Online GPS für Permafrost Monitoring

Semesterarbeit

Josua Hunziker

22. Dezember 2009

Betreut durch Dr. Jan Beutel und Matthias Keller

Prof. Dr. Lothar Thiele

Institut für Technische Informatik und Kommunikationsnetze
Departement für Informationstechnologie und Elektrotechnik

ETH Zürich

Danksagung

Zuerst möchte ich Prof. Dr. Lothar Thiele dafür danken dass ich diese Arbeit im Rahmen seiner Forschungsgruppe schreiben durfte. Parallel zum Besuch seiner vorzüglichen Vorlesung "Hardware / Software Codesign" konnte ich einige der darin enthaltenen Prinzipien und Methoden hier direkt anwenden und vertiefen.

Ich danke Dr. Jan Beutel und Matthias Keller für ihre engagierte Betreuung dieser Arbeit, die vielen Anregungen, Einwände und Diskussionen. Weiter danke ich Roman Lim, Christoph Walser und Mustafa Yücel für all ihre Hilfe und Unterstützung. Ihr alle leistet in PermaSense tolle Arbeit und ich bin stolz meinen Teil zu diesem spannenden Projekt beitragen zu dürfen. Ein ganz besonderer Dank geht an Dr. Philippe Limpach vom Institut für Geodäsie und Photogrammetrie der ETH Zürich für seine grossartige Unterstützung im Bereich GPS und den damit verbundenen Herausforderungen.

Nicht zuletzt danke ich meinem privaten Umfeld: meiner Freundin Romina für ihre Geduld mit mir und all die liebevollen Aufmunterungen wenn auch ab und zu Überstunden angesagt waren. Meinen beiden Mitbewohnern Jonas und Stefan für ihre Freundschaft. Ein grosses Dankeschön gebührt meinen Eltern für ihre andauernde Unterstützung, sowohl moralisch als auch finanziell. Was ihr investiert ist unermesslich.

Ich danke Gott, meinem Vater im Himmel - für mein Leben und was Er hineingelegt hat.

Inhalt

Danksagung *i*

1: Einleitung *1*

2: GPS in der Überwachung von Massenbewegungen *3*
2.1 Das NAVSTAR Global Positioning System 4
2.2 Grundsätzliche Funktionsweise eines satellitengestützten Navigationssystems 5
2.2.1 Positionsbestimmung 5
2.2.2 Das Problem der Uhrensynchronisation 6
2.3 Systembedingte Fehlerquellen und aktuelle Lösungsansätze 7
2.3.1 Fehlerquellen 7
2.3.2 Ansätze zur Steigerung der Genauigkeit 7
2.4 Fallbeispiel: Überwachung des Dirru Gletschers mit GPS Einzelfrequenzempfängern 9

3: Formulierung der Problemstellung *13*
3.1 Grundsätzliche Herausforderungen 13
3.2 Aufgabenstellung 14
3.3 Teilprobleme 15

4: Ein generisches Kommunikations- und Kontrollsystem *17*
4.1 Motivation 17
4.2 Anforderungsprofil 18
4.3 Designvarianten 20
4.4 Spezifikation 21
4.4.1 Spezifikation in Worten 23
4.4.2 Spezifikation als Statechart 24
4.4.3 Analyse der Fehlertoleranz 27
4.5 Implementierung 28
4.5.1 Implementierung auf dem Gumstix Embedded PC 28
4.5.2 Implementierung auf dem TinyNode 33
4.5.3 Implementierung im Backend 33

5: Das GPS Subsystem *37*
5.1 Der GPS Empfänger: u-blox ANTARIS 4 37
5.2 Spezifikation des GPS Subsystems 38
5.3 Implementierung 39
5.3.1 Implementierung auf dem Sensor: Das PSBacklog-Plugin . . . 39
5.3.2 Implementierung im Backend 40
5.3.3 Datenkonversion in RINEX: csv2rinex.php 41

6: Betrieb des GPS Systems und erste Testergebnisse *43*
6.1 Beschreibung des Systemtests 43
6.2 Systemkonfiguration 45
6.2.1 Konfiguration des GPS Empfängers 45
6.2.2 Konfiguration des Gumstix Embedded PC 46
6.2.3 Weitere Konfiguration 47
6.3 Diskussion der Ergebnisse 47
6.3.1 Auswertung der GPS Messungen 47
6.3.2 Energieverbrauch 49
6.3.3 Übertragene Datenmengen 49
6.3.4 Allgemeine Bemerkungen 50

7: Fazit *51*

Abbildungen

2-1 Das Grundprinzip von GPS: Messung der Signallaufzeit 5
2-2 Bestimmung der Position durch den Schnitt von drei Kugeln 6
2-3 Die Messstelle am Dirru Gletscher 10
2-4 Tägliche statische Lösungen mit lokaler Referenzstation 11
2-5 Kinematische Lösungen mit lokaler Referenzstation 11
2-6 Kinematische Lösungen mit AGNES Referenzstation Zermatt 12

4-1 PermaSense XL Netzwerk Architektur 18
4-2 Skizze der für unser Problem relevanten Systemkomponenten 20
4-3 Designvarianten für das Kontroll- und Kommunikationssystem 22
4-4 Statechart der Kommunikations- und Kontrollschicht 26

5-1 u-blox AEK 4T Evaluation Kit 38

6-1 Die GPS Empfängerstation 44
6-2 Versuchsaufbau auf dem Dach des ETZ Gebäudes 44
6-3 Resultate der Positionsbestimmung 47
6-4 Resultate der Positionsbestimmung in topozentrischer Darstellung . . 48
6-5 Aufteilung des Stromverbrauchs einer Empfängerstation 49

Tabellen

2-1 Fehlerquellen in GPS 7

4-1 Verbrauch der Basestation in verschiedenen Betriebsmodi 21
4-2 Events und Actions im Statechart 25
4-3 Nachrichten zur Kommunikation mit dem TinyNode 30
4-4 Mögliche Wakeup Reasons 31
4-5 Die Beacon Nachrichten um den Gumstix ein- und auszuschalten . . . 34

5-1 Parameter des `GPSPlugins` 39

1

Einleitung

Das PermaSense Projekt[1] befasst sich mit der Beobachtung physikalischer Parameter des Permafrostes in den Schweizer Alpen. Zu diesem Zweck entwickelt und betreibt das Projekt verschiedene Messsysteme. Zu den existierenden Messungen von Temperaturen, Spaltausdehnung und Leitfähigkeitsprofilen sollen nun neue Sensoren entwickelt werden, die es erlauben großflächig verteilte Massenbewegungen zu erfassen.

Zu diesem Zweck kann das satellitengestützte GPS System verwendet werden. Die standardmässig verfügbaren GPS Empfänger erlauben jedoch nur eine Positionsbestimmung mit unzureichender Genauigkeit für die geplante Anwendung im PermaSense Projekt. Die Alternative sind Doppelfrequenzempfänger, welche speziell für die Geodäsie entwickelt werden. Diese sind jedoch sehr teuer und mit einem erheblichen Ressourcenaufwand verbunden. Neuere Entwicklungen zeigen aber dass mit Hilfe von geeigneten Postprocessingmethoden und Algorithmen auch relativ kleine, billige und stromsparende Empfänger eingesetzt werden können und trotzdem eine hohe Genauigkeit erzielt werden kann. Zu diesem Zweck müssen die Rohdaten des GPS Empfängers an einen zentralen Server zur Nachbearbeitung übertragen werden.

Diese Arbeit dokumentiert die Entwicklung eines solchen Systems basierend auf der bestehenden PermaSense Architektur und handelsüblichen GPS Empfängern der Firma u-blox. Sie ist in folgende Kapitel gegliedert:

- **Kapitel 1** ist diese Einleitung.
- **Kapitel 2** beschäftigt sich mit GPS im allgemeinen und geht näher auf die Anwendungsmöglichkeiten im Kontext von Massenbewegungen im Gelände ein.
- **Kapitel 3** formuliert die konkrete Problemstellung auf welche sich diese Arbeit fokussiert.

[1] http://www.permasense.ch

- **Kapitel 4** dokumentiert die Entwicklung einer generischen Kommunikations- und Kontrollschicht welche in der PermaSense Architektur für experimentelle Sensoren eingesetzt werden kann.
- **Kapitel 5** beschreibt wie die GPS Messung technisch in PermaSense integriert wurde.
- **Kapitel 6** beschäftigt sich am Beispiel eines ersten Systemtests mit der Frage wie das entwickelte GPS System in PermaSense betrieben werden kann und gibt diesbezügliche Schritt-für-Schritt Anweisungen.
- **Kapitel 7** zieht schliesslich ein Fazit aus der geleisteten Arbeit und diskutiert kurz Möglichkeiten zur Verbesserung und Weiterentwicklung der vorgestellten Komponenten.

Der resultierende Prototyp ist zugegebenermassen noch nicht ganz feldtauglich, beschäftigt sich doch diese Semesterarbeit am Beispiel von GPS Messungen mit viel allgemeineren Fragen bezüglich flexiblen und gleichzeitig verlässlichen Betriebsmodi für experimentelle Sensoren im Kontext von PermaSense. Er schafft aber eine solide Basis für zukünftige Experimente mit welchen die nötigen Erfahrungen für einen produktiven Einsatz im Feld verhältnismässig einfach gewonnen werden können.

2

GPS in der Überwachung von Massenbewegungen

Die Idee der Beobachtung grossflächiger Massenbewegungen mittels GPS ist keineswegs neu. Mehrere Teams von verschiedenen Universitäten haben in den letzten Jahren diesbezüglich Methoden entwickelt und experimentelle Erfahrungen gesammelt (z.B. [9], [6], [14] u.a.). Bis anhin wurden für solche Anwendungen aber meist Hochpräzisionsempfänger eingesetzt (sog. geodätische Zweifrequenzempfänger), welche zwar eine sehr genaue Messung der Position ermöglichen, aber auch dementsprechend teuer sind: ein solcher Empfänger samt Antenne kostet gut und gerne mehrere zehntausend Schweizer Franken. Dies wiederum macht sie denkbar ungeeignet für Messungen an sehr exponierten Stellen welche beispielsweise Steinschlag oder anderen starken Umwelteinflüssen ausgesetzt sind - zu gross wäre der finanzielle Aufwand um ein solches Präzisionsgerät als sog. "sacrificial Sensor"[1] einzusetzen.

Die Notwendigkeit für solch hochpräzise Messungen wird erst völlig klar, wenn man die Präzisionsanforderung eines Systems zur Überwachung von Massenbewegung mit der Genauigkeit eines "normalen" GPS-Empfängers (wie sie z.B. in Navigationssystemen für Autolenker verwendet werden) vergleicht: Die Geschwindigkeit eines solchen Rutschhanges liegt meist bei maximal einigen Zentimetern pro Tag, während ein Consumer-GPS Genauigkeiten von bestenfalls ca. 5m erreicht. Auf den ersten Blick scheint also der Einsatz solch günstiger Geräte für diese Anwendungsklasse durchwegs ausgeschlossen. Neuere Entwicklungen und Arbeiten zeigen aber, dass mit geeigneten Mess- und Postprocessingmethoden die systembedingten Fehler dieser günstigen Empfänger weitgehend eliminiert werden können und somit einer Verwendung dieser Komponenten in der Überwachung von grossflächigen Massenbewegungen nichts mehr im Wege steht.

In diesem Kapitel werden die Grundzüge des Global Positioning Systems so weit skizziert dass die problematischen Punkte ersichtlich werden. Die für diese Arbeit

[1] also ein "sich aufopfernder Sensor" welcher im Laufe der Messungen auch zerstört werden kann

relevanten Lösungsansätze zur Minderung der systembedingten Messfehler werden kurz beschrieben und schliesslich ein aktuelles Experiment präsentiert, in welchem günstige GPS Empfänger erfolgreich zur Überwachung eines Rutschhangs eingesetzt werden.

2.1 Das NAVSTAR Global Positioning System

Die Geschichte des *NAVSTAR Global Positioning System* - im allgemeinen kurz als "GPS" bezeichnet - reicht bis in die frühen 70er Jahre des letzten Jahrhunderts zurück. Die ersten Satelliten wurden 1978 in die Umlaufbahn gebracht, seither wurde das System ständig erweitert und auf eine Gesamtzahl von 32 Satelliten ausgebaut. Ursprünglich als rein militärisches System konzipiert, wurde 1983 auch die Nutzung für den zivilen Gebrauch geöffnet. Bis ins Jahr 2000 war allerdings das mit zivilen Geräten empfangene Signal künstlich gestört, so dass die eigene Position nur auf ca. 100m genau bestimmt werden konnte. Erst im Mai 2000 wurde auch diese Hürde abgeschafft und fortan waren auch mit günstigen Empfängern Messungen mit einer Genauigkeit von wenigen Metern möglich.

Das NAVSTAR GPS ist das bisher einzige Satellitennavigationssystem welches vollen Funktionalitätsstatus erreicht hat. Weitere Systeme befinden sich aber derzeitig im Aufbau:

- Das russische **GLONASS** Projekt, welches seit 1973 in der Entwicklung ist und kurz vor der Fertigstellung steht. Bereits sind Positionsbestimmungen mit einer Genauigkeit von unter 10m möglich, im Jahr 2010 - wenn alle Satelliten im All sind - sollen es weniger als 5m sein.
- Auch das europäische **GALILEO** System befindet sich in der Endphase der Entwicklung und soll bis 2014 zu 100% einsatzbereit sein. GALILEO bietet mehrere Services mit verschiedener Genauigkeit; die Standard-Genauigkeit des sog. "Open Service" - also des frei erhältlichen Services - soll 15m in horizontaler und 35m in vertikaler Richtung betragen.
- Das chinesische **Beidou** System war in einer ersten Version - welche auf geostationären Satelliten basierte - nur lokal für China verfügbar, in einer weiteren Ausbauphase soll es nun zu einem globalen System ausgebaut werden. Der derzeitige Projektstatus ist unklar; offiziell wird als Vollendungsdatum das Jahr 2013 angepeilt.

