Buffer immer die letzten  $n$  Samples, wobei  $n$  die Buffergröße ist.

Um herauszufinden, an welcher Stelle im Buffer gerade geschrieben wurde, dienen die beiden Variablen `DSCS.transfers` und `DSCS.total_transfers`.

`DSCS.transfers` gibt an, wie viele Samples im aktuellen Zyklus bereits im Buffer gespeichert wurden und wird immer auf 0 gesetzt wenn ein neuer Zyklus beginnt. `DSCS.total_transfers` gibt die Gesamtzahl der bisher digitalisierten Werte in der gesamten Operation an. Damit ist es möglich, die Daten in der korrekten Reihenfolge in eine Datei zu speichern.

Die weiteren in Listing 8 aufgeführten Variablen sind für unseren Anwendungsfall nicht relevant.

## 5. Auslesen und Speichern der Daten

Wie bereits beschrieben verläuft das Speichern der neuen Samples in den Ringbuffer automatisch durch die Interrupts. Sobald der `dump_threshold` erreicht ist, werden die Daten vom FIFO in den Buffer übertragen. Um sie von dort aus in eine Datei zu schreiben, bedienen wir uns einer Programmschleife, die mehrmals pro Sekunde überprüft, ob neue Samples vorliegen (Lst. 9).

```
1 DWORD sleep_ms = 300;
2
3 do {
4     dscSleep (sleep_ms) ;
5     dscGetStatus (dscb , &dscs ) ;
6
7     if ( dscs . overflows ) {
8         printf("Operation failed : FIFO overflowed\n") ;
9         break ;
10    }
11
12    if ( dscs . total_transfers == last_total_transfers ) {
13        printf("Operation failed : no new samples taken in %d ms\n"
```

```

14         , sleep_ms) ;
15     break;
16 }
17 new_sample_count = dscs.total_transfers - last_total_transfers
18 ;
19 /* Number of new samples should never exceed the size of the
   circular buffer. If it does it means that either "
   sleep_ms" should be smaller so you check status more often
   , or "num_conversions" should be bigger so the circular
   buffer is bigger */
20 if ( new_sample_count > num_conversions ) {
21     printf("Operation failed: not processing data fast enough.
           %d samples lost\n",
22           new_sample_count - num_conversions);
23     break;
24 }
25
26 /* AUSLESEN UND SPEICHERN DER DATEN */
27 /* FOLGT IM NAECHSTEN LISTING */
28
29 last_transfers = dscs.transfers;
30 last_total_transfers = dscs.total_transfers;
31
32 if ( dscs.total_transfers >= stop_after_transfers )
33     break;
34
35 } while ( dscs.op_type != OP_TYPE_NONE );
36
37 dscCancelOp(dscb);

```

**Listing 9:** Programmschleife, die den Buffer auf neue Daten überprüft und diese in eine Datei schreibt. Der Code zum Auslesen und Schreiben der Samples ist in diesem Listing entfernt und separat in Listing 10 und 11 abgebildet.

In Zeile 1 wird mit `sleep_ms = 300` festgelegt, dass die Schleife ca. 3 Mal pro Sekunde ausgeführt wird. Zunächst wird überprüft, ob der FIFO übergelaufen ist (Zeile 7), es überhaupt neue Samples seit des letzten Schleifendurchlaufs gab (Zeile 12) und ob die Daten rechtzeitig aus dem Buffer in eine Datei geschrieben wurden (Zeile 17 und 20).

Schlägt einer dieser Checks fehl, so wird das Programm mit einer entsprechenden

Fehlermeldung beendet.

Treten keine Fehler auf, werden die Daten aus dem Ringbuffer herausgelesen und in eine Datei gespeichert. Dies geschieht so lange, bis die Zahl der aufzunehmenden Samples erreicht ist (Zeile 32 und 33).

Da der Buffer zirkulär ist, also beim Erreichen des Endes wieder vom Anfang beschrieben wird, gibt es zwei Möglichkeiten wo die neuen Daten darin gespeichert wurden. Im einfachen Fall befinden sie sich zwischen der zuletzt ausgelesenen Position und dem Bufferende. In diesem Fall wird der Code in Listing 10 ausgeführt.

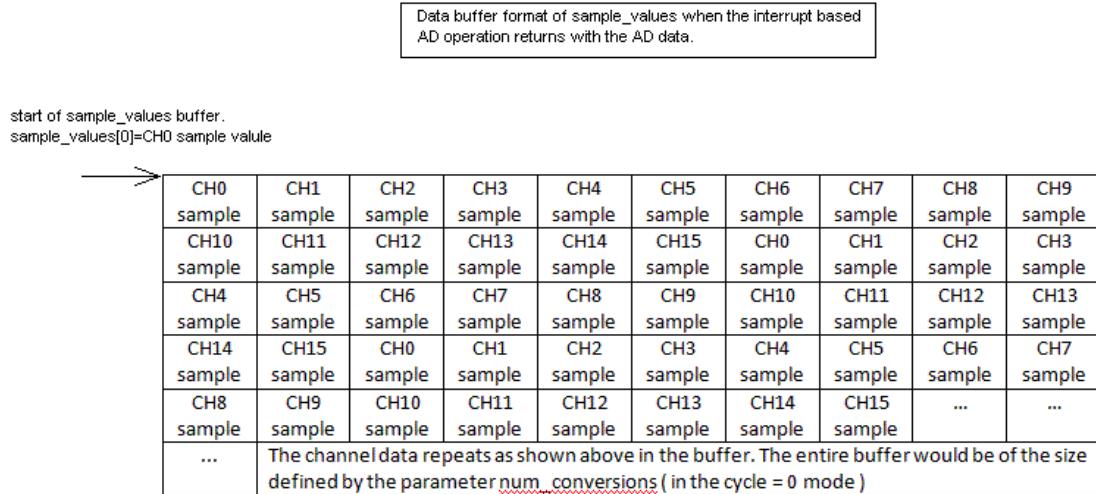
```

1 if ( dscs.transfers > last_transfers ) {
2   for ( i = last_transfers; i < dscs.transfers; i++ ) {
3     if ( i%4 == 0 ) {
4       uhr = mvTime::Now(); // get actual time
5       for ( k = 0; k < 3; k++ ) { // conversion into volts
6         dscADCodeToVoltage(dscb, dscadsettings, dscaoint.
7           sample_values[ i+k ], &voltage[ k ] );
8       }
9       fprintf(logFile, "%s counts: %5d %5d %5d  Volts: %5.3lf %5.3
1f %5.3lf\n", uhr.GetTimeStamp().c_str(), dscaoint.
sample_values[ i ], dscaoint.sample_values[ i+1 ], dscaoint.
sample_values[ i+2 ], voltage[ 0 ], voltage[ 1 ], voltage[ 2 ] );
}

