Das Handy als Black Box

Inhalt

Kurzfassung

Hatten Sie schon mal einen Verkehrsunfall? Wenn ja, hatten Sie womöglich große Probleme zu beweisen, was wirklich passiert ist. Wenn Sie Pech haben, endet die Sache im Papierkrieg. Wenn Sie noch mehr Pech haben, müssen Sie für den Schaden selbst aufkommen, obwohl Sie keine Schuld trifft.
Mit meinem System, gehört dieses Problem endgültig der Vergangenheit an. Es zeichnet nämlich den Unfallhergang auf Video auf und speichert zusätzlich noch die Beschleunigung. Und das nicht mit unbezahlbar teuren Geräten, wie sie die Industrie anbietet, sondern mit Ihrem ganz normalen Handy und einem kleinen Zusatzmodul, das höchstens 25€ kostet.
Während der Fahrt filmt das Handy und erhält laufend Fahrdaten, die aufgezeichnet werden. Falls nun ein Unfall passiert, hat man u.a. ein Video, wo man sieht was *vor*, *während* und *nach* den Unfall abgelaufen ist. Mit den ebenfalls aufgezeichneten Beschleunigungswerten, kann man z.B. herausfinden, wie groß Ihre Reaktionszeit war oder wie hart die Kollision war. So können Sie z.B. Ihrer Versicherung beweisen, das der Schaden tatsächlich von dem Unfall stammt. Überhaupt lässt sich der Unfall, allein schon mit dem Video, in den allermeisten Fällen restlos aufklären.

1. Einleitung

Ein schwerer Verkehrsunfall bringt in den USA die Diskussion um den Unfalldatenschreiber wieder ins Rollen. Die Ursachen von Verkehrsunfällen sollen so leichter ausfindig gemacht werden. Der schwere Unfall, der die Gerichte in Sanxta Monica, Kalifornien, beschäftigte, hatte 10 Todesopfer gefordert und 63 Verletzte. Unfallverursacher war ein 86-jähriger Mann, der ungebremst mit seinem Wagen in einen Wochenmarkt hineinraste. Der Mann verweigerte vor Gericht jede Aussage.
Die Leiterin der US-Bundesanstalt für Transportsicherheit, Ellen Englemann Conners empfahl daraufhin für alle Autos eine Pflicht für Unfalldatenschreiber, auch

Abb. 1: Verkehrsunfall: Mit der Blackbox soll der Verursacher ermittelt werden

„Black Box“ genannt, einzuführen. Wie die Flugschreiber in Flugzeugen, mit denen Ermittler die Absturzursachen eines Fliegers im nach hinein feststellen können, soll die Blackbox im Auto mögliche Unfallursachen aufzeichnen.
2002 war Saab in Europa wegen solcher Datenspeicherung in die Schlagzeilen geraten: Die zentrale Motorsteuerung einiger Modelle zeichnet im Falle eines Unfalls Daten der letzten fünf Sekunden auf. Die schwedische Polizei hatte nach einem Unfall die Herausgabe der Daten gefordert. Das rief natürlich die Datenschützer aufs Parkett.
In Deutschland sind nur wenige Unfalldatenschreiber in Straßenfahrzeugen verbaut. 40.000 der "unbestechlichen Zeugen" hat Siemens VDO, das einzige Unternehmen, das die Geräte in Deutschland vertreibt, bisher verkauft.
[entnommen aus „US-Behörden fordern Blackbox-Pflicht im Auto“ in „Spiegel Online“ 06. August 2004]

2. Stand der Technik

Kennen Sie jemanden der eine Black Box im Auto hat? Wahrscheinlich nicht. Dabei gibt es schon längst die entsprechenden Geräte, die zur Aufklärung von Fahrunfällen beitragen würden (siehe Siemens VDO). Allerdings findet so eine Black Box bisher nur in Fahrzeugen von Polizei und Rettungskräften ihren Einsatz, was bei den hohen Anschaffungskosten von über 700€ nicht verwunderlich ist.
Von solchen modernen Methoden der Unfallaufklärung kann der Privatmann in der Regel nur träumen. Er kann nur hoffen, dass es couragierte Zeugen gab, die den Unfall genau beobachtet haben. Aber auch ein Sachverständiger, der versucht den Unfall mit Hilfe von Bremsspuren und Fotos zu rekonstruieren, kann oft den Schuldigen nicht eindeutig festmachen. Und so bekommen bei Gericht beide Parteien meist nur eine Teilschuld, was natürlich einen erheblichen Versicherungsschaden verursacht.
Ein weiter Nachteil der oben genannten Black Box ist, dass sie, anders als die subjektiven Zeugen, den Unfallhergang nicht sieht. Weder an die Black Box für Flugzeuge noch an die für Straßenfahrzeuge ist eine Videokamera angeschlossen. Mir fällt ehrlich gesagt kein vernünftiger Grund ein, warum das so ist. Mit einem Video des Unfalls, wüsste man doch sofort was passiert ist und hätte einen unschlagbaren Beweis in der Hand. Große Airlines kommen da mit dem Argument, dass eine Videoaufzeichnung zu teuer wäre. Wahrscheinlich trifft das auch auf Straßenfahrzeuge zu.
Ein weiterer Grund für diesen Missstand sind auch die Datenschützer. Sie argumentieren, dass ein Auto keine persönlichen Daten, wie ein Video der Autofahrt, aufzeichnen darf, das dann vom Gericht beschlagnahmt werden kann. Es lässt sich darüber streiten, ob jemand, der weiß, dass er schuldig ist, sich selbst belasten muss, wenn er eine Black Box eingebaut hat. Schließlich kann das Gericht herausfinden, ob man eine Black Box hat oder nicht. Ich denke, das ist mit der Hauptgrund, weshalb die breite Gesellschaft Unfalldatenschreiber ablehnt. Ich meine, wer will schon etwas kaufen, was einen eventuell hinter Gitter bringt?