Im Moment ist das NAVSTAR GPS jedoch mit Sicherheit das am häufigsten verwendete satellitengestützte Navigationssystem. Aktuelle Entwicklungen in der Empfängertechnologie versuchen die Signale der verschiedenen Systeme zu kombinieren und somit eine bessere Verfügbarkeit des Ortungsservices aufgrund der höheren Anzahl an verfügbaren Satelliten zu erreichen. Solche Kombi-Empfänger sind bereits im Handel und werden wohl - zumindest für die zivile Nutzung - in absehbarer Zeit die auf nur ein System beschränkten Empfangsgeräte vom Markt verdrängen. Die in dieser Arbeit verwendeten Empfänger sind aber aus einer Generation welche noch alleine auf den Empfang von NAVSTAR GPS Signalen beschränkt war (u-blox ANTARIS 4).

2.2 Grundsätzliche Funktionsweise eines satellitengestützten Navigationssystems

2.2.1 Positionsbestimmung

Satellitengestützte Navigationssysteme basieren alle auf dem selben Grundprinzip: Aufgrund von Laufzeitmessungen von mehreren Satellitensignalen wird die eigene Position bestimmt. Sendet also der Satellit eine Nachricht welche die genaue Sendezeit des Signals beinhaltet, kann der Empfänger aus der Zeitdifferenz zwischen Sende- und Empfangszeit die Signallaufzeit feststellen. Diese ergibt wiederum multipliziert mit der Lichtgeschwindigkeit die Distanz vom Empfänger zum Satelliten - die so genannte Pseudodistanz.

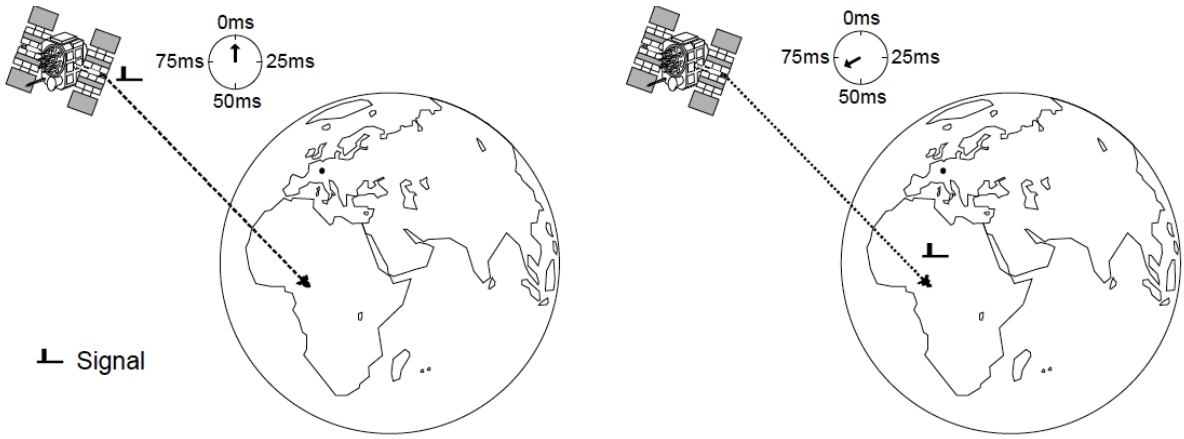


Abbildung 2-1
Das Grundprinzip von GPS: Messung der Signallaufzeit. Das Signal wird zum Zeitpunkt $t = 0ms$ *beim Satelliten abgesendet und erreicht den Empfänger zum Zeitpunkt* $t = 67.3ms$. *Die Distanz zum Satelliten errechnet sich durch* $d = \Delta t \cdot c$. *Quelle: [15]*

Da eine genaue Positionsbestimmung drei Koordinaten - also Breite, Länge und Höhe - beinhaltet, benötigt die Bestimmung dieser Grössen auch mindestens drei sichtbare Satelliten. Die möglichen Empfängerpositionen aus jeder Einzelmessung liegen alle auf einer Kugeloberfläche um den Satelliten mit Radius d; bei drei Satelliten schneiden sich diese drei Kugeln in zwei Punkten. Einer dieser Lösungspunkte kann ausgeschlossen werden da er sich zu weit weg von der Erde befände, der andere Schnittpunkt entspricht der Empfängerpostion in Raumkoordinaten. Abbildung 2-2 verdeutlicht dieses Prinzip.

Da zu dieser Berechnung auch die genauen Positionen der Satelliten bekannt sein müssen, überträgt jeder Satellit fortlaufen präzise Informationen über seine aktuelle Umlaufbahn - die so genannte Ephemeris. Auch ungefähre Informationen über die Umlaufbahnen aller anderen Satelliten - der so genannte Almanach - werden regelmässig übertragen; dies erlaubt es einem Empfänger Prognosen über zu erwartende Satellitenkonstellationen in der Zukunft anzustellen.

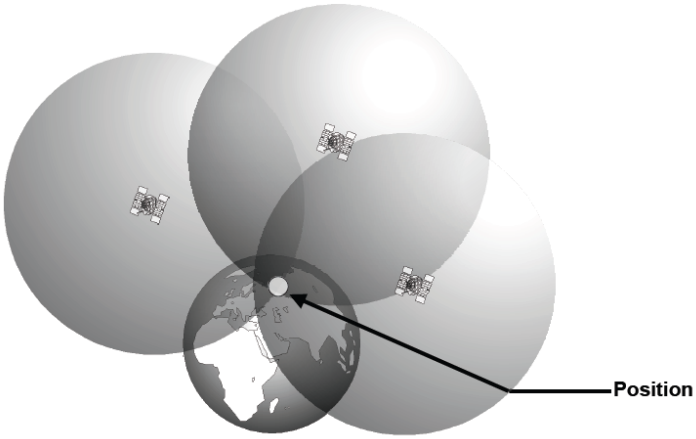


Abbildung 2-2
Bestimmung der Position durch den Schnitt von drei Kugeln. Quelle: [15]

2.2.2 Das Problem der Uhrensynchronisation

Die im Vorigen Abschnitt beschriebene Methode zur satellitengestützten Positionsbestimmung funktioniert nur, wenn die Uhren aller Satelliten untereinander und die lokale Uhr des Empfängers exakt aufeinander synchronisiert sind. Die Satelliten sind dazu alle mit hochpräzisen Atomuhren ausgerüstet welche regelmässig durch ein bodengestütztes Netz von Kontrollstationen miteinander synchronisiert werden. Im Falle der Empfänger wird hingegen meist mit quarzbasierten Uhren gearbeitet. Da eine Abweichung in der Zeit zwischen Empfängeruhr und Satellitenuhr aber enorme Auswirkungen auf die erreichte Präzision hat (schon eine Zeitdifferenz von $1\mu s$ wirkt sich in einem Positionierungsfehler von $300m$ aus) muss als Kompensation für diesen Uhrenfehler ein vierter Satellit zur Positionsbestimmung hinzugezogen werden - er liefert eine vierte Gleichung für die zusätzliche Unbekannte welche durch den Uhrenfehler im System besteht.

Meist sind sogar mehr als 4 Satelliten an der Empfängerposition sichtbar; Konstellationen mit 10 und mehr sichtbaren Sendern sind keine Seltenheit. Die zusätzlichen Satelliten können in die Berechnungen mit einbezogen werden um die Genauigkeit zu erhöhen; alternativ kann der Empfänger eine Auswahl treffen und nur die 4 Satelliten mit der besten Empfangsqualität bzw. Konstellation für die Berechnungen hinzuziehen.

Wir halten also fest:

- Mit GPS kann nicht nur die eigene Position, sondern auch die **genaue Zeit** bestimmt werden
- Für eine Positionsbestimmung mit GPS müssen mindestens 4 Satelliten sichtbar sein

2.3 Systembedingte Fehlerquellen und aktuelle Lösungsansätze

2.3.1 Fehlerquellen

Es gibt in satellitengestützten Navigationssystem und somit auch in GPS mehrere Fehlerquellen, welche die Genauigkeit der Positionsbestimmung erheblich beeinflussen; die wichtigsten sind in Tabelle 2-1 aufgeführt:

Störungen durch die Ionosphäre	$\pm 5m$
Schwankungen der Satellitenumlaufbahnen	$\pm 2.5m$
Uhrenfehler der Satelliten	$\pm 2m$
Mehrwegeffekt	$\pm 1m$
Störungen durch die Troposphäre	$\pm 0.5m$
Rechnungs- und Rundungsfehler	$\pm 1m$

Tabelle 2-1: Fehlerquellen in GPS. Quelle: [11]

Offensichtlich hat die *Ionosphäre* den grössten Einfluss auf die Genauigkeit des Systems. Diese äussere Schicht der Erdatmosphäre beginnt in einer Höhe von ca. 80km über der Erdoberfläche und geht in einer Höhe von ca. 600km in den interplanetaren Raum über. Die Teilchen der Ionosphäre sind durch die ultraviolette Strahlung der Sonne geladen. Diese Ladung konzentriert sich in mehreren Schichten innerhalb der Ionosphäre und diese Schichten brechen die vom Satelliten ausgesandten elektromagnetischen Wellen wodurch die Signallaufzeit des Signals verlängert wird. Grundsätzlich ist dieser Effekt bekannt und wird in jedem GPS Empfänger bei der Positionsberechnung ausgeglichen - jedoch ist dort nur eine Korrektur für Standardbedingungen möglich; unvorhersehbare Änderungen in der Ionosphäre - z.B. aufgrund starker Sonnenwinde - sind nur kompensierbar wenn diese Änderungen gemessen werden können. Mehr dazu im Abschnitt 2.3.2.

Insgesamt ergeben die aufsummierten Werte aus Tabelle 2-1 einen Fehler von $\pm 12m$, wobei sich die einzelnen Effekte gegenseitig auch gegenseitig abschwächen oder eliminieren können. Dadurch ist die erreichte Genauigkeit im Normalfall besser und liegt bei guten Bedingungen bei ca. $5m$.

2.3.2 Ansätze zur Steigerung der Genauigkeit

Wir haben im vorigen Abschnitt festgestellt, dass der Fehler durch die Ionosphäre im Normalfall den grössten Unsicherheitsfaktor darstellt. Gelingt es also diesen Fehler zu eliminieren oder zumindest möglichst klein zu halten, so hat sich die erreichbare Präzision schon massgeblich verbessert. Dafür existieren mehrere Ansätze und Möglichkeiten:

2.3.2.1 Mehrere Frequenzen zur Ionosphärenkompensation

Zur Kompensation des Ionosphärenfehlers ist eine Gegenmassnahme besonders effektiv: Die Verwendung einer zweiten Trägerfrequenz für das übermittelte Signal. Im Fall von GPS spricht man auch von der L1 und der L2 Frequenz. Das Prinzip

basiert auf der Tatsache dass Veränderung der Signalgeschwindigkeit in der Ionosphäre proportional zu $\frac{1}{f^2}$ ist, wobei f die Signalträgerfrequenz bezeichnet. Wird nun das Signal auf zwei Frequenzen parallel übertragen, so kann aus den verschiedenen Signallaufzeiten der aktuelle Ionosphärenfehler abgeschätzt und in der Berechnung kompensiert werden. Wie schon in der Einleitung dieses Kapitels erwähnt sind Empfänger für dieses Verfahren aber sehr kostspielig und empfindlich und kommen daher für einen Einsatz in Gebieten wo sie den Naturgewalten praktisch hilflos ausgesetzt wären nicht in Frage.

2.3.2.2 Differentielles GPS

Unter dem Begriff "differentiellem GPS" (kurz: DGPS) werden mehrere Methoden zusammengefasst, welche sich aber alle das selbe Grundprinzip zur Steigerung der Genauigkeit zu Nutze manchen: Liegen mehrere Empfängerstationen geographisch nahe zusammen, so sind die beobachteten systematischen Fehler bei allen Empfängern in etwa gleich gross. Ist nun eine Referenzstation vorhanden deren genaue Position schon im Voraus bekannt ist, so können aus der Abweichung zwischen gemessenen und tatsächlichen Werten Rückschlüsse auf die aktuell im System vorhandenen Fehler - einführt durch Ionosphäre, ungenaue Satellitenuhren etc. - gezogen werden. Wenn nun diese Korrekturdaten an weitere Empfängerstationen in der Nähe weitergegeben werden - entweder in Echtzeit oder später im Postprocessing (siehe Abschnitt 2.3.2.3) - so können diese auch dort in die Positionsberechnung mit einfliessen.

Grundsätzlich gibt es zwei Methoden wie die Referenzstation den aktuellen Positionsfehler bestimmen kann:

- Durch Vergleich der gemessenen Pseudodistanzen mit den tatsächlichen Entfernungen zu den einzelnen Satelliten. Diese Methode ist nicht sehr aufwändig und kann daher gut in Echtzeit durchgeführt werden. Erreichbare Genauigkeiten für umliegende Stationen unter Verwendung der so gewonnenen Korrekturdaten liegen im Bereich von ca. 1m.
- Durch eine Messung der Trägerphase - also der Phase des empfangenen Satellitensignals verglichen mit dem internen Referenzsignal der Empfänger. Da die Signalphase mit relativ wenig Aufwand sehr genau bestimmt werden kann, können mit dieser Messmethode im besten Fall Genauigkeiten von einigen Millimetern erreicht werden. Jedoch sind dafür sehr komplexe mathematische Berechnungen mit Messdaten über längere Zeiträume nötig, welche im Normalfall nicht auf den Empfängern selbst durchgeführt werden. Diese Methode ist also meist nur in Kombination mit Postprocessing machbar (siehe Abschnitt 2.3.2.3), allerdings ist dies die einzige Möglichkeit um Positionsbestimmungen in der Präzision zu erhalten, welche für unsere Anwendung - nämlich die Überwachung von Massenbewegungen im Gelände - nötig ist.