```

**Listing 10:** Auslesen und Speichern der Daten. Fall 1: Im zirkulären Buffer wurden neue Samples nur zwischen der zuletzt ausgelesenen Position und dem Ende des Speichers geschrieben.

In Zeile 1 findet die Überprüfung statt, ob die aktuelle Position im Buffer (`dscs.transfers`) größer ist als die zuletzt ausgelesene Position `last_transfers`, also Fall 1 eingetreten ist. Um die nächsten Zeilen zu verstehen, ist es wichtig zu wissen, wie die einzelnen Samples im Buffer strukturiert sind. Abbildung 29 zeigt diese Struktur.



The entire buffer is contiguous with each sample being a 2 byte value. As can be seen from the pictorial representation of the sample buffer, the data pattern repeats in the buffer.

Abb. 29: Struktur des Buffers, in dem die Samples stehen. [Systems, 2008]

Alle Kanäle werden hintereinander gespeichert. Um diese nun nach Kanal sortiert auszugeben, nutze ich in Zeile 3 den Modulo Operator damit  $i$  immer Kanal 0 enthält und die anderen Kanäle mit  $i + 1$  und  $i + 2$  auszulesen sind.

Die Werte müssen noch von AD-Counts mittels der Funktion `dscADCodeToVoltage` in Volt umgewandelt werden (Zeilen 5 bis 7).

Das Schreiben in eine Datei findet in Zeile 8 statt. Dabei werden sowohl die umgewandelten Voltzahlen als auch die ursprünglichen Counts gespeichert und mit einem Zeitstempel versehen.

Im zweiten Fall wurden neue Daten sowohl nach der zuletzt ausgelesenen Position geschrieben, als auch wieder an den Anfang des Buffers. Fall 2 ist in Listing 11 zu sehen und beinahe identisch mit Fall 1.

Es gibt allerdings zwei for Schleifen. Die erste liest Samples, die nach der zuletzt gelesenen Position gespeichert wurden (Zeile 2) und die zweite liest die Samples, welche sich zwischen dem Anfang und der aktuellen Position befinden (Zeile 10).

```

1 } else if ( dscs.transfers <= last_transfers ) {
2   for ( i = last_transfers; i < num_conversions; i++ ) // after the
      last_transfers
3   if ( i%4 == 0 ) {
4     uhr = mvTime::Now(); // get actual time
5     for ( k = 0; k < 3; k++ ) { // conversion into volts

```

```

6         dscADCodeToVoltage(dscb, dscadsettings, dscaioint.
7             sample_values[i+k], &voltage[k]);
8         }
9     }
10    fprintf(logFile, "%s counts: %5d %5d %5d Volts: %5.3lf %5.3
11        lf %5.3lf \n", uhr.GetTimeStamp().c_str(), dscaioint.
12        sample_values[i], dscaioint.sample_values[i+1], dscaioint
13        .sample_values[i+2], voltage[0], voltage[1], voltage[2]);
14    }
15   for ( i = 0; i < dscs.transfers; i++ ) // in between beginning and
16       last transfers
17   if ( i%4 == 0 ) {
18       uhr = mvTime::Now(); // get actual time
19       for ( k = 0; k < 3; k++ ) { // conversion into volts
20           dscADCodeToVoltage(dscb, dscadsettings, dscaioint.
21               sample_values[i+k], &voltage[k]);
22       }
23       fprintf(logFile, "%s counts: %5d %5d %5d Volts: %5.3lf %5.3
24           lf %5.3lf \n", uhr.GetTimeStamp().c_str(), dscaioint.
25           sample_values[i], dscaioint.sample_values[i+1], dscaioint
26           .sample_values[i+2], voltage[0], voltage[1], voltage[2]);
27   }
28 }
```

**Listing 11:** Auslesen und Speichern der Daten. Fall 2: Im zirkulären Buffer wurden neue Samples zwischen der zuletzt ausgelesenen Position und dem Ende und zusätzlich am Anfang des Speichers geschrieben.

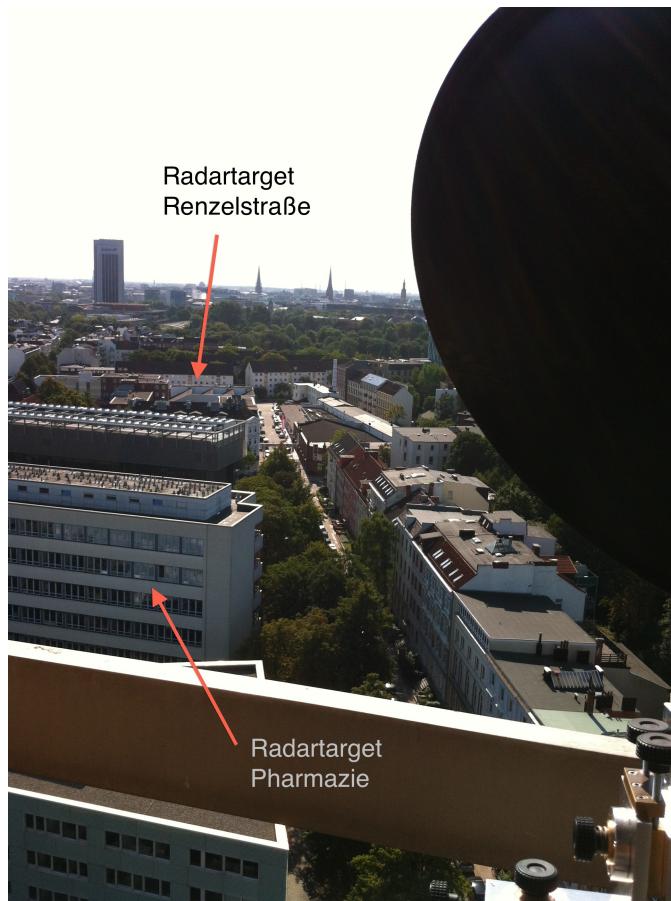
Die Zeitstempel erhalte ich durch eine Methode aus der `mvTime` Klasse. Diese stammt aus der Radarserver Software von Malte Vöge<sup>8</sup>.