3. Das Handy als Lösung

In meinem letzten „Jugend forscht“ Projekt ging es darum, Handys im Physikunterricht einzusetzen, um das Lernen interessanter und effektiver zu machen. Ich hatte nämlich herausgefunden, dass man mit dem Handy sehr viel mehr machen kann als nur telefonieren. So konnte ich zeigen, dass man mit meiner Handy-Software *Mobile Physics*, das Handy z.B. zum Messen von Spannungen, als kleines Oszilloskop, zur Versuchsauswertung usw. nutzen kann. Ich war überzeugt, dass die Möglichkeiten des Handys weit unterschätzt sind und man sie in den verschiedensten Anwendungsbereichen einsetzen könnte. Der Physikunterricht ist dabei nur ein Bereich von vielen, und ich wollte versuchen, noch mehr Verwendungs-möglichkeiten für Handys zu finden.
Als ich jetzt von diesem Artikel mit dem Autounfall hörte, kam ich auf die Idee, das Handy als Black Box einzusetzen. Für diese Aufgabe ist es nämlich wie geschaffen. Zum einen ist es in der Gesellschaft weit verbreitet und mittlerweile nicht mehr wegzudenken. Zum anderen sind die allermeisten Geräte mit einer Videokamera ausgestattet und können damit den Unfall filmen. Das Ganze habe ich mir so vorgestellt: Bevor der Anwender mit dem Auto losfährt, startet er meine Handy-Software und steckt sein Handy in die dafür vorgesehene Halterung. Die ist so ausgerichtet, dass die Handykamera nach draußen „sehen“ kann und trotzdem noch das Armaturenbrett im Blick hat. Während der Fahrt filmt das Handy und erhält laufend Fahrdaten, die aufgezeichnet werden. Welche das sind und wie das geht, verrate ich später. Sollte nun ein Unfall passieren, merkt das Handy das und stoppt die Aufzeichnung. Der Anwender kann nun die Daten auf seinen PC übertragen und dort, softwaregestützt, auswerten lassen. Auf diese Weise kann er nun nachweisen, was genau bei dem Unfall abgelaufen ist.
Ich würde sagen, damit liegen die Vorteile des Handys, gegenüber der herkömmlichen Black Box, klar auf der Hand:
Erstens hat fast jeder ein Handy und muss deshalb nicht über 700€ ausgeben, wenn er will. dass sein Unfall aufgezeichnet wird.
Zweitens hat der Anwender ein Video vom Unfall. Und das reicht meistens schon aus, um den Schuldigen eindeutig auszumachen. Außerdem fordern Unfallforscher schon seit langem die Videoaufzeichnung für Unfallschreiber.
Der dritte Punkt ist sehr wichtig und hat die Leute bisher abgeschreckt eine Black Box für ihr Auto anzuschaffen: der Datenschutz. Wenn man aber ein Handy als Black Box (miss-)braucht, kann man sich nicht selbst belasten. Schließlich sind alle aufgezeichneten Daten nur auf dem persönlichen Handy gespeichert und können zur Not auch vom Anwender gelöscht werden. Weil das Handy nicht fest mit dem Fahrzeug verbunden ist, kann außerdem niemand nachweisen, ob der Unfall irgendwie aufgezeichnet wurde, und man kann deshalb auch nicht gezwungen werden die Daten preiszugeben. Bei der herkömmlichen Black Box ist das anders. Hier kann man die ganze Elektronik, die dazu gehört, nicht „mal eben“ ausbauen.

Die Idee habe ich mir übrigens als Patent schützen lassen, weil ich glaube, dass die Erfindung Marktpotenzial haben könnte.
Aktenzeichen beim Deutschen Patent- und Markenamt: 10 2006 000 670.4.

4. Umsetzung

4.1 Planung

Zuerst muss ich mir einen Überblick verschaffen. Was muss mein System können und wie soll es funktionieren? Also der Reihe nach. Fest steht, dass das Handy mindestens ein Video des Unfalls aufzeichnen muss. So weit so gut. Doch woher weiß es, ob ein Unfall passiert ist? Theoretisch müsste ich die gesamte Fahrt filmen. Das geht natürlich nicht, der Speicher im Handy ist viel zu klein um eine mehrstündige Autofahrt aufzunehmen. Also habe ich überlegt, was eigentlich bei einem Unfall passiert. Aha, die Airbags zünden! Und woher wissen die, wann sie aufgehen sollen? Ganz einfach: im Airbag sitzt ein Beschleunigungssensor. Steigt die Beschleunigung des Autos über ein bestimmtes Limit, weil man kollidiert, also ein Unfall passiert, dann öffnet sich der Airbag. Also halte ich fest: das Handy muss irgendwie an die Beschleunigung herankommen.
Ich brauche also auch so einen Sensor, der aber empfindlicher sein soll, denn ich will auch leichte Auffahrunfälle aufzeichnen, wo die Airbags normalerweise noch nicht zünden. Aber die Beschleunigungswerte sind nicht nur wichtig, um den Unfall zu markieren, sondern sind auch nützlich um das Unfallgeschehen zu rekonstruieren. So kann man z.B. anhand der Beschleunigung bestimmen, wann der Fahrer gebremst hat, also wie hoch seine Reaktionszeit war. Natürlich lassen sich noch weitere Parameter aus der Beschleunigung ablesen, wie die ungefähre Geschwindigkeit, Anzahl der Überschläge und vieles mehr. Deswegen beschloss ich, die Beschleunigung ebenfalls aufzuzeichnen.
Übrigens hatte es einen Sinn, dass ich sagte, man solle die Handykamera auch auf das Armaturenbrett richten. So sieht man nämlich auf dem Video z.B. nämlich den Zustand der Lichter oder das Tachometer. Das sind alles Dinge, die eine herkömmliche Black Box auch aufzeichnet. Bei meinem System hat man dann noch den Vorteil, dass man den Zusammenhang, wegen dem Video, besser erkennt.
Bezüglich der Aufzeichnung während des Unfalls habe ich mir noch was besonderes einfallen lassen: Es wäre doch sehr hilfreich, wenn man nicht nur Informationen, über das hätte, was *nach* dem Unfall passiert ist, sondern auch noch wüsste was *vorher* abgelaufen ist, wie es also zu dem Unfall kommen konnte. Wie ich das umsetze dazu später mehr.
Jetzt bliebe noch zu klären, was man als nächstes macht, wenn man einen Unfall hatte, den das eigene Handy aufgezeichnet hat. Da es zu mühsam wäre, den Unfall auf dem Handy selbst zu rekonstruieren, habe ich mir gedacht, ich schreibe ein PC-Programm, das dabei hilft. Der Anwender muss also die Daten vom Handy nach dem Unfall auf einen PC übertragen.

Das ist also das Ziel, das ich mir gesetzt habe. Natürlich weiß ich, dass es nicht einfach wird, aus einer Handvoll Komponenten, die eigentlich gar nichts mit einander zu tun haben, etwas Funktionierendes zusammen zu basteln, aber ich will es wenigstens versuchen.