2.3.2.3 Postprocessing

Postprocessing ist ein Sammelbegriff unter welchem jegliche Methoden der nachträglichen - also nicht in Echtzeit stattfindenden - Datenverarbeitung zur Positionsbestimmung zusammengefasst werden können. Anstatt dass also die GPS-

Empfänger ihre Positionen "online" direkt bestimmen - möglicherweise auch unter Zuhilfenahme von Korrekturdaten aus einer Referenzstation welche in Echtzeit übermittelt werden - werden die gemessenen Daten wie Pseudodistanz, Trägerphase und aktuelle Dopplerverschiebung (welche durch die Bewegung des Satelliten relativ zum Empfänger entsteht) gespeichert und später einer spezialisierten Software übergeben. Diese Programme sind in der Lage die systematischen Fehler in Messungen über genügend lange Zeit weitgehend zu kompensieren und dadurch genauere Positionsbestimmungen zu produzieren als dies online in den Empfängern möglich wäre, da diese im Normalfall nicht mit genügen Rechenkapazität ausgestattet sind.

2.3.2.4 Kombinationen

Natürlich können obige Ansätze miteinander kombiniert werden: So verspricht beispielsweise DGPS mit Zweifrequenzempfängern bessere Resultate als die entsprechenden Ansätze alleine, und auch Postprocessing kann mit differentiellen GPS-Messungen eine höhere Präzision erreichen als wenn nur die Daten eines Empfängers prozessiert werden. Es gilt also die für eine Anwendung optimale Kombination der Ansätze zu finden welche die Anforderungen an Genauigkeit, Preis, Aufwand und Geschwindigkeit entsprechend gewichtet berücksichtigt.

Der Ansatz welcher in dieser Arbeit näher betrachtet wird ist eine Kombination aus DGPS und Postprocessing. Allerdings werden für das DGPS ausschliesslich günstige Einzelfrequenzempfänger eingesetzt, und auch die Position der Referenzstation ist nicht zwingend mit höchster Präzision bestimmt. Da aber in unserem Fall weniger die absolute Position als vielmehr die relative Bewegung der Empfänger in Bezug auf eine fixe Referenzstation interessant ist, hat letzterer Faktor keinen weiteren Einfluss auf die Resultate.

2.4 Fallbeispiel: Überwachung des Dirru Gletschers mit GPS Einzelfrequenzempfängern

Ein konkretes Beispiel des im letzten Abschnitt erwähnten Ansatzes - also der Kombination aus DGPS mit Einzelfrequenzempfängern und Postprocessing - ist die Messkampagne welche durch das Institut für Geodäsie und Photogrammetrie der ETH Zürich im Sommer 2009 gestartet wurde. Dabei wurde die Bewegung des Dirru Gletschers oberhalb von Breitmatten im Mattertal (Kanton Wallis) untersucht. Eingesetzt wurden dafür drei GPS Empfängerstationen, und zwar alles Einzelfrequenzempfänger der Firma u-blox (ANTARIS 4 Empfängerchips). Eine Station wurde als Referenzstation fix an geeigneter Stelle neben dem Gletscher installiert, die anderen beiden Stationen auf dem sich bewegenden Geröll welches die Gletscherzunge bedeckt.

Die Rohdaten welche von den GPS Empfängern gemessen wurden (im Wesentlichen die Pseudodistanz, Trägerphase und Dopplerverschiebung jedes zum Samplingzeitpunkt sichtbaren Satelliten) wurden lokal auf einem Datenlogger gespeichert. Nach 4 Monaten Messzeit - durchgehend mit einer Samplingrate von 12 Samples pro Minute - wurden die Datenlogger ausgetauscht und die bis dahin gemessenen Daten

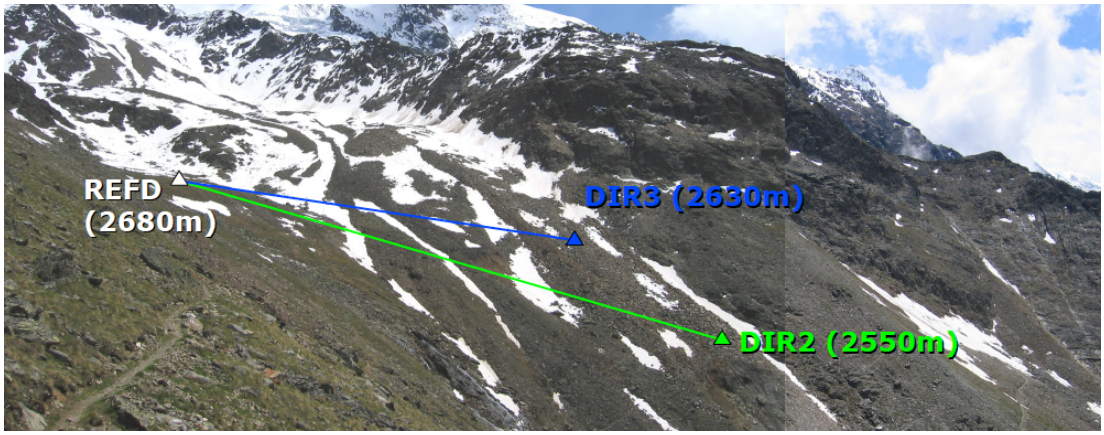


Abbildung 2-3
Die Messstelle am Dirru Gletscher. REFD ist neben dem Gletscher positioniert, DIR2 und DIR3 auf dem Gletscher. Der Abstand zwischen der REFD und DIR2 beträgt $360m$ *horizontal und* $130m$ *vertikal, derjenige zwischen REFD und DIR3* $285m$ *horizontal und* $50m$ *vertikal. Photo: Dr. Philippe Limpach, Institut für Geodäsie und Photogrammetrie, ETH Zürich*

ausgewertet. Da offenbar schon wenige Tage nach Messbeginn die Referenzstation durch ungeklärte Umstände beschädigt wurde, konnten deren Daten nur für diese kurze Zeitperiode als Referenzpunkt verwendet werden. Für die restliche Messdauer musste auf die Daten der AGNES[2] Station in Zermatt ausgewichen werden, welche sich ca. $15km$ entfernt im $700m$ tiefer gelegenen Talgrund befindet. Für die Datenauswertung wurde die vom Astronomischen Institut der Universität Bern entwickelte *Bernese GPS Software 5.0*[3] verwendet.

Bei der Auswertung zeigte sich, dass die Bewegung des Gletschers tatsächlich in beachtlicher Genauigkeit mitverfolgt werden konnte. Die Samples wurden entweder zu einer so genannten "daily static solution" aggregiert welche pro Tag eine Positionslösung ergeben, oder zu so genannten "kinematic solutions" wo alle korrigierten Positionslösungen einzeln betrachtet werden. Die Resultate dieser Auswertungen sind in Abbildungen 2-4, 2-5 und 2-6 dargestellt.

Ohne an dieser Stelle auf die genaueren statistischen Untersuchungen dieser Messresultate eingehen zu wollen, kann aus der gleichmässig erkennbaren Bewegung in den täglichen Lösungen sowie der konstant geringen Variabilität der kinematischen Lösungen um die Bewegungsachse herum darauf geschlossen werden dass die gewonnenen Messdaten tatsächlich die erforderliche Präzision für zentimetergenaue Positionsbestimmungen bieten und somit zur Erfassung von Massenbewegungen im hochalpinen Gelände geeignet sind.

[2] Automatisches GPS Netz Schweiz - ein vom Bundesamt für Landestopographie betriebenes Netz von GPS Referenzstationen für DGPS

[3] `http://www.bernese.unibe.ch`

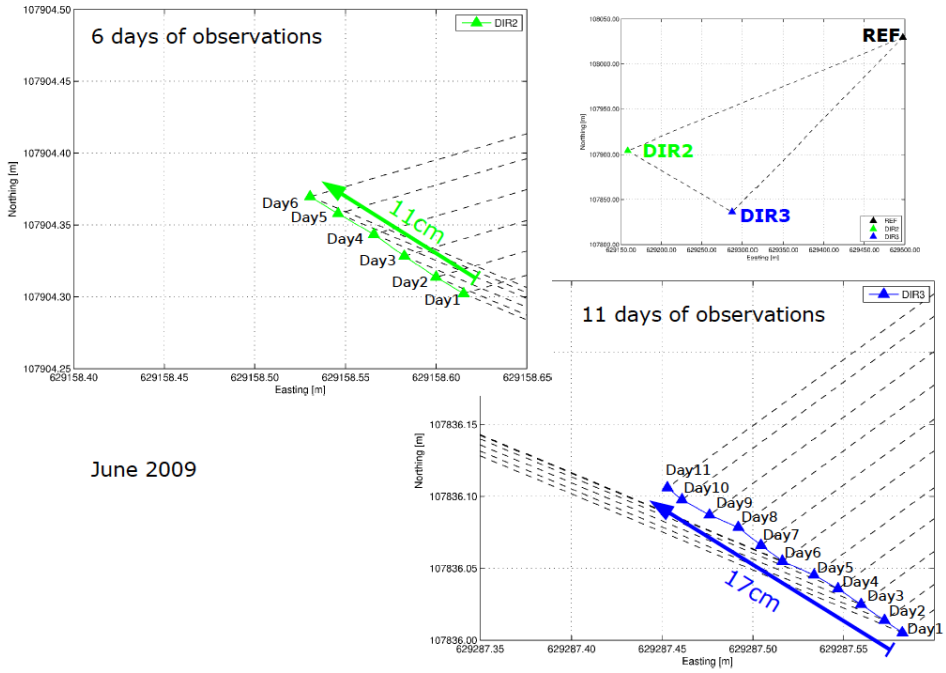


Abbildung 2-4
Tägliche statische Lösungen mit lokaler Referenzstation. Quelle: Dr. Philippe Limpach, Institut für Geodäsie und Photogrammetrie, ETH Zürich

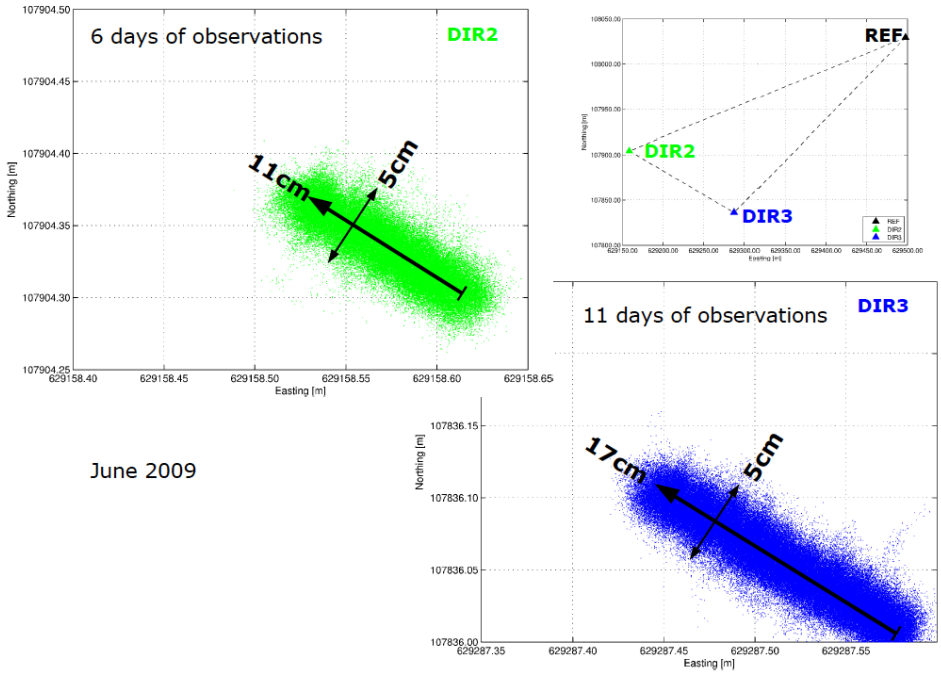


Abbildung 2-5
Kinematische Lösungen mit lokaler Referenzstation. Quelle: Dr. Philippe Limpach, Institut für Geodäsie und Photogrammetrie, ETH Zürich

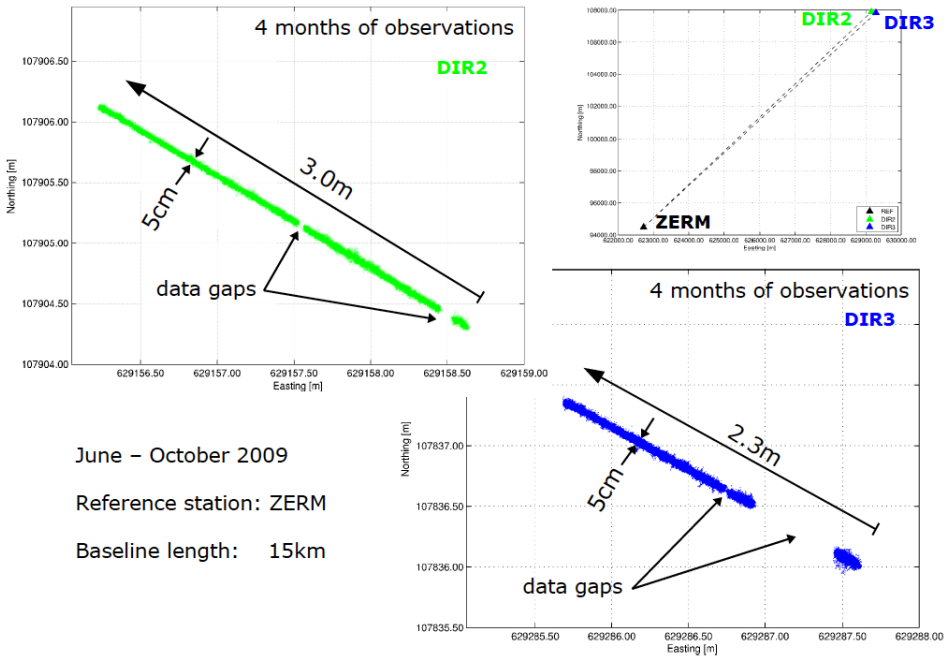


Abbildung 2-6
Kinematische Lösungen mit AGNES Referenzstation Zermatt. Quelle: Dr. Philippe Limpach, Institut für Geodäsie und Photogrammetrie, ETH Zürich