Der Nachteil bei dieser Art der Datenaufnahme ist, dass ein neuer Zeitstempel nur bei jedem Schleifendurchlauf (drei Mal pro Sekunde) entsteht. Es haben daher immer alle neu aufgenommenen Samples den Zeitstempel, der bei der Auslösung des Interrupts aktuell ist. In der Auswertung muss also zunächst ein neuer Zeitstempel generiert werden.

---

<sup>8</sup>Die entsprechenden Klassen befinden sich auf der CD

## 7 Experimente am Geomatikum



**Abb. 30:** Ausrichtung des Radars auf ein Haus in der Renzelstraße bzw. auf ein Gebäude der Pharmazie

Der Zweite Sensor wurde auf dem Balkon des 13. Stocks des Geomatikums getestet. Gemeinsam mit Lea Scharff habe ich das Radar zunächst auf ein etwa 400m entferntes Haus in der Renzelstraße ausgerichtet, später auch auf das Gebäude der Pharmazie in etwa 100m Entfernung. Der Sensor war in der Mitte der Radarschüssel fixiert.

Wir haben mit dem Radar eine Phasenmessung durchgeführt, um kleine Änderungen in der Entfernung zwischen Radarsystem und dem jeweiligen Gebäude aufzunehmen. Dann haben wir mit der Hand an der Radarschüssel gerüttelt und später das Radar auf einem Verschiebetisch vor und zurück bewegt.

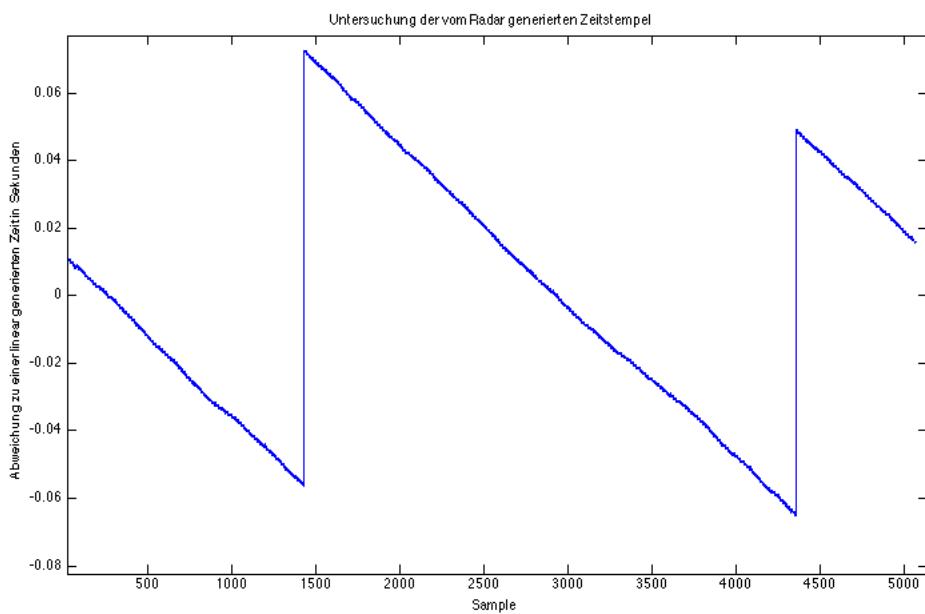
Da die Gebäude unbeweglich sind, sollten sich die Bewegungsdaten des Beschleunigungssensors und die Phasendaten entsprechen.

Im folgenden betrachte ich ausschließlich die Bewegung in Richtung des Radarstrahls, da sich nur diese direkt vergleichen lässt.

Nach dem Import der Daten versehe ich die Beschleunigungsdaten zunächst mit neuen Zeitstempeln, indem ich zwischen den nur drei Mal pro Sekunde erneuerten Werten linear interpoliere. Anschließend lässt sich aus den Daten der gleiche Ausschnitt wählen und untersuchen.

### Probleme mit den Zeitstempeln

Ebenso wie bei den Experimenten in Waakirchen gibt es ein Problem mit den Zeitstempeln der Daten. Die Radardaten enthalten normalerweise Informationen über die Genauigkeit der Synchronisation der internen Uhr mit der vom NTP-Server bereitgestellten GPS-Zeit. Leider fehlen diese Informationen, was nahelegt, dass das Radar zum Zeitpunkt der Messung noch keine erfolgreiche Synchronisation durchgeführt hat.



**Abb. 31:** Vergleich der Zeitstempel des Radars mit einem generierten linear verlaufenden Zeitvektor.

Vergleicht man den Zeitstempel des Radars mit einer linear verlaufenden Zeit, so erhält man Unterschiede von bis zu 80ms in einigen Samples (Abb 31). Ähnlich sieht es bei den Zeitstempeln des Beschleunigungssensors aus.

Eine mögliche Erklärung ist, dass während der Messungen die Computeruhren von

Domaintime angepasst wurden.