4.2 Der Beschleunigungssensor

Den Beschleunigungssensor hatte ich noch von einem früheren Projekt. Es handelt sich um den SMD-Chip ADXL202. Er kostet ca. 12€. Gemäß dem Datenblatt baute ich die kleine äußere Beschaltung auf. Jetzt ist die Frage, wie die Daten, die der Chip liefert, ins Handy übertragen werden. Leider hat es keinen einfachen Anschluss für solche Zwecke. Warum auch?

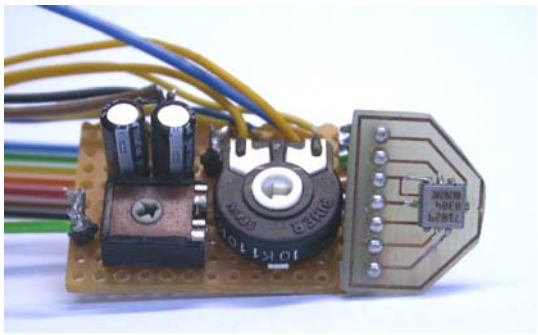

Abb. 4.2.1: Der Sensor + Beschaltung

Nun gibt es mehrere Möglichkeiten:

1. Bluetooth:
   Mit dieser Technik, können zwei Bluetoothgeräte beliebige Daten über einige Meter Entfernung, austauschen. Aber nicht alle Handys haben Bluetooth eingebaut und ein Bluetooth-Sender kostet über 100€.
2. Infrarot:
   Eine prima Möglichkeit um Daten einfach zu übertragen. Leider haben nur die wenigsten Geräte eine Infrarotdiode.
3. Akustische Datenübermittlung:
   Die Daten werden in Töne umgewandelt und vom Handymikrofon aufgenommen.
   Vorteil: Geht mit jedem Handy. Nachteil: Schwer umzusetzen und fehleranfällig.

Ich entschied mich für die dritte Methode. Wie gern hätte ich es mit Infrarot gemacht, aber leider hat mein Handy keine Infrarotdiode und neue Modelle sowieso nicht mehr. Jetzt geht es also darum, die Beschleunigungswerte in Töne umzuwandeln. Dazu habe ich mir erst mal das Signal, das der Sensor jeweils für die x- und y-Achse ausgibt, angesehen:

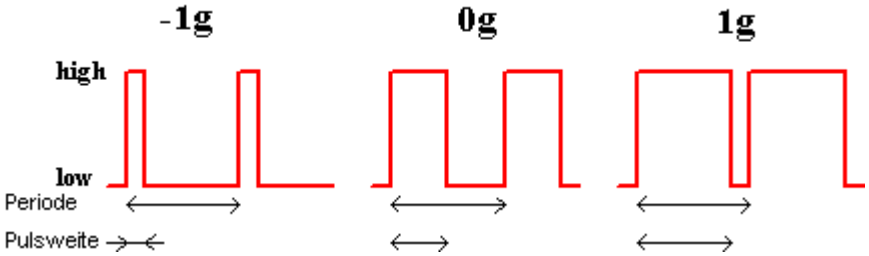

Abb. 4.2.2: die Signale bei verschiedenen Beschleunigungen.

Wie man sieht, ist das Signal pulsweitenmoduliert. Dabei ist die Periode immer gleich lang (ca. 2ms), während die Pulsweite abhängig von der aktuellen Beschleunigung ist (ca. 600µs – 1800µs). Die genauen Zeiten kann man mit der äußeren Beschaltung einstellen. Nun kann ich dieses Signal aber nicht einfach an den Lautsprecher geben. Es würde stark verzerrt und damit verfälscht werden. Schließlich kam mir die Idee, dieses Signal von einem Mikroprozessor auswerten zu lassen. Dann hätte ich nämlich einen digitalen Beschleunigungswert, den dann wiederum der Mikroprozessor in Töne umwandelt würde.
Beim Prozessor entschied ich mich für den PIC 16F84A, weil ich schon aus einem früheren Projekt mit diesem Modell zu tun hatte.

Dieser PIC kostet ungefähr 6€ und besitzt insgesamt 13 digitale Ports, die entweder als Eingang oder Ausgang benutzt werden können. An zwei dieser Ports habe ich dann den Sensor angeschlossen. Ein Pin für die x- und ein Pin für die y-Achse. Und dann konnte ich auch schon anfangen zu programmieren.
PIC-Programme werden üblicherweise in Maschinensprache, sprich Assembler, geschrieben. Wenn man sein Programm testen möchte, kompiliert (übersetzt) man es auf dem PC und sendet es dann über ein serielles Kabel an den PIC. Leider ist er sehr simpel gestrickt und man braucht als Programmierer eine Menge Kreativität um aus den sehr wenigen Befehlen ein vernünftiges Programm zu entwickeln. Nichtsdestotrotz hatte ich nach einiger Zeit eine Routine geschrieben, die eine Achse ausliest. Sie funktioniert nach folgendem Prinzip:

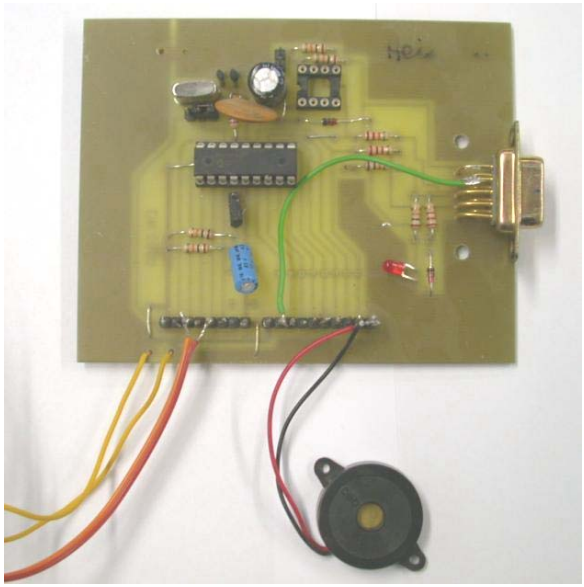

Abb. 4.2.3: Der PIC mit Programmierschaltung. Man sieht noch den Piezo und den Anschluss zum Sensor.