3

Formulierung der Problemstellung

3.1 Grundsätzliche Herausforderungen

Im letzten Abschnitt des vorigen Kapitels wurde am Beispiel der Überwachung des Dirru Gletschers aufgezeigt, dass sich auch mit handelsüblichen GPS Einzelfrequenzempfängern Messgenauigkeiten im Zentimeterbereich erzielen lassen. Denkt man aber über mögliche Anwendungen solcher Messsysteme nach - z.B. der Einsatz als Warnsystem für Erdrutsche und ähnliche Umweltkatastrophen - so weist die vorgestellte Versuchsanordnung einen entscheidenden Nachteil auf: *Die - technisch gesehen - komplette Isolation des Messsystemes von der Umwelt.* Der Messanordnung fehlt jegliche Möglichkeit zur Kommunikation mit der Aussenwelt; die Sensoren sind weder untereinander noch mit anderen Anschlusspunkten vernetzt. Dieser Fakt bringt einige Implikationen mit sich, welche vorliegendes System für Anwendungen wie z.B. ein Warnsystem ungeeignet machen:

- *Die Daten sind erst verfügbar nachdem die Datenlogger manuell ausgewechselt und ausgelesen wurden.* Es besteht keine Möglichkeit herauszufinden, ob sich der Gletscher z.B. in der heutigen Nacht besonders stark bewegt hat und ob eine akute Gefahr von ihm ausgeht.
- *Es existiert keine Möglichkeit zur Systemüberwachung.* In vorgestelltem Experiment wurde erst nach vier Monaten bemerkt, dass die Referenzstation während fast der ganzen Messperiode keine brauchbaren Daten geliefert hatte. Der "Gesundheitszustand" des Systems kann nur durch einen - unter Umständen sehr aufwändigen - Besuch auf der Messstelle selbst beurteilt werden. Die fehlende Möglichkeit zur Fernüberwachung hat auch zur Folge dass die Systemverlässlichkeit in Frage gestellt werden muss: Wenn nur Messdaten geliefert werden ohne Informationen dazu *unter welchen Umständen* diese Messungen erfolgten, so ist die Beurteilung der Verlässlichkeit dieser Daten sehr schwierig oder gar unmöglich.
- *Die Systemparameter sind fix und können nur mit grossem Aufwand geändert werden.* Soll z.B. die Samplingrate adaptiv verändert werden, muss das Sy-

stem dies autonom entscheiden können. Der Benutzer hat keine Möglichkeit die Messung veränderten Umständen anzupassen ausser dass er das System am Messstandort rekonfiguriert.

Ziel dieser Arbeit ist es, das erfolgreiche Messkonzept in die bestehende PermaSense Infrastruktur zu integrieren und damit den Grundstein für eine zeitnahe Datenauswertung sowie eine zuverlässige Fernüberwachung zu legen. Weiter wurde zum Ziel gesetzt einen möglichst effizienten Betriebsmodus zu evaluieren - statt den energieintensiven Messungen rund um die Uhr sollen Messzeitpunkt, -dauer und -samplingrate parametrisiert werden können. Damit kann ein für die jeweilige Fragestellung optimaler Tradeoff zwischen Energieverbrauch und erreichter Präzision erreichet werden.

3.2 Aufgabenstellung

Die originale Aufgabenstellung ist wie folgt formuliert:

> In dieser Arbeit soll auf Basis der existierenden PermaSense Systemarchitektur eine GPS Anwendung entwickelt und erprobt werden. Der Sensor soll sich optimal in das stromsparende PermaDozer Netzwerk einbinden lassen und kann ggf. auch eine weitere Netzwerkressource zur Datenübertragung (WLAN) benutzen. Der Sensor soll für den Betrieb im Hochgebirge ausgelegt sein, keine manuellen Interventionen benötigen und von einer kleinen Solarzelle gespiesen werden.
>
> Die Knappheit an verfügbarer Energie erfordert ein angepasstes Betriebskonzept für Komponenten, deren Dauerbetrieb die verfügbaren Ressourcen überbeanspruchen würde. Während es sowohl möglich sein soll den Betriebsplan dynamisch über eine Netzwerkverbindung aus der Ferne zu modifizieren, so soll auch bei einer Nichtverfügbarkeit der Netzwerkverbindung zur Systemkontrolle ein autonomer Betrieb des Sensors gewährleistet sein. Als dritte Möglichkeit soll zudem vor Ort ein "Handbetrieb" des Sensors möglich sein.
>
> **Aufgabenstellung**
>
> - Erstellen Sie einen Projektplan und legen Sie Meilensteine sowohl zeitlich wie auch thematisch fest. Erarbeiten Sie in Absprache mit dem Betreuer ein Pflichtenheft.
> - Machen Sie sich mit den relevanten Arbeiten im Bereich Sensornetze, GPS und Low-Power Wireless vertraut. Führen Sie eine Literaturrecherche durch. Suchen Sie auch nach relevanten neueren Publikationen. Prüfen Sie welche Ideen/Konzepte Sie aus diesen Lösungen verwenden können.
> - Arbeiten Sie sich in die relevanten Technologien GPS, GPS Postprocessing, PermaDozer, Data Management, PermaSense Baseboard und GSN ein.

- Erstellen Sie eine Übersicht der Anforderungen an einen GPS Sensor für das PermaSense Projekt. Beziehen Sie hierzu auch die bestehenden Erfahrungen des PermaSense Projektes mit ein. In Bezug auf Know-how im Bereich GPS soll auf das Wissen der Gruppe von Prof. Alain Geiger zurückgegriffen werden.
- Achten Sie darauf, dass die angestrebte Lösung auch in der Praxis anwendbar ist. Dies gilt insbesondere für die Konzepte und Implementierung der Software für die Steuerung.
- Implementieren Sie dieses Konzept in einem lauffähigen Prototypen.
- Dokumentieren Sie Ihre Arbeit sorgfältig mit einem Vortrag, einer kleinen Demonstration, sowie mit einem Schlussbericht.

3.3 Teilprobleme

Aus einer Analyse obiger Aufgabenstellung ergeben sich folgende Teilprobleme:

- **Erfassung eines Anforderungsprofils an das Kommunikations- und Kontrollsystem**

  Da die angestrebte Systemarchitektur genug Flexibilität für weitere Sensortypen bieten soll, müssen als erstes die genauen Anforderungen an ein solches Basissystem definiert werden. Nur so kann sichergestellt werden dass die folgende Spezifikation und Implementierung für die weitere Verwendung geeignet ist.

- **Spezifikation des Kommunikations- und Kontrollsystems**

  Aus dem erstellten Anforderungsprofil wird das grundsätzliche Systemverhalten spezifiziert. Anschliessend werden verschiedene Varianten zur Realisierung dieses Verhaltens evaluiert und die geeignetste ausgewählt. Es folgt eine Detailspezifikation für das Verhalten der einzelnen Komponenten sowie der erforderlichen Kommunikationsprotokolle.

- **Implementierung des Kommunikations- und Kontrollsystems**

  Das spezifizierte System wird implementiert und getestet.

- **Spezifikation des GPS Subsystems**

  Die zu verwendenden Postprocessing Methoden werden studiert, besonders im Hinblick auf die dafür benötigten Daten. Daraus werden die Spezifikationen des GPS Systems abgeleitet.

- **Implementierung des GPS Subsystems**

  Das spezifizierte GPS Subsystem wird - aufgesetzt auf das zuvor entwickelte Basissystem - implementiert und getestet.

- **Systemtest und Fazit**

  Das komplette System wird unter möglichst realen Bedingungen auf korrekte Funktion getestet. Möglichkeiten zur Weiterentwicklung und Effizienzsteigerung werden aufgezeigt und kurz diskutiert.

4

Ein generisches Kommunikations- und Kontrollsystem

4.1 Motivation

Die PermaSense Systemarchitektur [3] war bisher vor allem darauf ausgelegt, relativ kleine Datenmengen zu übertragen. Die so genannten Sensor Nodes - basierend auf der Shockfish TinyNode Plattform [5] - kommunizieren über das äusserst energieeffiziente Dozer Netzwerkprotokoll [4], welches genau für solche Szenarien ausgelegt ist - kleine Datenmengen in regelmässigen Zeitabständen zuverlässig zu transportieren. Für wenig datenintensive Messungen wie z.B. zur Temperatur- oder Feuchtigkeitsbestimmung hat sich dieser Ansatz auch gut bewährt und das System verrichtet seit mehr als einem Jahr auf 3'500 m.ü.M nahezu reibungslos seinen Dienst. Für grössere Datenmengen ist er jedoch ungeeignet - zu gross wäre die Belastung für das Dozer Netzwerk, zu schnell wäre die Energiekapazität der Sensor Node Batterien erschöpft. Da von geowissenschaftlicher Seite her ein grosses Interesse daran besteht auch komplexere Sensoren wie z.B. hochauflösende Kameras, GPS Empfänger oder seismische Mikrofone einzusetzen, muss die existierende Architektur erweitert werden um auch Messungen mit Datenmengen im Bereich von mehreren Megabytes[1] durchführen und verarbeiten zu können.

Basis für die in diesem Kapitel entwickelte Architektur bilden die Erfahrungen welche im Rahmen von Mountainview [10] gesammelt wurden. Dafür wurde eine hochauflösende Kamera am Messstandort Matterhorn installiert, mit deren Bilder Aussagen über den aktuellen Zustand des überwachten Geländes - z.B. in Bezug auf Schneebedeckung - gemacht werden können. Der Aufnahmevorgang wird dabei immer mittels einer Beacon Nachricht über das Dozer Netzwerk ausgelöst. Dieser erste Ansatz birgt jedoch das Problem dass bei einem Ausfall der Dozer Verbindung die Kamera komplett offline ist - es existiert keine Möglichkeit mehr mit dem Gerät

[1] ein hochauflösendes Foto im RAW Format hat eine Grösse von ca. 10MB während die Pakete in PermaDozer um die 30 Bytes gross sind

zu kommunizieren bis es sich wieder mit Dozer verbinden konnte. Das Mountainview Konzept soll nun in einem nächsten Schritt erweitert und mit mehr Flexibilität ausgestattet werden. Die Hauptkriterien für diese Weiterentwicklung sind:

- Flexibilität in Bezug auf Datenaquisition (*wann* werden *welche* Daten *wie* gemessen) und zu übertragende Datenmenge. Neue Sensoren sollen schnell und einfach integrierbar sein.
- enge Integration in die PermaSense Gesamtarchitektur
- möglichst umfangreiche Fernsteuer und -überwachbarkeit

Die folgenden Überlegungen gehen dabei von der PermaSense XL Network Architecture aus, welche nebst dem bisherigen Dozer Netzwerk ein WLAN Netzwerk zur Übertragung grösserer Datenmengen vorsieht (welches auch in Mountainview zur Übertragung der Bilder genutzt wird). Trotzdem sollen die "XL Sensoren" (in Abb 4-1 als "GPS Receiver" beschriftet) wenn möglich auch ins Dozer Netzwerk integriert werden, zwecks der daraus entstehenden zusätzlichen Möglichkeit mit dem Knoten zu kommunizieren.

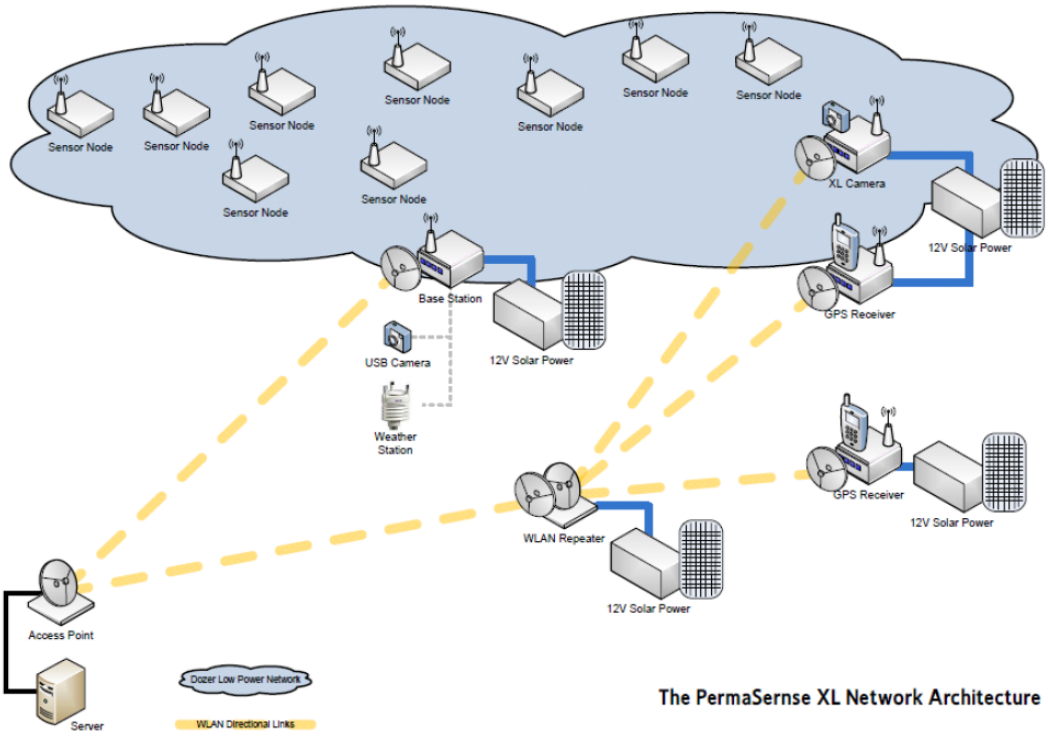


Abbildung 4-1
PermaSense XL Netzwerk Architektur

4.2 Anforderungsprofil

Allgemein formuliert benötigen wir für obige Problemstellung ein **unabhängiges und selbstgesteuertes System** welches jedoch **nach Möglichkeit neue Befehle**

und Konfigurationsänderungen anfordert und diese umsetzt. Konkret werden folgende Eigenschaften angestrebt:

- Das System fragt immer nach neuen Befehlen wenn die Möglichkeit dazu besteht
- Das System folgt immer den letzten erhaltenen Befehlen
- Das System führt seine Funktion immer aus, und dies ohne Abhängigkeit von der korrekten Funktion anderer Subsysteme
- Das System verliert keine Daten und übermittelt diese so bald wie möglich ins Backend
- Das System schaltet wenn immer möglich alle energieintensiven Komponenten aus um wertvolle Ressourcen zu sparen

Da die zu entwickelnde Kommunikations- und Kontrollschicht auf den Erfahrungen von Mountainview aufbauen sollte war für meine Arbeit von Beginn an klar, dass die zu Grunde liegende Plattform die *PermaSense Basestation v2* sein würde. Diese bietet aufgrund der integrierten Linux Plattform (Gumstix Embedded PC mit Openembedded Linux), der vorhandenen USB Schnittstellen, der schnellen Netzwerkanbindung (kabelgebunden oder WLAN über einen angeschlossenen WLAN Router) und ihrer Anbindung an das Dozer-Netzwerk durch den ebenfalls integrierten TinyNode viel Flexibilität in Bezug auf Betriebsmodi und angeschlossene Sensortypen. Alle Komponenten werden bereits produktiv auf für die laufenden Messungen am Matterhorn eingesetzt und funktionieren mit hoher Zuverlässigkeit.

Das grundlegende Konzept ist folgendes: Der Gumstix Embedded PC dient als Controller für die Datenaquisition und Datenübermittlung an den GSN Server im Datenbackend. Mit dem TinyNode und dessen Dozer Netzwerklink steht eine zusätzliche Kommunikationsverbindung ins Backend zur Verfügung.

Ausgehend von den Erfahrungen mit Mountainview wurden folgende Designkriterien für die Kommunikations- und Kontrollschicht definiert:

- **Allgemein**
  - Die auszuführenden Aufgaben/Tasks sollen mit einem Zeitplan (im Folgenden "Schedule" genannt) konfigurierbar sein, so dass zu jeder beliebigen Tageszeit ein ganz bestimmter Task ausgeführt werden kann.
  - Das System soll einfach rekonfigurierbar sein; insbesondere sollen die auszuführenden Tasks leicht aus dem Backend änderbar sein.
  - *Fehlertoleranz*: Störungen wie z.B. Netzwerkunterbrüche, Energieausfälle aufgrund leerer Batterien, Backend-Ausfälle etc. dürfen das System nicht in einen undefinierten Zustand bringen.
  - *Energieeffizienz*: Alle PermaSense Systemkomponenten im Feld müssen mit begrenzten Energieressourcen auskommen - im Falle der Basestation eine Batterie welche mittels Solarzellen gespeist wird. Ein sorgsamer Umgang mit den vorhandenen Ressourcen ist also unumgänglich.

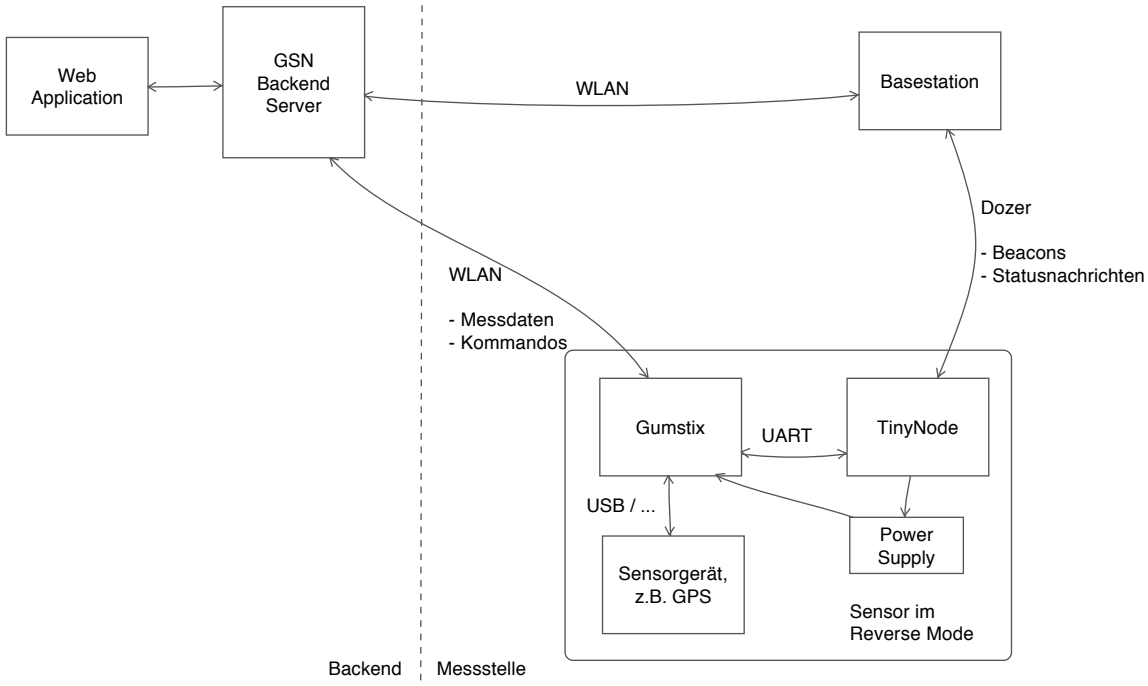


Abbildung 4-2
Skizze der für unser Problem relevanten Systemkomponenten

  - Möglichkeit zum "Handbetrieb": Das Ein- und Ausschalten der energieintensiven Systemkomponenten soll nicht nur per Software möglich sein, sondern auch manuell per Schalter.

- **TinyNode**
  - Statusinformationen über den derzeitigen Gumstix-Status (eingeschaltet, ausgeschaltet) sollen verwaltet und bei Bedarf abgerufen werden können.
- **Gumstix Embedded PC**
  - "Daily Service Wakeup": Unabhängig vom Task-Schedule soll der Gumstix täglich für eine bestimmte Zeitspanne online sein. Dieses "Service Window" kann genutzt werden um sich auf der Maschine anzumelden und Wartungsarbeiten durchzuführen.

4.3 Designvarianten

Schnell wurde klar dass insbesondere das Kriterium der Energieeffizienz nur dann zufriedenstellend erfüllbar ist, wenn der Gumstix Embedded PC in Ruhephasen ganz ausgeschaltet wird. Eine einfache Verbrauchsmessung welche in Tabelle 4-1 dokumentiert ist ergab nämlich, dass der Stromverbrauch auch im tiefsten Schlafmodus (der sog. Suspend to Memory Modus) immer noch zu hoch ist um von *Low-Power* sprechen zu können.

Normalbetrieb	Stanbdy Modus	Suspend to Memory Modus
140mA	100mA	70mA

Tabelle 4-1: Verbrauch der Basestation in verschiedenen Betriebsmodi bei einer Betriebsspannung von 12V

Es mussten also Möglichkeiten evaluiert werden, wie der Gumstix in den Ruhephasen ganz abgeschaltet werden kann. Nichtsdestotrotz soll das System nicht von einer existierenden Netzwerkverbindung abhängig sein um seine Tasks auszuführen, ansonsten würden Netzwerkausfälle zu unerwünschten Messlöchern führen (wie dies bis anhin bei Mountainview der Fall war). Das Ein- und Ausschalten via Kommandos übers Netzwerk soll jedoch im Sinne einer Notfallsteuerung weiterhin möglich sein.

Mit der in einer Basestation vorhanden Hardware sind mehrere Betriebsmodi denkbar, welche evaluiert wurden und in Abb. 4-3 beschrieben sind.

All diese Varianten wurden eingehend diskutiert. Der Entscheid fiel schliesslich auf Variante 2 *Full power cycle mit Gumstix schedule - TinyNode controlled*, und zwar aus folgenden Überlegungen:

- Die Verwaltung des Schedules ist auf dem Gumstix viel einfacher zu implementieren als auf dem TinyNode. Zwischen Gumstix und TinyNode steht eine UART-Schnittstelle zur Verfügung, auf welcher ohne viel Aufwand ein Kommunikationsprotokoll umgesetzt werden kann um die Wakeups zu koordinieren.
- Die Variante mit der RTC (Real Time Clock) würde zum einen Mehraufwand durch zusätzliches Hardwaredesign erfordern, zum anderen könnte es problematisch werden wenn sowohl der TinyNode als auch der RTC die Stromversorgung des Gumstix kontrollieren. Eine Lösung mit nur einem Master ist deshalb vorzuziehen.

Mit gewählter Lösung wurden die Verantwortungsbereiche der einzelnen Systemkomponenten so weit als möglich voneinander separiert: Die gesamte Kontrollintelligenz steckt im Gumstix; der TinyNode hat alleinigen Zugriff auf die Stromversorgung der energiehungrigen Subsysteme und schaltet diese auf externe Anweisung hin - entweder vom Gumstix oder durch das Dozer Netzwerk - ein oder aus.

4.4 Spezifikation

Nachdem die umzusetzende Variante nun feststeht ist es insbesondere in Hinblick auf die Anforderungen bezüglich Fehlertoleranz immens wichtig, das Systemverhalten sorgfältig zu spezifizieren und allfällige Fehlerszenarien schon am Modell zu evaluieren. Dieses Vorgehen minimiert die Möglichkeit von Fehlverhalten welches sich aus Unachtsamkeit schon in der Entwurfsphase einschleichen kann.

	Full power cycle mit TinyNode schedule	Full power cycle mit Gumstix schedule - TinyNode controlled	Full power cycle mit Gumstix schedule - RTC controlled
Beschreibung	Der Wakeup-Schedule wird per Dozer auf den Tinynode übertragen und dort verwaltet. Der Gumstix wird vom TinyNode gesteuert ein- und wieder ausgeschaltet. Zudem kann der Gumstix per Dozer-Beacon ausserplanmässig aufgeweckt werden.	Der Schedule wird via IP-Link auf den Gumstix übertragen und dort gespeichert. Nach Erledigen seiner Tasks handelt der Gumstix mit dem TinyNode den nächsten wakeup aus. Danach wird der Gumstix ganz ausgeschaltet und vom TinyNode nach dem ausgehandelten Timeout wieder gestartet. Zudem kann der Gumstix per Dozer-Beacon ausserplanmässig aufgeweckt werden.	Der Schedule wird via IP-Link auf den Gumstix übertragen und dort gespeichert. Nach Erledigen seiner Tasks programmiert der Gumstix einen externen Timer so dass er ihn aus- und nach einem timeout wieder einschaltet. Zudem kann der Gumstix per Dozer-Beacon ausserplanmässig aufgeweckt werden.
Energieeffizienz	**+** (Gumstix wird ganz ausgeschaltet)	**+** (Gumstix wird ganz ausgeschaltet)	**+** (Gumstix wird ganz ausgeschaltet)
Programmieraufwand	TinyNode: **++** Gumstix: **+**	TinyNode: **+** Gumstix: **+**	TinyNode: - Gumstix: **+**
Sonstiger Aufwand	**-**	**-**	**+** Es ist zusätzliche Hardware (Timer) oder ein Baseboard-Patch nötig um den Power via Timer auzuschalten
Robustheit: Dozer-Verbindung	- Wenn Dozer-Verbindung ausfällt kann der Schedule nicht mehr geändert werden.	- Dozer-Verbindung für den Betrieb nicht nötig (ausser manuelle wakeups).	- Dozer-Verbindung für Betrieb nicht nötig (ausser manuelle wakeups)
Robustheit: TinyNode Abhängigkeit	- Falls der TinyNode „stirbt" ist der Gumstix nicht mehr erreichbar.	- Falls der TinyNode „stirbt" ist der Gumstix nicht mehr erreichbar.	- TinyNode für Betrieb nicht nötig (ausser manuelle wakeups)

Abbildung 4-3
Designvarianten für das Kontroll- und Kommunikationssystem

4.4.1 Spezifikation in Worten

In Worten kann der grundlegende Mechanismus der Basisschicht wie folgt beschrieben werden:

Wenn die externe Stromversorgung eingeschaltet wird (entweder bei der Installation im Feld oder nach einem Versorgungsausfall), werden sowohl der Gumstix als auch der TinyNode eingeschaltet. Der Gumstix führt nach *jedem* Wakeup folgende weiteren Schritte durch:

- **Booten des Betriebssystems**: Der Gumstix bootet sein Betriebssystem (ein embedded Linux) ordungsgemäss.
- **Synchronisation des Service Windows**: Der Gumstix teilt dem TinyNode mit, wann das nächste "Daily Service Window" sein wird. Der TinyNode setzt einen entsprechenden internen Timer.
- **Grund für Wakeup erfahren**: Der Gumstix fordert beim TinyNode Auskunft über den Grund des Wakeups an. Die Antwort entscheidet über das weitere Vorgehen.
- **Service Mode**: Dieser Schritt wird ausgeführt falls entweder das tägliche Service Window ansteht oder der Gumstix via Wakeup-Beacon manuell aufgeweckt wurde. Der Gumstix wartet in diesem Fall eine gewisse Zeitspanne ab und fährt dann beim Punkt "Nächster Wakeup planen" fort.
- **Konfiguration aktualisieren**: Der Gumstix versucht eine Verbindung zum Datenbackend herzustellen um nachzufragen, ob ein Update seiner Konfiguration bereitliegt. Falls eine neue Konfiguration hinterlegt wurde, wird diese geladen und verarbeitet.
- **Geplanter Task ausführen**: Falls der Wakeup im Schedule geplant war, wird der geplante Task jetzt ausgeführt.
- **Nächster Wakeup planen**: Der Gumstix teilt dem TinyNode mit wann der nächste Wakeup für einen geplanten Task stattfinden muss. Der TinyNode setzt einen entsprechenden internen Timer.
- **Ausschalten der Stromversorgung planen**: Der Gumstix teilt dem TinyNode mit, wann er voraussichtlich heruntergefahren ist und daher die Stromversorgung ausgeschaltet werden kann. Der TinyNode setzt einen entsprechenden internen Timer.
- **Herunterfahren**: Der Gumstix fährt sein Betriebssystem ordnungsgemäss herunter.
- **Ausschalten der Stromversorgung**: Sobald der entsprechende Timer abgelaufen ist, schaltet der TinyNode die Stromversorgung des Gumstix ganz aus.
- **Auf den nächsten Wakeup warten**: Der TinyNode wartet auf das nächste Event welches einen Gumstix Wakeup triggert. Möglich sind:
  - Der nächste Task im Schedule ist fällig
  - Das Daily Service Window ist fällig
  - Der TinyNode empfängt ein Wakeup-Beacon über das Dozer Netzwerk

Sobald eines dieser drei Events auftritt wird die Stromversorgung des Gumstix wieder eingeschaltet und der ganze Ablauf beginnt erneut.