Damit diese Anomalien keinen Einfluss auf die Datenbearbeitung haben, habe ich in Matlab einen neuen Zeitvektor generiert, indem ich zwischen dem ersten und letzten Zeitpunkt der Messung linear interpoliert habe. Dieser wird bei den Berechnungen verwendet.

Das Problem mit der Zeitsynchronisation muss auf jeden Fall noch näher untersucht werden.

### **Vergleich von aufgenommenen und berechneten Beschleunigungsdaten**

Um zu überprüfen, ob sich die aufgenommenen Daten ähneln, bilde ich die zweifache Ableitung der Phasendaten des Radars (nach dem neu definierten Zeitvektor) und erhalte damit eine Beschleunigung. In Abbildung 32 ist diese verglichen mit der vom ASC Sensor aufgenommenen Beschleunigung zu sehen. Dabei habe ich aus den Daten des ASC Sensors mit einem 40Hz Tiefpassfilter ein wenig hochfrequenten Noise entfernt.

Vor allem zu Beginn der Messreihe sind sich die Daten sehr ähnlich. Zu späteren Zeitpunkten sind die aus den Phasendaten berechneten Werte deutlich größer.

Die Ähnlichkeit der Daten zeigt, dass die Datenaufnahme und Restitution korrekt durchgeführt wurde. Außerdem wird deutlich, dass die vom Radar gemessene Entfernungsänderung im direkten Zusammenhang mit der Bewegung des Radars steht.

### **Integration der Beschleunigung zu Wegdaten**

Um aus den Beschleunigungsdaten des ASC Sensors Bewegungen des Radars zu errechnen, müssen diese zweifach über die Zeit integriert werden (siehe Kapitel 3.1).

Dabei wird das in Abschnitt 3.2 bereits besprochene Problem der sich aufaddierenden Fehler deutlich. Schon bei der einfachen Integration zur Geschwindigkeit ist eine Drift zu sehen (Abb. 33), die deutlich größer ist als das eigentliche Signal. Integriert man ein weiteres Mal, führt das dazu, dass das Bewegungssignal komplett von der Drift überlagert ist.

Das Ergebnis ist in Abbildung 34 zu sehen. Die berechnete Position ist nach dem Messzeitraum von ca. 3 Minuten auf  $-140m$  gefallen, was ein sehr unrealistisches Ergebnis ist.

Um dieses Verhalten zu verhindern, nehme ich an, dass die 'wahre' Geschwindigkeit im Allgemeinen um Null oszilliert. Indem ich einen gleitenden Mittelwert (running mean) bilde, und diesen von den Geschwindigkeitswerten abziehe, erreiche ich eine Oszillation um Null [vlg. Neitzel et al., 2007]. Die Größe des Filterfensters ist so zu wählen, dass der unerwünschte Trend aus den Daten entfernt wird, das Signal aber intakt bleibt. Ich habe es durch Ausprobieren auf 300 Samples festgelegt.

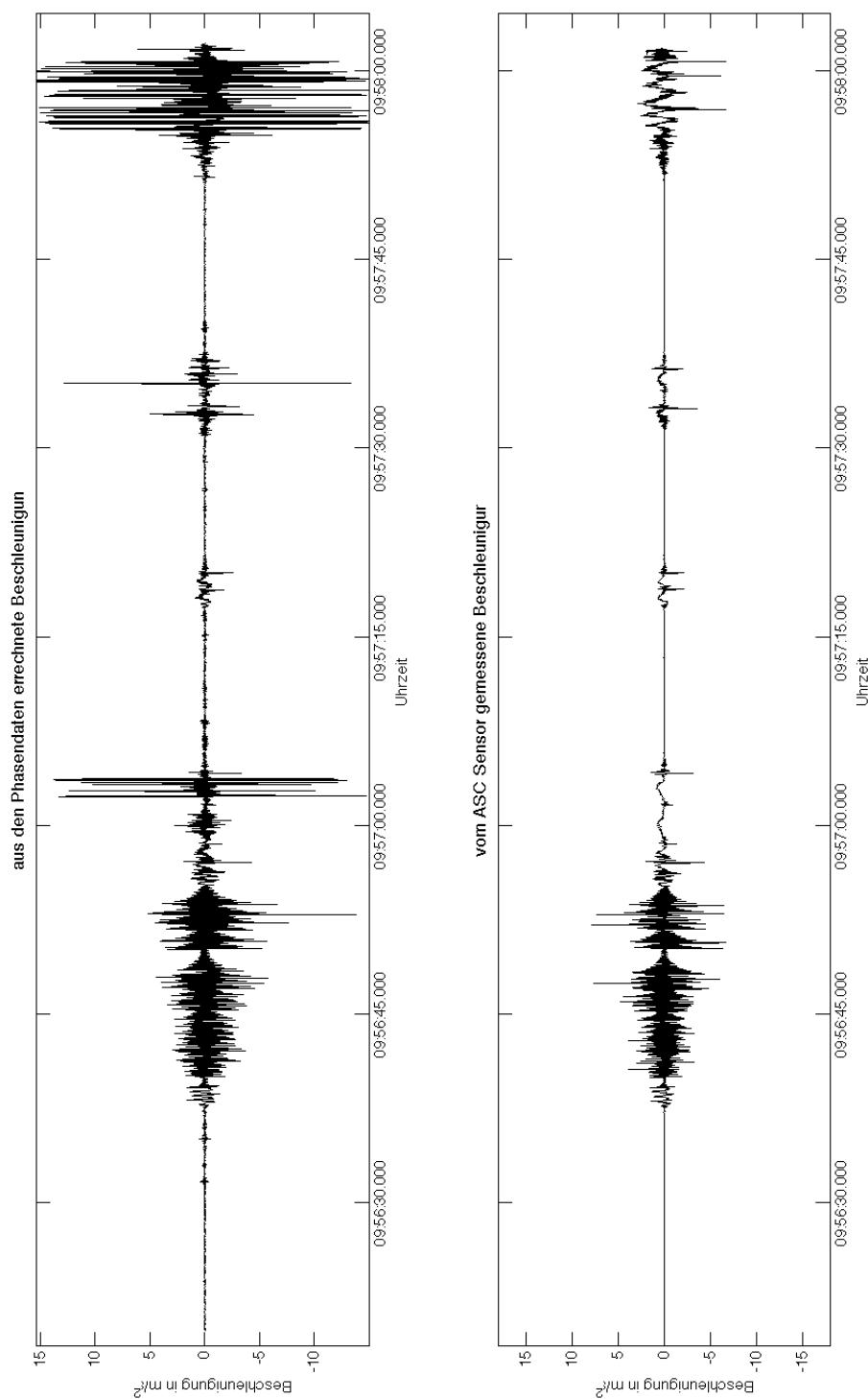
In Abbildung 35 sind die so gebildeten Geschwindigkeiten mit den einmal abgeleiteten Phasendaten verglichen. Eine Kreuzkorrelation ergibt zwar nur eine maximale Übereinstimmung von 0.25 (bei einer Normierung auf 1), qualitativ lassen sich einzelne Strukturen aber sehr gut wiedererkennen.