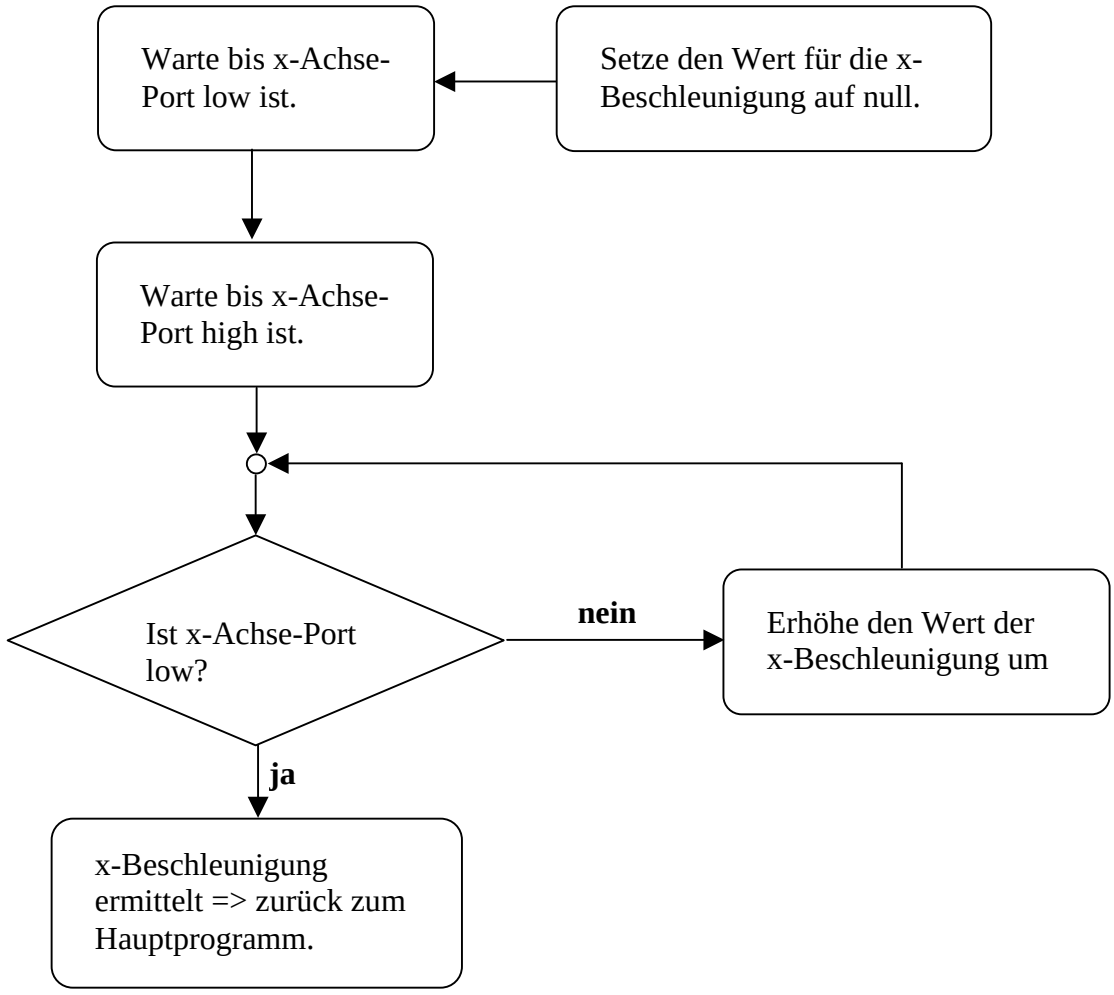

Abb. 4.2.4: Flussdiagramm

Aus diesem Flussdiagramm erkennt man, dass der Wert für die x-Beschleunigung umso größer ist, je länger der Port für die x-Achse high ist, weil dann die Schleife öfter durchlaufen wird. Genauso funktioniert es mit der y-Achse. Diese beiden Routinen werden in einer Endlosschleife vom Hauptprogramm hintereinander ausgeführt.

Nun hielt ich es nicht für klug, jetzt direkt mit der Handy-Programmierung weiterzumachen. Was, wenn der Sensor total ungenau ist oder viel zu langsam arbeitet? Oder wenn sich herausstellt, dass die Sache mit dem Sensor ein ganz falscher Ansatz war? Dann müsste ich wieder ganz von vorne anfangen und mir irgendetwas anderes einfallen lassen. Ob die Sache mit dem Sensor überhaupt funktionieren kann, wollte ich *jetzt* wissen und nicht wenn es schon zu spät ist. Deshalb beschloss ich, die Werte, die der Sensor ausspuckt, irgendwie sichtbar zu machen. Am besten in einem Diagramm. So etwas kann man eigentlich nur am PC machen und ich fragte mich wie ich die x- ,y-Werte vom PIC am besten dorthin übertrage. Im Grunde kam nur die serielle Schnittstelle in Frage. Statt nun ein Programm vom PC in den PIC zu laden, drehte ich den Datenstrom einfach um, und übertrug die Messwerte von PIC an den PC. Dabei nutzte ich die Hilfe eines Freeware Datenlogger-Programms aus dem Internet, um die empfangenen Datenmengen zu speichern.
Um die Daten auszuwerten, entwickelte ich ein Programm in C++, das erstens die Messwerte in einem Diagramm veranschaulicht und zweitens diese Beschleunigungswerte in Geschwindigkeits- und Ortsangaben umrechnet (die ebenfalls im Diagramm angezeigt werden). Theoretisch ist diese Umrechnung ganz einfach:

$$v(t) = v(t-1) + a(t) \cdot \Delta t$$

$$s(t) = s(t-1) + v(t) \cdot \Delta t$$

Wenn das wirklich so in der Praxis funktioniert, hätte ich zu jedem Zeitpunkt t die Geschwindigkeit und den Standort des Autos! Daraus könnte man dann eine Animation des Unfalls entwickeln!
Wie man in den Formeln sieht, muss man annehmen, dass zum Zeitpunkt **t** = 0, **v** und **s** null sind. Bei **s** ist das kein Problem, wohl aber bei **v**. Schließlich ist man am Anfang der Datenaufzeichnung (**t** = 0) noch am Fahren. Allerdings wohl nicht mehr am Ende, wenn man einen Unfall gebaut hat. Wenn man jetzt also **t** = 0 an das Ende setzt, und die Zeit rückwärts laufen lässt, hat man das Problem geschickt gelöst.
So weit die Theorie. Kommen wir nun zum Praxistest. Ich möchte hier nicht im Detail auf den Quelltext meines Analyse-Programms eingehen. Das würde definitiv den Rahmen sprengen. Genauso ist es mit dem PIC- und dem Handyprogramm. Stattdessen zeige die wichtigen Algorithmen.