4.4.2 Spezifikation als Statechart

Das in 4.4.1 beschriebene Verhalten lässt sich mit dem Statechart Formalismus [8] visualisieren. TinyNode und Gumstix bilden dabei zwei parallele Subsysteme welche mittels globaler Events miteinander kommunizieren. Dieser Statechart ist in Abb 4-4 dargestellt.

Stehen bei einem Übergang mehrere auslösende Events, so muss *nur eines* davon auftreten damit der Übergang aktiviert wird. Heisst eine Action gleich wie ein Event, so bedeutet dies dass das gleichnamige Event gefeuert wird. Die States entsprechen der Beschreibung in 4.4.1. Es folgt eine Auflistung der definierten Events / Actions und ihrer Bedeutung:

power_on	Der TinyNode schaltet die Stromversorgung des Gumstix ein
power_off	Der TinyNode schaltet die Stromversorgung des Gumstix aus
gstx_ready	Das Betriebssystem ist fertig gebootet
service_sync	Der Gumstix sendet eine Nachricht an den TinyNode welche die Zeit bis zum nächsten Service Wakeup enthält.
service_wakeup	Der Gumstix erfährt vom TinyNode dass es sich um einen Service-Wakeup handelt (Service Window oder Beacon Wakeup)
task	Der Gumstix erfährt vom TinyNode dass es sich um einen im Schedule geplanten Wakeup handelt
no_conn	Der Gumstix kann keine Verbindung zum Backend herstellen
no_cfg	Es liegt keine neue Konfiguration im Backend bereit
cfg	Es liegt eine neue Konfiguration im Backend bereit
timeout	Die Zeit welche der Gumstix im Servicemodus wartend verharrt ist abgelaufen
init_scheduling	Der Gumstix sendet eine Nachricht an den TinyNode um den nächsten Schedule Wakeup sowie das Ausschalten der Stromversorgung zu planen
halted	Das Betriebssystem des Gumstix ist vollständig heruntergefahren
scheduled_timer	Der nächste Task im Schedule ist fällig
service_window	Das "Daily Service Window" ist fällig
power_on_beacon	Der TinyNode empfängt ein Beacon über das Dozer Netzwerk welches ihm befielt die Stromversorgung des Gumstix sofort einzuschalten
power_off_req	Die mit dem Gumstix vereinbarte Zeit ist abgelaufen und die Stromversorgung kann ausgeschaltet werden.
power_off_beacon	Der TinyNode empfängt ein Beacon über das Dozer Netzwerk welches ihm befielt die Stromversorgung des Gumstix sofort auszuschalten

Tabelle 4-2: Events und Actions im Statechart

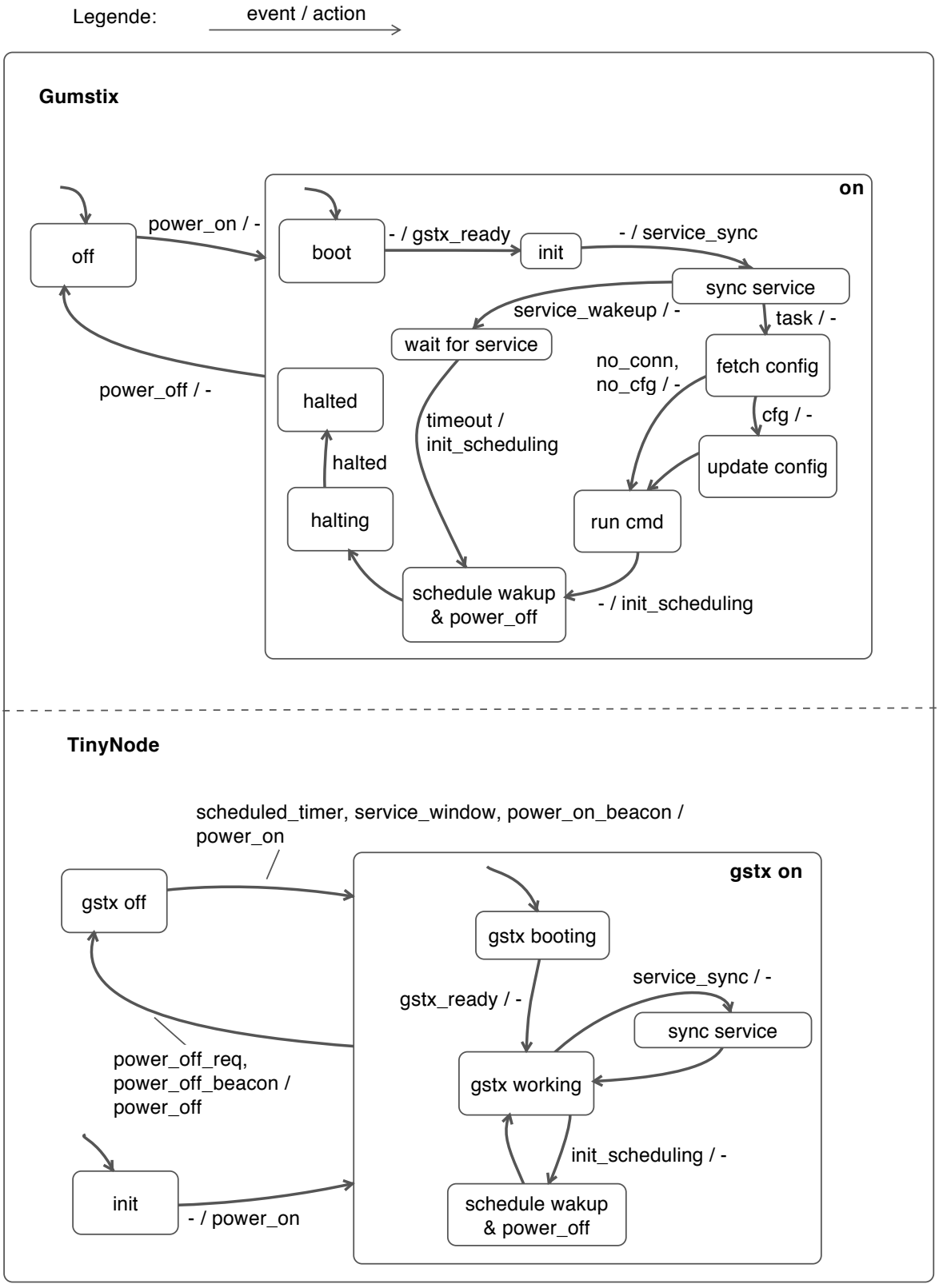


Abbildung 4-4
Statechart der Kommunikations- und Kontrollschicht

4.4.3 Analyse der Fehlertoleranz

Da die Systeme in PermaSense unter schwierigsten Umweltbedingungen zuverlässig arbeiten müssen, ist es von enormer Wichtigkeit Fehlfunktionen nicht stiefmütterlich zu behandeln sondern im Designprozess gebührend zu beachten. Gemäss Siewiorek und Swarz [13] können fehlertolerante Systeme anhand ihrer **Verfügbarkeit** und ihrer **Verlässlichkeit** bewertet werden, wobei sich Verfügbarkeit auf den Bruchteil der Zeit bezieht in welcher ein System tatsächlich seine Aufgabe erfüllt, die Verlässlichkeit hingegen fokussiert sich darauf *wie lange* ein System korrekt funktioniert. Gemäss [13] ist eine hohe Verlässlichkeit besonders dann wichtig, wenn das System

- im Falle eines Ausfalls nicht repariert werden kann (z.B. Satelliten)
- eine kritische Funktion ausführt und deshalb auch ein reparabler Ausfall nicht toleriert werden kann (z.B. Flugzeugsteuerungen)
- zwar prinzipiell ausfallen dürfte, eine Reparatur aber unverhältnismässig teuer zu stehen käme

Der letzte Punkt hat bei PermaSense durchaus seine Richtigkeit; im Moment sind zwar noch verhältnismässig viele Wartungsarbeiten auf der Site nötig, die Langzeitperspektive des Projekts ist aber eine ungewartete Systemlebensdauer von drei Jahren und somit sollten alle involvierten Systeme über mindestens diesen Zeitraum zuverlässig funktionieren.

Die in den vorigen Abschnitten vorgestellte Spezifikation ist das Resultat vieler Diskussionen, Vorschläge und Erfahrungen und hat zum Ziel, Ausfälle aufgrund Fehlfunktion des spezifizierten System selbst oder eines anderen Subsystems so weit wie möglich zu minimieren. Insbesondere wurden folgende Fehlerszenarien - welche in PermaSense bis anhin relativ häufig beobachtet wurden - berücksichtigt:

- **Verbindungsunterbruch zum Dozer Netzwerk**: Das System arbeitet beinahe unbeeinträchtigt weiter, einzig die Möglichkeit der manuellen Steuerung via Beacon-Nachrichten fällt weg.
- **Verbindungsunterbruch zum Datenbackend** (aufgrund von Netzwerkproblemen oder einem Ausfall des Backendservers): Das System arbeitet mit der aktuellen Konfiguration und dem aktuellen Schedule weiter und speichert die Daten lokal. Bei der nächsten erfolgreichen Verbindung wird versucht die aufbewahrten Daten nach und nach ins Backend zu transportieren.
- **Ausfall der Stromversorgung** (z.B. aufgrund mangelnder Sonneneinstrahlung): Nach einem Stromausfall startet das System in die "initiale Runde" und bringt sich selbst in einen definierten Zustand. Der Schedule wird wieder normal aufgenommen, jedoch werden ausgefallene Tasks nicht nachgeholt.
- **Der TinyNode wird geresettet** (z.B. durch einen Watchdog): Auch nach einem Reset wird die "initiale Runde" durchgeführt und das System befindet sich in einem definierten Zustand.

Als besonders hilfreich erweist sich die Forderung dass das System sich nach jedem "fatalen" Ausfall in einen definierten Zustand bringt - Gumstix und TinyNode

synchronisieren sich und können danach nach gegebenem Schema zusammen funktionieren.

All diese Eigenschaften sind letztlich Massnahmen welche die **Verlässlichkeit** des entworfenen Systems - also die Wahrscheinlichkeit dass es möglichst lange in einem konkreten Experiment überlebt und funktioniert - steigern sollen. Dies geschieht primär durch eine möglichst starke Entkoppelung der internen Funktion von anderen Subsystemen. Selbstverständlich bringt z.B. ein Verbindungsunterbruch zum Datenbackend eine Einschränkung der Systemüberwachbarkeit und der Datenverfügbarkeit mit sich, sie hat aber keinen Einfluss auf die ausgeführten Messungen an sich. Konkrete Aussagen über die effektive Verfügbar- und Verlässlichkeit der realisierten Implementierung sind nur durch Beobachtung über eine längeren Zeitraum hinweg möglich und sprengen damit den Rahmen einer Semesterarbeit.

4.5 Implementierung

Gemäss obiger Spezifikation wurde schliesslich auf der gegebenen Hardware (der *PermaSense Basestation v2*) die erforderliche Funktionalität implementiert. Die dazu entwickelte Software ist in mehrere Teile aufgeteilt:

- **Die Steuersoftware auf dem Gumstix Embedded PC**: Hier wurde auf der bereits existierenden Software *PSBacklog* aufgebaut, welche einen zuverlässigen Kommunikationskanal von einer PermaSense Basestation zum Backend implementiert. Details dazu finden sich in 4.5.1.
- **Die Steuersoftware auf dem TinyNode Sensorknoten**: Diese wird in 4.5.2 behandelt.
- **Die Backendsoftware zur Rekonfiguration des Gumstix**: Bei PermaSense wird im Backend die GSN Middleware [2] verwendet um Daten mit der Messstelle auszutauschen. Diese Software wurde wie in 4.5.3 beschrieben um die nötige Funktionalität erweitert.

Grundsätzlich kann festgestellt werden, dass die Verwendung von drei verschiedenen Plattformen zur Lösung des Problems einige Herausforderungen mit sich bringt. Die verschiedenen plattformabhängigen Eigenheiten (wie z.B. Datenformate) bedürfen grosser Aufmerksamkeit damit sich keine (meist schwer aufspürbaren) Fehler einschleichen. Umso wichtiger ist daher eine genaue Spezifikation der verwendeten Kommunikationsmechanismen und Datenformate.