Integriert man die bearbeiteten Geschwindigkeitsdaten, erhält man die in Abbildung 36 sichtbaren Positionsdaten.

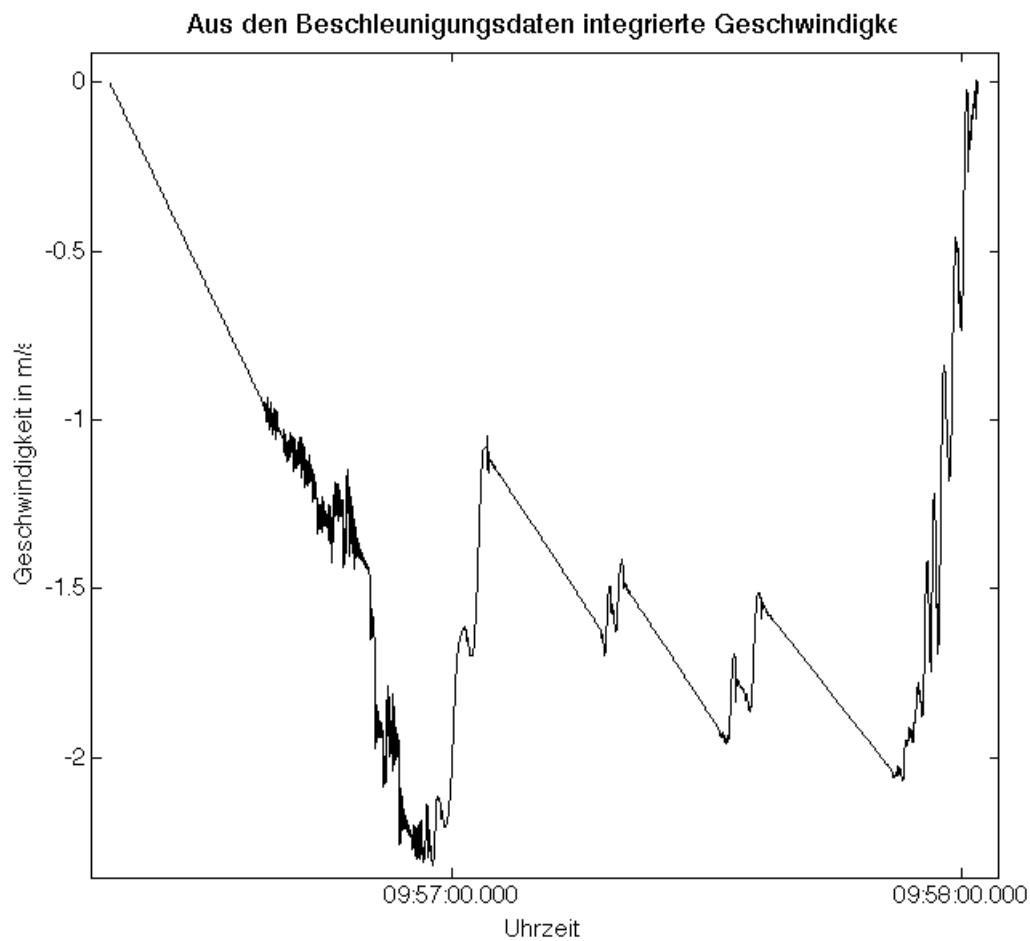
Es fällt sofort auf, dass durch die Entfernung des Trends aus den Geschwindigkeitsdaten auch wirklich vorhandene Trends verloren gehen. Das führt dazu, dass nur kurzeitige Änderungen in der Position erfasst werden können. Dauerhafte Änderungen in der Entfernung zum Objekt werden entfernt.

Dazu kommt das Problem der nicht erfassten Positionsänderung des Radars in die beiden anderen Raumrichtungen. Da die Fläche des Zielobjekts nicht rechtwinklig zum Radarstrahl verlief, kann ein kleines Verdrehen des Radars bereits eine sehr große Distanzänderung verursachen.