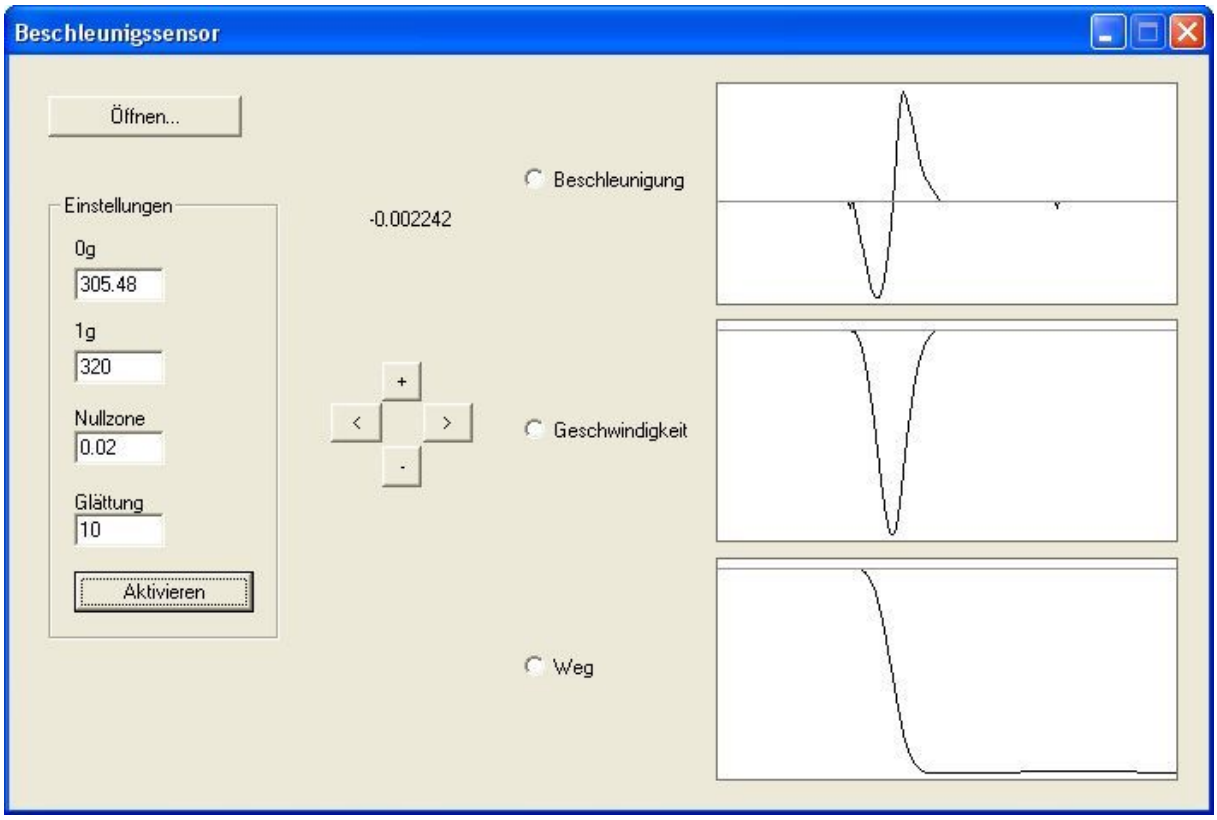

Abb. 4.2.5: Das **a-, v-** und **s**-Diagramm mit einigen Einstellmöglichkeiten.

Da der Sensor die Daten ohne Einheit sendet, muss man definieren, welcher Wert 0g und welcher Wert 1g entspricht. Außerdem habe ich noch eine Nullzone definiert. Sie sorgt dafür, dass Werte nahe bei null, auf null gesetzt werden. Zudem habe ich festgestellt, dass die Ergebnisse besser werden, wenn ich das Signal glätte.

Obwohl die Diagramme sehr plausibel aussehen, war ich doch sehr enttäuscht. Denn nur durch Herumprobieren bei den Einstellungen auf der linken Seite, konnte ich so exakte Ergebnisse erzielen. Offenbar genügt eine winzig kleine Ungenauigkeit beim Sensor, die, nach einiger Zeit, große Fehler verursacht. Trotzdem muss man sagen, dass der Sensor sehr genau arbeitet. Später haben mir Fachleute bestätigt, dass die Berechnung von Geschwindigkeit und Strecke nur anhand von Beschleunigungswerten, zumindest über längere Zeit, sehr schwierig, wenn nicht gar unmöglich, ist.
Nichtsdestotrotz hielt ich an diesem Ansatz fest, weil dieser Test mir gezeigt hat, dass der PIC eigentlich nichts für die Berechnungsfehler kann. Diese Fehler treten eben bei jedem Beschleunigungssensor auf. Außerdem kann man nie wissen, wie sich das System in der Praxis verhält, wenn alles fertig ist. Zudem sind in den reinen Beschleunigungswerten, wie gesagt, ja auch eine Menge Informationen enthalten.

4.3 Die akustische Kopplung

Irgendwie musste ich die Messwerte vom PIC in das Handy übermitteln. Aus genannten Gründen entschied ich mich, dies über akustischem Wege zu machen. Klären wir nun erst mal die Theorie, bevor wir zur Praxis kommen. Also der PIC misst laufend die Beschleunigung der beiden Achsen. Jetzt kommt noch eine Routine hinzu, die die Messdaten in Töne umwandelt und an den Lautsprecher gibt. Der Lautsprecher erzeugt Schallwellen, die das Handy aufnimmt und alle vier Sekunden in die Messwerte zurückverwandelt. Nun kommen auch schon die ersten Fragen: Woher weiß ich, welche Frequenzen ich benutzen soll und wie lange muss eine Informationseinheit mindestens dauern?
Leider bauen die Jungs, die die Handys herstellen, nur sehr schlechte Mikrofone ein. Sie taugen höchstens, um Sprache zu übertragen. Ich aber wollte so viele Messdaten wie möglich pro Sekunde senden. Der Kampf um jedes Bit hatte begonnen.
Zuerst wollte ich herausfinden welchen Frequenzbereich das Handymikrofon einigermaßen verzerrungsfrei aufnehmen kann. Dazu erzeugt ich mit Goldwave einen Sinus-Sweep von 0Hz–10kHz. Außerdem entwickelte ich ein Handy-Programm das vier Sekunden Sound aufnimmt und abspeichert. Die so entstandene Wave-Datei konnte ich dann per Bluetooth an meinen PC übertragen und mir wieder in Goldwave anschauen. Was ich da sah sollte zu einem großen Problem werden. Denn nur die Frequenzen von etwa 300Hz-1800Hz waren einigermaßen verzerrungsfrei aufgenommen worden. Diese sehr geringe Bandbreite musste ich so effektiv wie möglich nutzen. Dabei hat das Herumprobieren, welche Parameter am besten sind, viel Zeit und Nerven gekostet.
Ein Beispiel: Ich erzeuge am PC zwei verschiedene Frequenzen, einmal 500Hz und einmal 1000Hz. 500Hz steht für eine binäre null, 1000Hz für eine binäre eins. Und ich lege weiterhin fest, dass ein Bit exakt 100 Sample lang ertönen muss. Bei 8000 Sample pro Sekunde, die das Mikrofon schafft, entspricht das einer Zeit von 1,25ms. Zwischen jedem Messwert wurde ca. 100 Sample Pause gelassen. Das aufgenommene Signal sah dann beispielsweise so aus:

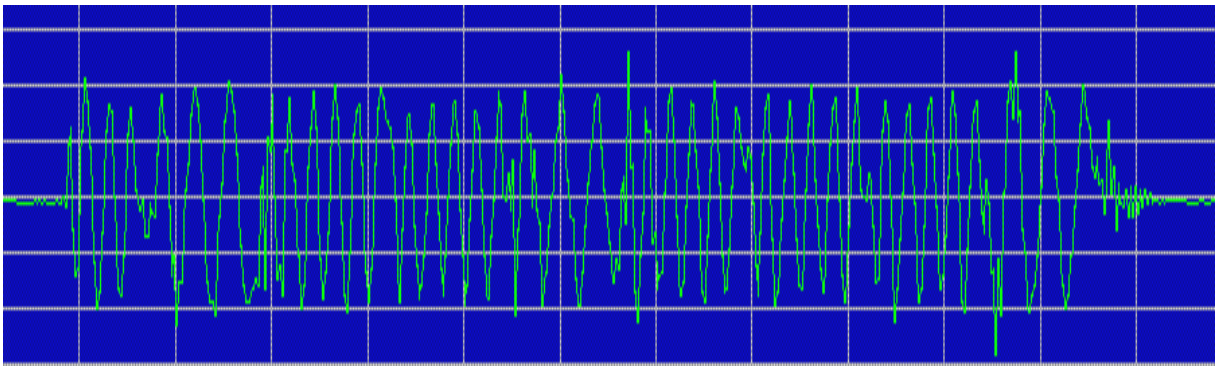

Abb. 4.3.1: „Einsen“ und „Nullen“ aufgenommen vom Handy-Mikrofon.

Man erkennt deutlich die beiden verschiedenen Frequenzen. Es fällt auf, dass bei einem Übergang von eins auf null, und umgekehrt, Verzerrungen auftreten. Das hatte zur Folge, dass ich von den 100 Sample, nur ungefähr 70-80 nutzen konnte, um diesen Verzerrungen aus dem Weg zu gehen. Um aus diesem Audiosignal wieder Zahlen zu bekommen, entwickelte ich nun ein Programm auf dem PC, das die Frequenz eines bestimmten Audioabschnittes (Bit) ermittelt. Normalerweise müsste man sich jetzt mit Fourieranalyse quälen, aber ich habe mir ein viel schnelleres und einfacheres Verfahren einfallen lassen, welches sehr effektiv ist. Ich messe nämlich einfach den Abstand zwischen dem ersten Wellenberg/tal und dem letzten Wellenberg/tal und teile das Ergebnis dann durch die Anzahl der Wellenberge und Wellentäler. Auf diese Weise erhalte ich den mittleren Abstand zwischen einem Wellenberg

und einem Wellental in einem bestimmten Audioabschnitt. Dabei gilt: je kleiner dieser Abstand, desto größer die Frequenz. Und so funktioniert der Zählalgorithmus:

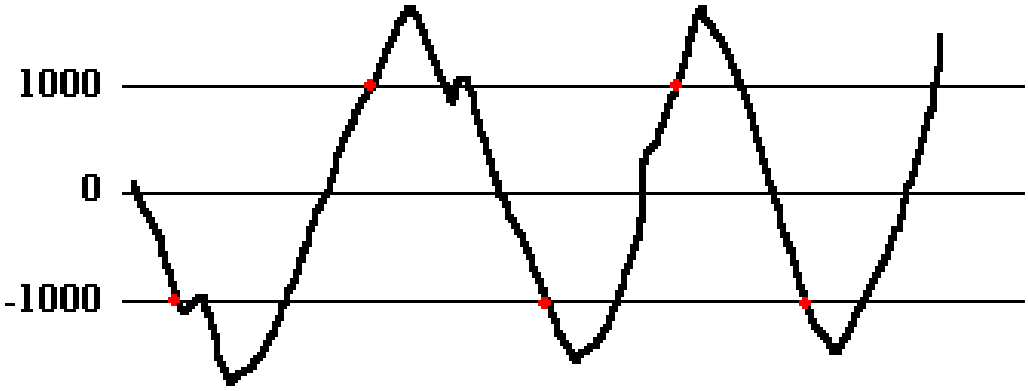

Abb. 4.3.2: Beispielhaftes Signal. Die roten Punkte markieren ein Wellenberg/tal.

Wesentlich ist die Variable **m**. **m** ist am Anfang null. Nun wird Sample für Sample nacheinander durchgegangen. Falls ein Sample kleiner ist als –1000, wird gefragt ob **m** = 1 ist. Wenn ja erhöht sich die Anzahl der Wellenberge/täler. In jedem Fall wird **m** = -1 gesetzt. Falls ein Sample größer ist als 1000, wird gefragt ob **m** = -1 ist. Wenn ja erhöht sich die Anzahl der Wellenberge/täler. In jedem Fall wird **m** = 1 gesetzt. Mit dieser Methode zähle ich jeden Wellenberg/jedes Wellental nur *einmal*. Außerdem darf das Signal ruhig etwas „zittern".
Als das Programm fertig war, habe ich festgestellt, dass sich bei nur zwei verschiedenen Frequenzen oder einer Bitlänge von 100 Sample, nur etwa 6 Messwerte pro Sekunde mit einem Fehleranteil von etwa 20%, übertragen lassen. Total inakzeptabel. Und überhaupt, was ich brauchte war eine Fehleranteil von absolut 0%. Ansonsten könnte das Handy, so mir nichts dir nichts, einen Unfall melden. Es gab eigentlich nur eine Lösung. Ich muss jeden Wert zweimal senden. Und nur wenn beide Werte gleich sind, wird er akzeptiert. Also nur noch 3 Messwerte pro Sekunde. Und was jetzt? Ich kam auf die Idee nur die ersten 10 Bits eines Messwerts, statt die vollen 16 Bit, zu übertragen. Die letzten 6 Bits waren nämlich sowieso immer null. Nun begann eine lange Suche nach den besten Parametern. Ich versuchte alle möglichen Kombinationen. Schließlich stand fest:
32 verschiedene Frequenzen
Bitlänge: 150 Sample
Abstand zwischen Frequenzen: 45Hz
Niedrigste Frequenz: 300Hz
Höchste Frequenz: 1695Hz

Mit diesen Parametern müsste ich 12-15 Werte pro Sekunde schaffen. Einen Lautsprecher, der 300Hz sauber abspielt, musste ich erst bestellen und warte bis jetzt noch auf ihn. In der Zwischenzeit benutzte ich einen kleinen Piezo, der aber nur für 900Hz – 1100Hz zu gebrauchen war. Es blieb mir nichts anderes übrig als nur zwei Frequenzen zu nehmen und 4 Werte pro Sekunde zu empfangen. Aber das ist ja zum testen egal.