4.5.1 Implementierung auf dem Gumstix Embedded PC

Die Implementierung der Software auf dem Gumstix Embedded PC baut auf der bereits bestehenden PSBacklog-Software auf, welche in Python geschrieben ist. PSBacklog ist ursprünglich für den Einsatz auf einer "normalen" PermaSense Basestation geschrieben und sorgt für einen zuverlässigen Kommunikationskanal zwischen der Messstelle und dem Backend. Alle Messdaten welche über das Dozer Netzwerk zur Basestation gelangen, werden über diesen Kanal (eine TCP/IP Verbindung) an den GSN Server im Backend weitergeleitet und dort gespeichert. PSBacklog bietet ein flexibles Plugin-Interface, mit welchem auch weitere Daten

wie z.B. Webcambilder oder Messdaten der direkt an die Basestation angeschlossenen Wetterstation weitergegeben werden können. Da diese Plugin-Struktur sehr flexibel ist und PSBacklog bereits produktiv und sehr zuverlässig arbeitet, lag der Gedanke nahe die neu zu implementierende Funktionalität - im Folgenden "Reverse Mode" genannt - als PSBacklog-Plugin zu umzusetzen.

Das entwickelte `ReverseModePlugin` misst selbst keine Daten, es erlaubt vielmehr die zeitlich geplante und parametrisierte Instantiierung separater Messplugins welche dann ihre Daten über den PSBacklog-Link ins Backend übertragen können. Der eigentliche Task - die *run_cmd*-Phase aus der Statechart-Spezifikation in 4.4.2 - wird also von spezialisierten Plugins ausgeführt. Alle anderen spezifizierten Aufgaben - Verwaltung des Schedules, Kommunikation mit dem TinyNode etc. - übernimmt das `ReverseModePlugin`.

4.5.1.1 Scheduleformat

Das "Herzstück" des `ReverseModePlugins` ist der Schedule - der Zeitplan also mit welchem alle Messaktivitäten der Station gesteuert werden. Der Schedule welcher auf die Basestation transferiert wird ist eine einfache Textdatei deren Inhalt dem in diesem Abschnitt beschriebenen Format entspricht. Jede Zeile des Schedules beschreibt einen "Task" und besteht aus folgenden Informationen:

- **Zeit**: Die Uhrzeit zu welcher dieser Task ausgeführt werden soll
- **Plugins**: Die Messplugins, welche für diesen Task instantiiert werden sollen
- **Plugin-Parameter**: Für jedes Plugin können zusätzliche Parameter spezifiziert werden, welche bei der Instantiierung an das Plugin weitergegeben werden. Dies erlaubt es dem Benutzer, je nach Tageszeit andere Messmodi einzusetzen.

Konkret folgt eine Schedulezeile folgendem Format:

```
hh:mm Plugin1:param1=value1,param2=value2,param3=value3 Plugin2:param1=value1,...
```

Dabei gilt:

- `hh:mm`: Zeitpunkt in Stunden und Minuten
- `PluginX`: Name eines zu instantiierenden Plugins
- `paramX`: Parametername welcher dem Plugin übergeben wird
- `valueX`: Wert für den vorangehenden Parameternamen

Konkret könnte ein kurzer Schedule z.B. so aussehen:

```
08:00 GPSPlugin:interval=5,measTime=60 WebCamPlugin
10:20 WebCamPlugin
11:15 WebCamPlugin GPSPlugin:measTime=120,interval=10
```

Damit starten um 08:00 und um 11:15 sowohl das WebCamPlugin als auch das GPSPlugin, um 10:20 Uhr wird nur das WebCamPlugin instantiiert. Beim ersten

Wakeup misst das GPSPlugin während 60 Sekunden im 5 Sekundentakt, beim zweiten Wakeup während 120 Sekunden im 10 Sekundentakt. Es ist zu beachten dass die Reihenfolge der Plugins bzw. Parameter keine semantische Bedeutung hat.

Die im Schedule übergebenen Parameter sind im aufgerufenen Plugin in der `inititialize` Methode als Elemente des Arguments `params` verfügbar; so übernimmt das `GPSPlugin` z.B. seine Parameter mit folgenden Zeilen:

```
self._interval = int(params['interval']) # The measurement interval in seconds
self._measTime = int(params['measTime']) # The measurement time in seconds
```

4.5.1.2 Kommunikationsprotokoll für Nachrichten an den TinyNode

Für die Kommunikation mit dem TinyNode wurde ein einfaches Kommunikationsprotokoll spezifiziert. Grundsätzlich wird die Kommunikation *immer* vom Gumstix initiiert. Der Gumstix schickt eine TinyOS Message mit TinyOS Message Type `0x21` an den TinyNode. Der Payload der Nachricht besteht aus *1 Byte Message ID* und einem *4 Byte Parameter*. Der TinyNode antwortet mit exakt der selben Nachricht für Message Codes 2, 3 und 4 sowie mit einem entsprechend gesetzten Parameter für Message Codes 1 und 5. Tabelle 4-3 zeigt die implementierten Message Codes, Tabelle 4-4 die verschiedenen Wakeup Reasons wie sie in der Nachricht "Aufweckgrund" verwendet werden.

Name	Beschreibung	Parameter	ID
Aufweckgrund	Der Gumstix fragt mit diesem Kommando nach dem Grund für den Wakeup. Die möglichen Werte sind in Tabelle 4-4 beschrieben	Der TinyNode gibt im Parameterfeld den Aufweckcode gemäss Tabelle 4-4 zurück	1
Service Window	Mit dieser Nachricht plant der Gumstix das nächste Service Window	Die Anzahl Sekunden bis zum nächsten Service Window	2
Nächster Wakeup	Mit dieser Nachricht plant der Gumstix den nächsten Duty Wakeup	Die Anzahl Sekunden bis zum nächsten Wakeup im Schedule	3
Shutdown	Mit dieser Nachricht befiehlt der Gumstix dem TinyNode die Stromversorgung zu unterbrechen	Die Anzahl Sekunden bis zum Unterbruch der Stromversorgung	4
Netzstatus	Mit dieser Nachricht erkundigt sich der Gumstix nach dem Dozer-Verbindungsstatus des TinyNode	Der TinyNode gibt 1 zurück falls er mit Dozer verbunden ist, ansonsten 0	5

Tabelle 4-3: Nachrichten zur Kommunikation mit dem TinyNode

Code	Beschreibung
0	Scheduled - der Wakeup war also im Schedule geplant
1	Service - das Daily Service Window ist fällig
2	Beacon - der TinyNode hat einen Wakeup-Beacon empfangen (siehe Tabelle 4-5)
3	Reboot - der TinyNode wurde neu gestartet, z.B. aufgrund eines Stromausfalls

Tabelle 4-4: Mögliche Wakeup Reasons

4.5.1.3 Konfigurationsparameter

Wie für alle anderen PSBacklog Plugins werden die Parameter für das `ReverseModePlugins` in der Datei `plugins.cfg` festgelegt. Das Plugin benötigt folgende Konfigurationsparameter:

- **`reverse_mode_schedule`**: Der vollständige Pfad unter welchem die Schedule-Datei abgelegt werden soll.
- **`reverse_mode_connection_wait`**: Die Anzahl Sekunden welche das `ReverseModePlugin` auf eine Verbindung vom GSN-Server warten soll. Verbindet sich in dieser Zeitspanne der GSN-Server nicht, wird von einem Verbindungsunterbruch ausgegangen und der Gumstix fährt fort ohne die aktuelle Konfiguration zu aktualisieren.
- **`reverse_mode_configuration_wait`**: Die Anzahl Sekunden welche das `ReverseModePlugin` auf neue Konfigurationswerte warten soll bevor nachdem es die entsprechende Anfrage an den GSN-Server abgeschickt hat. Nach Ablauf dieser Zeit wird davon ausgegangen dass keine neue Konfiguration vorliegt und das Programm wird fortgesetzt.
- **`reverse_mode_service_wakeup`**: Die Uhrzeit des Daily Service Window im Format `hh:mm`.
- **`reverse_mode_service_uptime`**: Die Anzahl Sekunden welche der Gumstix während eines Service Windows im Wartezustand bleiben soll. In diesem Zeitfenster kann sich ein Benutzer aus dem Backend z.B. per SSH einloggen.
- **`reverse_mode_shutdown_offset`**: Die Anzahl Sekunden welche der Gumstix Embedded PC benötigt um nach dem Senden der Shutdown-Nachricht an den TinyNode das Betriebssystem herunterzufahren. Dieser Offset wird in der Shutdown-Message an den TinyNode weitergegeben. Dieser wird - ohne Rücksicht auf Verluste - nach dieser Zeitspanne die Stromversorgung ausschalten. Daher sollte dieser Wert genug gross gewählt werden. Mit der zum Zeitpunkt dieser Dokumentation gültigen Konfiguration hat sich ein Wert von 20s gut bewährt.

Es ist zu beachten, dass die Plugins welche vom `ReverseModePlugin` instantiiert werden sollen **nicht** in `plugins.cfg` unter `[plugins]` aufgeführt werden dürfen. Alle Plugins die an dieser Stelle erscheinen werden von PSBacklog bei jedem Start und unabhängig vom Schedule initialisiert. Jedoch können Plugin-Parameter welche für jede Instantiierung gleich bleiben in `plugins.cfg` abgelegt werden und müssen nicht per Schedule jedes mal separat spezifiziert werden. Der Programmierer eines neuen Plugins hat also die Wahl ob ein Plugin-Parameter pro Installation (in `plugins.cfg`) oder pro Aufruf (im Schedule) spezifiziert werden muss

Für weitere Details zur Implementierung des `ReversemodePlugins` sei auf die ausführlichen Kommentare im Quellcode verwiesen.

4.5.1.4 Einbettung in das Betriebssystem

Der PSBacklog-Prozess hat auf einem System im Reverse Mode eine besondere Bedeutung: Er muss nicht nur automatisch beim Start des Betriebssystem geladen werden, sondern muss das System nach getaner Arbeit auch wieder herunterfahren. Zu diesem Zweck wurde das bestehende Skript `psbacklog.sh` entsprechend modifiziert:

```
#!/bin/sh

PIDFILE="/var/run/psbacklog.pid"

python -O /usr/lib/python2.5/psbacklog/BackLogMain.py -c /etc/plugins.cfg -l /m$

wait

if [ -f "/etc/reversemode.conf" ]; then
  echo "Halting system for we're in reverse mode..."
  /sbin/halt
fi
```

Nach der Beendigung des PSBacklog Hauptprogramms wird also nach der Datei `/etc/reversemode.conf` gesucht. Falls diese existiert, so wird das System heruntergefahren wie es für den Reverse Mode spezifiziert ist.

Manchmal ist es nötig PSBacklog im laufenden Betrieb zu stoppen, z.B. um Wartungsarbeiten via SSH auszuführen. Dazu wird am einfachsten das PSBacklog-Initscript verwendet werden:

```
root@tik-gumstixXX> /etc/init.d/psbacklog stop
```

Mit

```
root@tik-gumstixXX> /etc/init.d/psbacklog start
```

wird PSBacklog nach getaner Arbeit wieder gestartet und läuft normal weiter. Wenn der Output von PSBacklog überwacht werden soll, so kann sich nach erfolgtem SSH-Login per

```
root@tik-gumstixXX> screen -x psbacklog
```

auf den entsprechenden Screen verbunden werden.

4.5.1.5 Handbetrieb

Der im Anforderungsprofil geforderte Handbetrieb ist mit der vorgestellten Implementierung durch einen Reset des TinyNode zu erreichen - damit wird auch der Gumstix gebootet. Falls z.B. für einen Systemtest bei der Installation eines neuen Sensors weitere Aktionen nötig sind, so müsste das `ReverseModePlugin` um einen dedizierten manuellen Modus mit entsprechender Funktionalität erweitert werden.

4.5.2 Implementierung auf dem TinyNode

Die Implementierung auf dem TinyNode gestaltet sich aufgrund des kleinen "Verantwortungsbereichs" relativ einfach. Da die eigentliche Aufgabe des TinyNode nur darin besteht im richtigen Moment die Stromversorgung des Gumstix ein- und auszuschalten, beschränkt sich die Programmlogik auf das in 4.5.1.2 beschriebene Kommunikationsprotokoll und der Verwaltung mehrerer Timer sowie einiger Statusvariablen.

Konkret müssen 3 Timer verwaltet werden:

- Der **Daily Service Wakeup Timer (`ServiceTimer`)** zählt die Sekunden bis zum nächsten Service Window. Wird dieser Timer gefeuert so wird der Gumstix eingeschaltet und in den Service Mode versetzt.
- Der **Schedule Timer (`ScheduleTimer`)** zählt die Sekunden bis zum nächsten geplanten Task. Beim Feuern dieses Timers wird der Gumstix eingeschaltet und in den Duty Mode versetzt - er führt also den geplanten Task aus.
- Der **Shutdown Timer (`ShutdownTimer`)** schliesslich zählt die Sekunden bis der Gumstix wieder ausgeschaltet wird. Er wird vom Gumstix gesetzt bevor dieser sein Betriebssystem herunterfährt und beträgt daher im Normalfall nur wenige Sekunden (ca. 20s). Sobald dieser Timer gefeuert wird schaltet der TinyNode die Stromversorgung des Gumstix aus.

Zusätzlich dazu gibt es zwei Statusvariablen:

- **Gumstix Status (`m_state`)**: In dieser Variable wird der (vermutete) aktuelle Zustand des Gumstix gespeichert. Mögliche Werte sind *schlafend (`GUMSTIX_SLEEP`)*, *startend (`GUMSTIX_STARTING`)*, *laufend (`GUMSTIX_STARTED`)* und *stoppend (`GUMSTIX_STOPPING`)*.
- **Aufweckgrund (`wakeup_reason`)**: Darin wird der Grund für das aktuelle Einschalten des Gumstix gespeichert. Siehe dazu Tabelle 4-4.