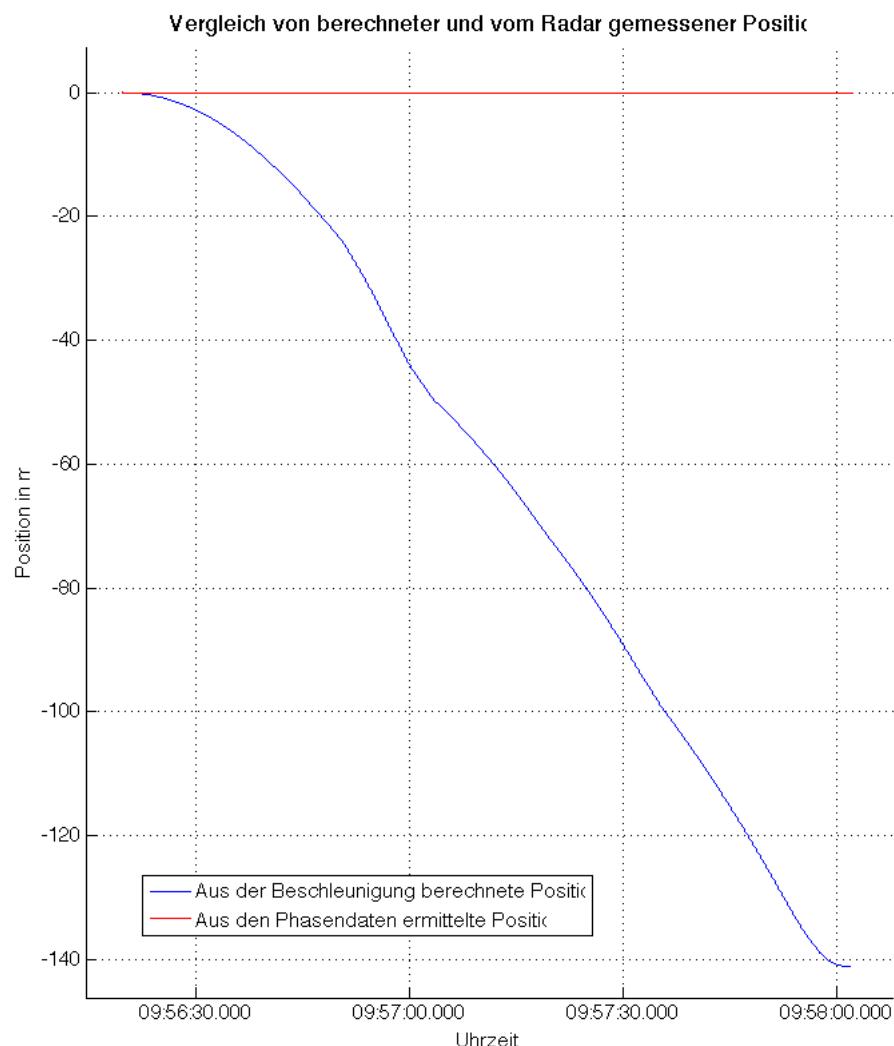
Das macht es natürlich schwierig, mit diesen Daten die gemessenen Phasendaten zu korrigieren, um die Bewegung des Radars gänzlich herauszurechnen. Dennoch können die Daten helfen, bei einer mit dem Radar aufgenommenen Bewegung zu ermitteln, ob diese möglicherweise durch eine Bewegung des Radarsystems zu erklären sind. Ist beispielsweise in den Beschleunigungsdaten keinerlei Bewegung zu sehen, so ist gezeigt, dass die gemessene Bewegung auf eine Bewegung des Messobjekts zurückzuführen ist.



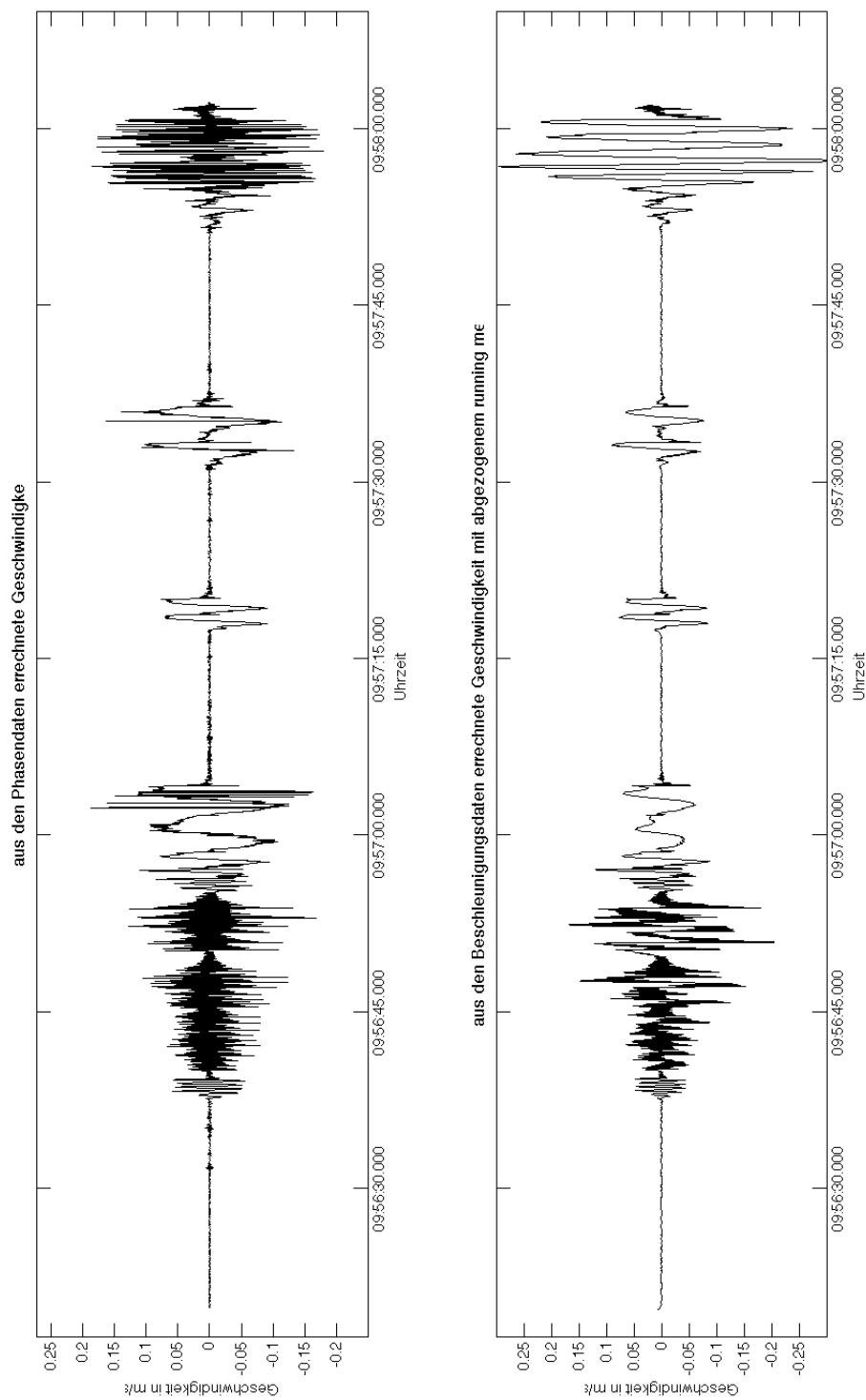
**Abb. 32:** Vergleich von Beschleunigungsdaten aus Experiment r4.  
 Links: aus Phasendaten mittels zweifacher Ableitung berechnete Beschleunigung.  
 Rechts: vom ASC Sensor aufgenommenen Beschleunigung (bei 40Hz Tiefpass gefiltert).