Übrigens, falls Sie sich fragen, wieso das Signal mehr nach einem Dreiecks- statt einem Sinussignal aussieht. Das liegt am PIC. Es wäre völlig undenkbar eine Sinusfunktion für so

einen Chip zu programmieren. In einer Hochsprache wie C das, abgesehen von der Schnelligkeit, aber jeder Programmierer wird sich für Assembler entscheiden, weil der Code erheblich schneller ist, da man genau weiß, was man braucht und was man weglassen kann. Jedoch braucht man für diese Art der Programmentwicklung auch sehr viel länger als in einer Hochsprache. Es gibt insgesamt nur 33 verschiedene Befehle für den 16F84A, doch ist selbst ein Vertippen gefährlich, da sich viele Befehle nicht sonderlich voneinander unterscheiden. Dazu muss man sich auch die Reihenfolge der Befehle gut überlegen, weil eine solche Kleinigkeit schon eine stundenlange Fehlersuche nach sich ziehen kann. Ich würde sogar sagen, dass Programmieren allgemein zu 80% Fehlerbeseitigung ist. Auch bei Profis.
Und so konnte ich „nur“ ein Rechtecksignal ausgeben, das dann durch die Trägheit der Membrane wie ein Dreieckssignal aussieht. Am schwierigsten aber war das Timing, dass jedes Bit exakt 100 Sample lang ist. Das hat sehr viel Denkarbeit gekostet, weil es nur mit kompliziertester Interrupt- und Timertechnik realisiert werden konnte. Ich musste dabei wirklich das Letzte aus diesem Chip „herauskitzeln“.

4.4 Das Handy

Ein Handy ist eine wahnsinnig komplizierte Maschine, wenn man sie programmieren möchte. Dadurch, dass viel weniger Ressourcen, sprich Speicher, als beim PC zur Verfügung stehen, muss man beim Programmieren ganz anders *denken.* Das C++ beim Handy hat fast nichts mehr mit dem „normalen“ C++ für PCs zu tun. Irrsinnig viele verschiedene Klassen mit Funktionen, und die gesamte Dokumentation natürlich in Englisch. Da gibt es keine Anleitung, wie man Symbian C++ (Symbian ist das „Windows“ für Handys) lernt. Man muss sich aus unzähligen Papers „Informationshäppchen“ zusammensuchen. Ich habe 2 Monate gebraucht um zu verstehen wie ein simples „Hello World“-Programm funktioniert.

Zuerst dachte ich, es wäre so ähnlich wie das Programmieren von Windows-Programmen, wo ich langjährliche Erfahrung habe. Dort gibt es eine *WinMain*-Funktion, wo das Programm startet und sonst kann der Programmierer selbst entscheiden, ob das Programm ein paar Zeilen oder tausend Zeilen groß wird. Bei einem *Series60*-Programm (Zusatz des *Symbian*-BS) braucht man für das kleinste „Hello World“ Programm mindestens sechs Dateien und ca. 300 Zeilen Code.
Zurück zu den Anfängen: Als erstes hörte ich, man würde Mobiltelefone mit Java programmieren. Java wurde 1994 von Sun Microsystems entwickelt, um Toaster und Kühlschränke über das Internet zu steuern. Sprich, man entwickelte eine Sprache, dessen Programme auf allen Maschinen laufen sollte, (PCs, MACs, Kaffeemaschinen, ...) so auch auf Handys. Das gute an Java war, dass es relativ einfach war. Leider stellte ich dann jedoch fest, dass man damit nur oberflächlich programmieren konnte und nicht *lowlevel*. D.h., ich konnte keine Schnittstellen wie Mikrofon oder Kamera ansteuern, was ja unbedingt nötig war.
Zum selben Zeitpunkt entdeckte ich *Symbian C++*. Damit konnte man das Handy in C++ programmieren und zwar *lowlevel*. Dazu lud ich mir das ganze Paket aus Bibliotheken, Tools und Includes herunter und versuchte ein simples Beispielspielprogramm zu kompilieren (in die „Handy-Sprache“ übersetzen). Zusätzlich brauchte ich noch andere Entwicklungs–programme, wie Active Perl, Visual C++ 6. Die ganze Misere dabei war das Zusammenspiel der einzelnen Komponenten. Aus den engl. Foren (sing. Forum) und Dokumentationen erfuhr ich in etwa wie die Programme kompiliert wurden. Als alles bereit war, ging nichts. Das lag zum größten Teil daran, dass die Handy-Programmierung in der Form noch sehr neu war und es weltweit nicht sehr viele Leute gab, die richtig Erfahrung hatten. Da hatte man die o.g. Komponenten notdürftig zusammengeflickt und wurde als Anfänger von Tausenden Fehlern überhäuft. Doch zum Glück war ich nicht alleine. Mindestens 50% der Beiträge im offiziellen

Nokia-Forum hatten mit Problemen bei der Kompilation zu tun. Auf diese Weise konnte ich viele Probleme lösen und kam dem Ziel das Beispielprogramm auf meinem Handy auszuführen immer wieder einen Schritt näher.
Ein Beispiel: Ich konnte ein Programm kompilieren, aufs Handy übertragen und installieren. Ich dachte jetzt würde es gehen, doch es erschien kein Programmsymbol im Menü. Letztendlich lag es an einem veralteten Pfad in der Installationsdatei. So vergingen zwei Wochen bis ich das erste „Hello World“ auf die Beine stellte.
Ich gebe mal eine kleine Einführung in der Welt der Handy-Programmierung:
Alle nicht hoffnungslos veralteten Handys haben Symbian als Betriebssystem. Symbian ist ein 32-Bit Multitasking BS (Betriebssystem), welches speziell für Mobiltelefone entwickelt worden und deshalb besonderen Wert auf geringen Stromverbrauch und Stabilität legt. Series 60 ist eine Erweiterung für Symbian und ist für die grafische Umgebung verantwortlich und ist ebenfalls sehr verbreitet. Mein Nokia 6630 hat Symbian 8.0a und Series60 2.0, die schon nicht mehr die aktuellsten Versionen sind.