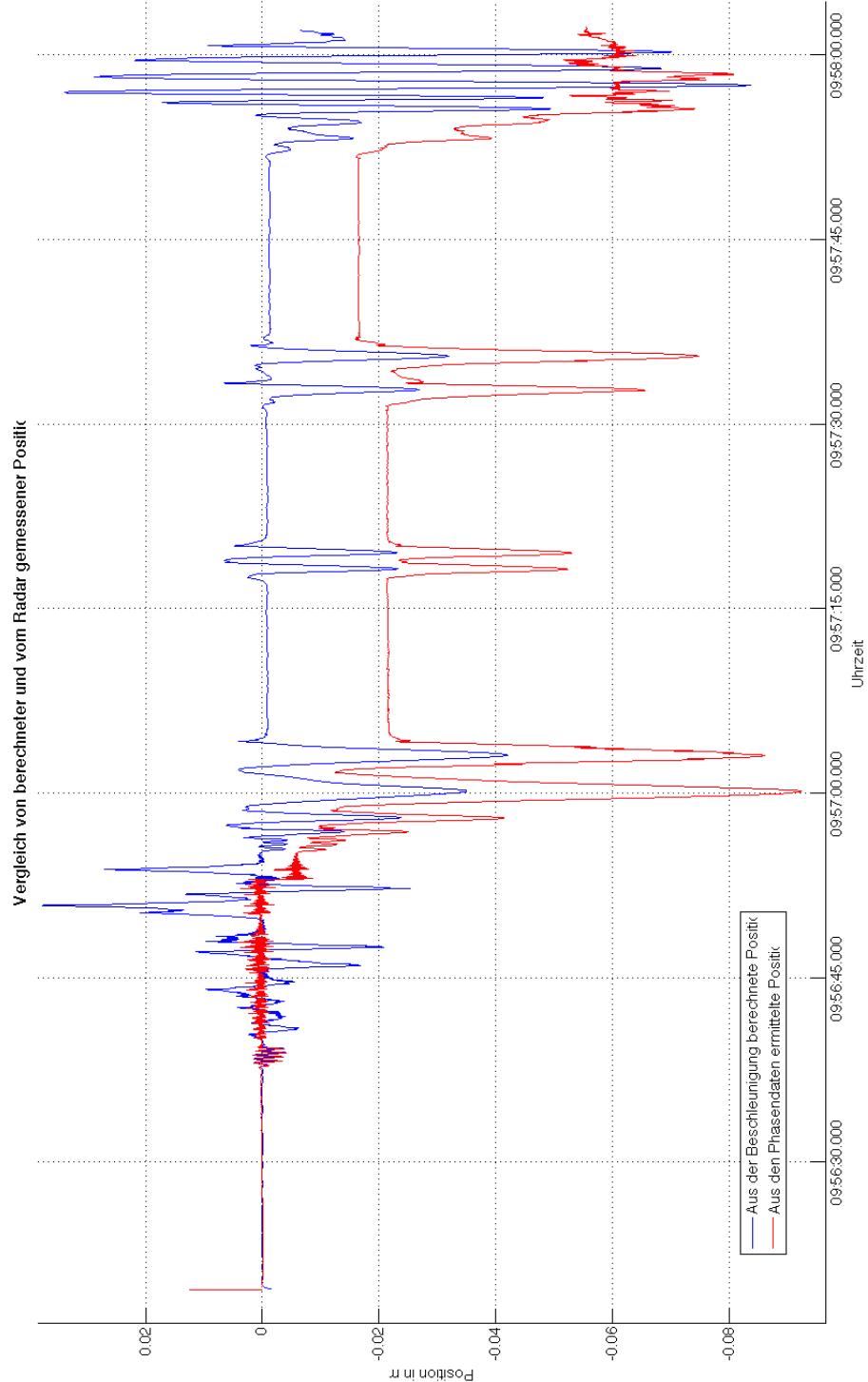
**Abb. 33:** Aus der gemessenen Beschleunigung berechnete Geschwindigkeit. Eine deutliche Drift ist zu erkennen. Die Geschwindigkeit ist über den gesamten Messzeitraum negativ



**Abb. 34:** Berechnete Position (blau) verglichen mit der vom Radar ermittelten Position (rot). Die berechnete Position ist um so viel größer, dass die Variation der vom Radar ermittelten Position in dieser Darstellung nicht zu erkennen ist.



**Abb. 35:** Vergleich der mittels 'running mean' gefilterten integrierten Geschwindigkeit (rechts) mit der durch Differenzieren der Phasendaten erhaltenen (links).



**Abb. 36:** Berechnete Position (blau) verglichen mit der vom Radar ermittelten Position (rot). Mit auf die Geschwindigkeit angewandter running-mean Korrektur. Der Rote Balken am Anfang der Datenreihe gibt die Wellenlänge des Radars an um eventuelle Fehler beim phase unwrapping zu erkennen.

## 8 Zusammenfassung und Ausblick

Im Rahmen meiner Arbeit habe ich zwei Beschleunigungsmesssysteme entwickelt und in Experimenten getestet. Dazu habe ich die Theorie der Beschleunigungs-messung untersucht und dann sowohl an der Hardware als auch an der Software gearbeitet.