Ein Handyprogramm besteht aus vier Komponenten:

1. *Application* ist abgeleitet von der Klasse (Funktionsbibliothek) *CAknApplication*. Es ist die Komponente, die beim Programmstart als erstes ausgeführt wird und ist dafür verantwortlich die nächste Komponente zu initialisieren.
2. *Document* ist abgeleitet von *CAknDocument*. Es kann für Textverarbeitung genutzt werden, wovon aber nur die wenigsten Programme Gebrauch machen. Fast immer ist es eine leere Klasse, die nur *Application UI* aufruft.
3. *Application UI* ist abgeleitet von *CAknAppUI*. Diese Komponente macht die Hauptarbeit. Es steuert die eingehenden Ereignisse, ruft BS-Funktionen auf und übernimmt eigentlich die ganze Datenverarbeitung. Zusätzlich startet es noch die letzte Komponente.
4. *View* : Das ist, was der Benutzer auf dem Display sieht. Es kann für Daten ausgeben, sowie Daten vom Benutzer einlesen.

Das war der wesentliche Programmaufbau. Wie man sieht, sind nur die letzten zwei Komponenten wichtig. Dort steht das *eigentliche* Programm. Der Rest ist quasi nur die *Hülle* und ist im Prinzip immer gleich. Anders als beim PC steuert beim Handy das BS das Programm und nicht umgekehrt. Damit meine ich, dass das BS die exe-Datei für jedes Programm erstellt, und *nicht* der Programmierer. Der Programmierer stellt eher seinen Code der exe-Datei zur *Verfügung*, die dann dafür sorgt, dass die Komponenten richtig aufgerufen werden.

Nach einigem Training kam ich so langsam dahinter wie das alles funktioniert. Es war zwar immer noch teilweise unverständlich, aber ich entwickelte langsam ein Gefühl für Symbian C++. Eines Tages war es soweit: Mein Handy konnte Geräusche aufnehmen und abspeichern. Damit habe ich dann die Versuche für die akustische Kopplung gemacht. Der nächste Schritt war jetzt klar. Ich wollte, dass die Beschleunigungswerte direkt im Handy berechnet werden. Beim Umschreiben des PC Programms, das diese Aufgabe bisher erfüllt hatte, in Symbian C++ machte ich zwar enorm viele Fehler, aber nach zwei Tagen hat es dann geklappt. Wahnsinn! Durch diesen piepsenden Piezolautsprecher wurden tatsächlich Daten übertragen! Ein tolles Gefühl, wenn etwas funktioniert, was auf diese Weise noch nie jemand zuvor gemacht hat. Die einzelnen Komponenten waren perfekt aufeinander abgestimmt.

Ich hatte am Anfang gesagt, dass es doch hilfreich wäre, wenn man nicht nur Informationen, über das hätte, was *nach* dem Unfall passiert ist, sondern auch noch wüsste was *vorher* abgelaufen ist, wie es also zu dem Unfall kommen konnte. Das mache ich mit einem so genannten Ringspeicher:
Jeder Datenspeicher ist so groß, dass er die Messwerte aus vier Sekunden Audio-aufnahme speichern kann. Wurde der Datenspeicher beschrieben, rücken alle Datenspeicher eins nach rechts. Passiert ein Unfall, wird ein ganz anderer Ringspeicher beschrieben. So hat man noch die Daten von vor dem Unfall. Zu den Daten gehört das Video und die Beschleunigung.

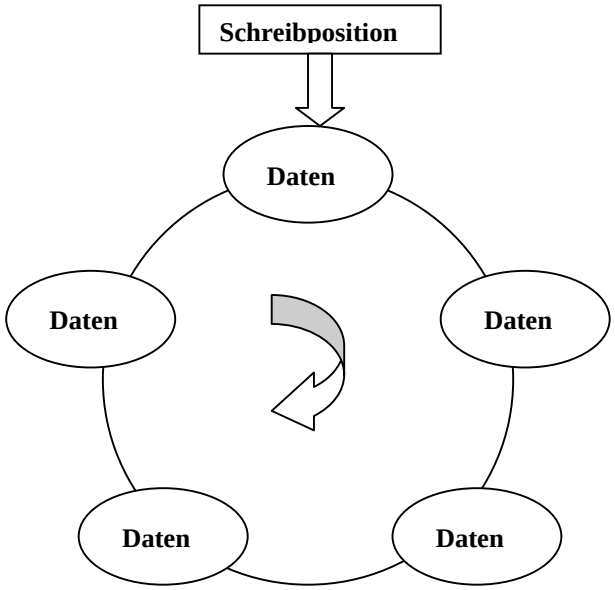

Abb. 4.4.1: Wenn ein Datenspeicher fertig beschreiben ist, rücken alle Datenspeicher eins nach rechts.

5. Der Knopf

Folgende Situation: Sie fahren in einem dichten Wohngebiet vor sich hin und plötzlich läuft ein Kind vor Ihr Auto. Sie machen eine Vollbremsung - ein Unfall kann gerade noch vermieden werden. Natürlich springt die Handy -Black Box nicht an, denn Sie hatten ja keine Kollision. Trotzdem wäre es wichtig zu wissen, was passiert ist. In diesem Fall drückt man einfach eine Taste auf dem Handy, das die selben Auswirkungen hat, wie ein Unfall. Man hat dann also ein Video und die Beschleunigungsdaten von dem Vorfall, der kein Unfall war.
Natürlich kann man sich noch viele weitere Situationen vorstellen, wo es wichtig ist die letzten Sekunden als Video zur Hand zu haben.

6. Ausblick

Zwar kann meine Black Box noch keine Videos aufnehmen, trotzdem habe ich das schwierigste jetzt hinter mir und bin sehr zuversichtlich, dass meine Erfindung funktionieren wird. Wenn ich das geschafft habe, werde ich noch ein PC-Programm entwickeln, das hilft die Handydaten auszuwerten.
Neulich habe ich gelesen, dass die Industrie plant, den Beschleunigungssensor ADXL330 in Handys einzubauen, um neuartige Bedienungstechniken zu ermöglichen. Das bedeutet, dass meine Idee mit hoher Wahrscheinlichkeit umgesetzt wird.
Natürlich hoffe ich, dass mein System in der Gesellschaft Anerkennung findet. Eigentlich spricht nichts dagegen, weil man für sehr wenig Geld riesige Vorteile bekommt.