Bereits mit einem relativ günstigen System lassen sich interessante Ergebnisse erzielen. Das größte Problem dabei ist zunächst nicht die notwendige Genauigkeit des Sensors, als vielmehr die zu geringe Abtastrate. Mit einigen Änderungen an der Software lässt sich diese jedoch möglicherweise noch erhöhen.

In Experimenten in Waakirchen konnte ich mit dem Prototyp zeigen, dass sich die Radarschüssel durch die Druckwelle der Explosion bewegt. Diese Bewegung ist, ebenso wie durch Wind induzierte Bewegung, deutlich zu gering, als dass sie die Geschwindigkeitsmessung des Radars beeinflussen oder verfälschen könnte.

Das gleiche gilt für die Bewegung des Autoklavenaufbaus. Die gemessenen Geschwindigkeiten sind zwar deutlich größer als die der Schüssel, aber immer noch zu gering, als dass sie einen merklichen Einfluss auf die Geschwindigkeitsmessung haben.

Des Weiteren habe ich mich mit fortgeschrittenen Hilfsmitteln der Datenaquisition, wie Interrupts und FIFOs, auseinandergesetzt und damit ein zweites, schnelleres Schwingungsmesssystem gebaut.

Durch die Experimente am Geomatikum wird deutlich, dass es zwar prinzipiell möglich ist, die Bewegung von Messobjekt und Radar zu unterscheiden, dieses aber durch Gerätedrift und den fehleraddierenden Effekt des Integrierens sehr kompliziert ist. Hier wäre es interessant, sich noch ausführlicher mit der Datenbearbeitung zu befassen und nach einer passenden Lösung zu suchen. Ein möglicher Ansatzpunkt wäre eventuell die Verwendung eines Kalmanfilters. [vlg. Filieri and Melchiotti, 2012]

Komplett unbeachtet ist bisher auch die Tatsache, dass sich das Radar auch seitlich bewegt und kippt. Um diese Effekte zu berücksichtigen, müsste ein geometrisches Modell des Zielobjektes entworfen werden. Beide Schwingungsmesser nehmen alle 3 Komponenten der Beschleunigung im Raum auf.

## Literatur

Analog Devices. *1g to 5g Single Chip Accelerometer with Signal Conditioning.* Datasheet, 1996.

Arduino. Arduino wire library, 2012.

URL <http://www.arduino.cc/en/Reference/Wire>.

I.N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik.* Verlag Harri Deutsch, Frankfurt am Main, 2. überarbeitete und erweiterte Auflage, 1995.

Antonio Filieri und Rossella Melchiotti. *Position recovery from accelerometric sensors algorithms analysis and implementation issues.* Technical report, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, 2012.

Matthias Hort, R. Seyfried, und Malte Vöge. *Radar doppler velocimetry of volcanic eruptions: theoretical considerations and quantitative documentation of changes in eruptive behaviour at Stromboli volcano, italy.* *Geophysical Journal International*, 154(2):515–532, August 2003.

Yozo Kanda. Piezoresistance effect of silicon. *Sensors and Actuators A: Physical*, 28(2):83–91, July 1991.

Lasse Klingbeil. *Entwicklung eines modularen und skalierbaren Sensorsystems zur Erfassung von Position und Orientierung bewegter Objekte.* Dissertation, Universität Bonn, 2006.

Dieter. Meschede. *Gerthsen Physik.* Springer Berlin Heidelberg, 12. Auflage, September 2001.

Jonathan Miller. *The benefits of autocalibration.* manual. Diamond Systems, 2006.

Frank Neitzel, Thomas Schwanebeck und Willfried Schwarz. *Zur Genauigkeit von Schwingwegmessungen mit Hilfe von Beschleunigungs- und Geschwindigkeitssensoren.* *Allgemeine Vermessungsnachrichten*, 114(6):202–211, 2007.

NXP. *UM10204 I2C-bus specification and user manual.* manual, NXP, rev4, February 2012.

Lea Scharff, Alexander Gerst, Florian Ziemen, Matthias Hort, und Jeffrey B. Johnson. *The dynamics of explosions at Santiaguito volcano, Guatemala (v31e-0718).* In *Fall Meeting 2007.* American Geophysical Union, 2007.

Wolf Dieter Schmidt. *Sensorschaltungstechnik.* Vogel Business Media, 3. Auflage, August 2007.

Eberhard Sengpiel. *Symmetrische und unsymmetrische signalübertragung*. Tutorium UdK Berlin, March 2001. URL <http://www.sengpielaudio.com/SymmetrischeUndUnsymmetrischeSignaluebertragung.pdf>.

BMA180 Bosch Sensortec. *BMA180 Digital, triaxial acceleration sensor Data sheet*, Datasheet 2.1 revision, December 2009.

Claude E. Shannon. *Communication in the presence of noise*. *Proceedings of the IRE*, 37(1):10–21, January 1949.

S.J. Sherman, W.K. Tsang, T.A. Core, R.S. Payne, D.E. Quinn, K.H.-L. Chau, J.A. Farash, and S.K. Baum. A low cost monolithic accelerometer; product/technology update. In *Electron Devices Meeting, 1992. IEDM '92. Technical Digest., International*, pages 501–504, 1992.

Oliver Spieler, Ben Kennedy, Ulrich Kueppers, Donald B. Dingwell, Bettina Scheul, and Jacopo Taddeucci. The fragmentation threshold of pyroclastic rocks. *Earth and Planetary Science Letters*, 226(1-2):139–148, September 2004.

William Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall International, 6 edition, 2000.

Diamond Systems. *Universal Driver Manual v5.92*, 2008.

Ulrich Tietze and Christoph Schenk. *Halbleiter-Schaltungstechnik*. Springer, Berlin, 12 Auflage, 2002.

**Erklärung**

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbstständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen - benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium.

Ich bin damit einverstanden, dass die Bachelorarbeit veröffentlicht wird.

Hamburg, den 25. September 2012

Sebastian Beyer