Um den Gumstix auch vom Backend aus ein- und ausschalten zu können wurden zwei Dozer Beacon Nachrichten definiert, mit welchen diese Aktionen aus der Ferne initiiert werden können:

4.5.3 Implementierung im Backend

Die Integration ins Datenbackend besteht im Wesentlichen aus drei Teilen:

- ein Plugin für den `BackLogWrapper` namens `ReverseModeCommandPlugin`

Nachricht	Beschreibung
0A 00 FF FF 50 7D **01** E0 FF FF 00 00 00 00 00 00	Gumstix einschalten
0A 00 FF FF 50 7D **03** E0 FF FF 00 00 00 00 00 00	Gumstix ausschalten

Tabelle 4-5: Die Beacon Nachrichten um den Gumstix ein- und auszuschalten

- ein virtueller Sensor in welchem das Webformular für den Schedule-Upload definiert wird.

4.5.3.1 Das ReverseModeCommandPlugin

Das `ReverseModeCommandPlugin` dient als Zwischenspeicher für den Fall dass der Benutzer einen neuen Schedule an unseren Sensor schicken möchte. Das Plugin nimmt einen neuen Schedule ggf. vom virtuellen Sensor entgegen und leitet diesen an die Sensorstation weiter sobald dieser eine entsprechende Nachricht an das Backend schickt (eine `BackLogMessage` mit Message Code `40`). Nachdem der Schedule an den Sensor übermittelt wurde, wird er samt aktuellem Timestamp in der GSN Datenbank gespeichert. So kann nachträglich nachvollzogen werden wann ein neuer Schedule an eine Sensorstation übermittelt wurde.

So lange der Schedule noch nicht an die Sensorstation übermittelt werden konnte, lagert er *nichtpersistent* im Arbeitsspeichers des GSN Servers. Erst nach erfolgreicher Übermittlung wird er in die Datenbank geschrieben. Wird also der GSN Server neu gestartet, gehen alle ungesendeten Schedules verloren und müssen neu im System hinterlegt werden.

Für weitere Implementierungsdetails zum `ReverseModeCommandPlugin` sei auf die Kommentare im Quellcode verwiesen.

4.5.3.2 Der virtuelle Sensor

Für die Übergabe des Schedules an das `ReverseModeCommandPlugin` muss pro Sensorstation ein einfacher virtueller Sensor spezifiziert werden. Dabei kann von einem simplen virtuellen Sensor in GSN ausgegangen werden; im Wesentlichen müssen nur folgende Punkte angepasst werden:

- das `class-name` Elements muss `gsn.vsensor.BridgeVirtualSensorBothDirections` beinhalten
- das `processing-class` Element muss ein `web-input` Element beinhalten damit in der GSN Webanwendung ein neuer Schedule in Form einer Textdatei deponiert werden kann
- der data-stream muss im `address` Element die `predicate` Elemente "deployment" und "plugin-classname" beinhalten, wobei der Wert von letzterem `gsn.wrappers.backlog.plugins.ReverseModeCommandPlugin` sein sollte.

Listing 4.1 verdeutlicht diese Aspekte:

Listing 4.1
XML Struktur des virtuellen Sensors für den Reverse Mode

```
<virtual-sensor name="...">
  <processing-class>
    <class-name>gsn.vsensor.BridgeVirtualSensorBothDirections</class-name>
    ...
    <web-input>
      <command name="reverse_mode_command">
        <field name="schedule" type="*binary:1mb">schedule</field>
      </command>
    </web-input>
    ...
  </processing-class>
  ...
  <streams>
    ...
    <stream name="data">
      <source alias="source" storage-size="1" sampling-rate="1">
        <address wrapper="ps-backlog">
          <predicate key="deployment">IP_OR_DNS_OF_SENSOR</predicate>
          <predicate key="plugin-classname">
            gsn.wrappers.backlog.plugins.ReverseModeCommandPlugin
          </predicate>
        </address>
        ...
      </source>
      ...
    </stream>
    ...
  </streams>
</virtual-sensor>
```

5

Das GPS Subsystem

Auf das im letzten Kapitel vorgestellten Kommunikations- und Kontrollsystem soll nun das eigentliche Modul zur Messung von GPS Daten aufgesetzt werden. Dazu wird zuerst die verwendete Hardware und die damit verbundenen Möglichkeiten vorgestellt, in einem weiteren Schritt werden die Möglichkeiten des GPS Subsystems kurz spezifiziert. Das Kapitel schliesst mit einer Dokumentation der Implementierung.

5.1 Der GPS Empfänger: u-blox ANTARIS 4

Schon zu Beginn der Semesterarbeit war klar, dass als GPS Empfänger wohl ein Chip der Schweizer Firma u-blox[1] eingesetzt werden würde. Da deren neuste Generation von Chips (die LEA 5 Serie) von einem Fehler in der Trägerphasenmessung betroffen ist, wurde für diese Arbeit die etwas ältere Generation der ANTARIS 4 Chips verwendet.

Um die für das Postprocessing mit der Bernese GPS Software benötigten RAW-Daten - also die Pseudodistanz, die Trägerphase und die Dopplerverschiebung - messen zu können, muss das spezielle Timing-Modul - in unserem Fall das AEK 4T - verwendet werden. Nur mit diesen etwas teureren Timing-Modulen ist die Ausgabe der RAW-Daten an einen der externen seriellen Ports möglich. u-blox stellt diese AEK 4T Empfänger als Evaluation-Kit zur Verfügung, welches ein GPS-Empfängermodul auf einer PCB-Platine samt USB- und Antennenanschluss beinhaltet. Auch eine einfache Patch-Antenne wird mitgeliefert.

Dem Evaluation Kit liegt auch die Software u-center bei, welche für die Konfiguration des Empfängers eine graphische Benutzeroberfläche bietet und sich gut eignet um erste Erfahrungen mit dem Gerät zu sammeln.

Für die Kommunikation mit dem ANTARIS 4 Chip können zwei verschiedene Protokolle verwendet werden:

[1] http://www.u-blox.com

Abbildung 5-1
u-blox AEK 4T Evaluation Kit

- Das **NMEA 0183** Protokoll, welches ursprünglich von der National Marine Electronics Association[2] als Kommunikationsprotokoll zwischen nautischen Geräten wie Echoloten, Anemometer, Autopilot und eben auch GPS Empfängern spezifiziert wurde. Dieses Protokoll ist ASCII basiert und auch für einen Menschen relativ gut lesbar. Allerdings bietet das NMEA-Protokoll keinen vollen Zugriff auf alle Features des ANTARIS 4.
- Das u-blox eigene **UBX** Protokoll wurde speziell für die Kommunikation mit allen u-blox Empfängern konzipiert. Die mit UBX ausgetauschten Nachrichten haben ein binäres Format und sind dadurch sehr kompakt, daher aber für einen Menschen nicht mehr so einfach lesbar wie NMEA. Doch nur mit UBX kann der volle Funktionsumfang des ANATRIS 4 Chips genutzt werden; unter anderem sind die für unsere Anwendung besonders interessanten RAW-Daten nur per UBX abrufbar.

Alle verfügbaren Nachrichten und Betriebsmodi des ANTARIS 4 sowie die genaue Spezifikation des UBX Protokolls sind in der frei erhältlichen ANTARIS Protocol Specification [1] enthalten.

5.2 Spezifikation des GPS Subsystems

Das GPS Subsystem soll in der Lage sein, in Bezug auf Messdauer und Samplingrate frei parametrisierbare GPS Messungen durchzuführen und die gemessenen Rohdaten an das Datenbackend weiterzugeben. Dazu soll es auf das bereits bestehende Kommunikations- und Kontrollsystem aufbauen, welches eine flexible zeitliche Steuerung der Messungen sowie einen verlässlichen Datentransport vom Sensor ins Backend realisiert. Folgende Daten sind in jedem Fall zu messen und zu speichern:

- Der genaue Zeitpunkt der Messung
- Die zu diesem Zeitpunkt sichtbaren Satelliten

[2] http://www.nmea.org

- Die Pseudodistanz zu jedem Satelliten
- Die Trägerphase jedes Satellitensignals
- Die Dopplerverschiebung jedes Satellitensignals
- Das Signal-Rausch-Verhältnis jedes Satellitensignals

Weiter soll eine Möglichkeit zur Verfügung stehen, die gemessenen Daten im RINEX-Format [7] zu exportieren, um den Austausch von Daten und insbesondere das Einspeisen in die Bernese GPS Software zu ermöglichen.

5.3 Implementierung

Da die Implementierung auf dem in Kapitel 4 vorgestellten Kommunikations- und Kontrollsystem aufbauen soll, ist deren grundlegende Struktur schnell klar: Der sensorseitige Teil muss als PSBacklog-Plugin umgesetzt werden, der backendseitige Teil wird als virtueller Sensor in GSN implementiert. Einzig die Umsetzung des Datenexports ins RINEX-Format ist nicht von vornherein klar; er wurde schliesslich als PHP-Skript realisiert, welches im CSV-Format vorliegende Messdaten ins RINEX-Format konvertiert.

5.3.1 Implementierung auf dem Sensor: Das PSBacklog-Plugin

Das in Python geschriebene PSBacklog-Plugin `GPSPlugin` erlaubt es, für beliebige Zeitspannen und mit beliebiger Samplingrate GPS Messungen durchzuführen. Die in Tabelle 5-1 beschriebenen Parameter werden im Schedule des `ReverseModePlugins` übergeben.

Parametername	Einheit	Beschreibung
`measTime`	[s]	Die Gesamtzeit der Messperiode
`interval`	[s]	Das Samplingintervall

Tabelle 5-1: Parameter des `GPSPlugins`

Die Schedulezeile

```
09:00 GPSPlugin:measTime=300,interval=5
```

veranlasst also das `GPSPlugin` dazu, um 9:00 Uhr während 5min im 5 Sekunden Intervall zu messen. Die gemessenen Daten werden direkt via PSBacklog an das Backend übertragen.

Der ANTARIS 4 Empfänger wird einfach an einen der USB-Ports am PermaSense Baseboard v2 angeschlossen. Damit das `GPSPlugin` die benötigten Daten vom Empfänger abfragen kann muss dieser zuerst entsprechend konfiguriert werden; mehr dazu findet sich im Kapitel 6. Das `GPSPlugin` benötigt nur einen Konfigurationsparameter im allgemeinen Konfigurationsfile `plugins.cfg` im `[options]` Abschnitt:

```
gps_device = /dev/ttyACM0
```

Diese Zeile kann im Normalfall 1:1 übernommen werden, es sei denn dass weitere ACM-Geräte an der Station betrieben werden und darum die Device-Adresse des GPS-Empfängers ändert.

Das `GPSPlugin` implementiert die nötige Funktionalität um mit dem Empfänger über das UBX Protokoll Nachrichten auszutauschen. Während der Initialisierung wird der Empfänger auf das gewünschte Samplingintervall konfiguriert - dies ist notwendig da die Samplingzeitpunkte der verschiedenen Empfänger im Feld für eine erfolgreiche Auswertung möglichst exakt übereinstimmen müssen. Sobald die `run` Methode des Plugins aufgerufen wird, wird für die Dauer der gesamten Messperiode im konfigurierten Intervall eine UBX `RMX-RAW` Nachricht vom Empfänger gepollt - das heisst es wird eine leere `RMX-RAW` Nachricht zum Empfänger gesendet und dieser antwortet mit den zuletzt gemessenen Rohdaten. Somit wird jegliche Kommunikation mit dem Empfänger vom `GPSPlugin` initiiert. Die `RMX-RAW` Nachricht enthält alle Informationen welche in der Spezifikation im Abschnitt 5.2 gefordert wurden und noch einige dazu:

- Die genaue GPS Zeit[3] in der Form GPS-Woche / Millisekunden seit dem Start der GPS Woche
- Pro Satellit
  - Die Satellitennummer
  - Die Trägerphase
  - Die Pseudodistanz
  - Die Dopplerverschiebung
  - Ein Qualitätsindikator für die Messung
  - Signalstärke
  - der so genannte LLI welcher im RINEX-Format benötigt wird

Für die genaue Spezifikation einer `RMX-RAW` Nachricht sei auf die ANTARIS Protocol Specification [1] verwiesen.

Um das Handling der Daten möglichst einfach zu halten, werden die Datensätze *pro Satellit getrennt* an das Backend übertragen - es wurde also pro Satellit eine PSBacklog-Nachricht generiert und damit ein Datensatz in der Backend Datenbank.

Wenn die Messperiode vorbei ist beendet sich das `GPSPlugin` und übergibt somit die Kontrolle wieder an das `ReverseModePlugin`.

5.3.2 Implementierung im Backend

Die Implementierung im Backend beschränkt sich auf einen virtuellen Sensor und ein Plugin für den `PSBacklogWrapper`. In diesem Plugin - `GPSPlugin` genannt - werden

[3] Die GPS Zeit unterscheidet sich von der UTC um einige Sekunden da deren Schaltsekunden nicht mitberücksichtigt werden. Mehr dazu unter `http://de.wikipedia.org/wiki/GPS-Zeit`

primär die binär übermittelten Daten in der `packetReceived` Methode auf die entsprechenden Java-Datentypen abgebildet um sie dann in der Datenbank speichern zu können. Der virtuelle Sensor für ein GPS-Plugin gestaltet sich wie folgt:

Listing 5.1
XML Struktur des virtuellen Sensors für einen GPS Sensor

```
<virtual-sensor name="...">
  <processing-class>
    <class-name>gsn.vsensor.BridgeVirtualSensorBothDirections</class-name>
    ...
    <output-structure>
      <field name="TIMESTAMP" type="BIGINT"/>
      <field name="gpsTime" type="INTEGER"/>
      <field name="gpsWeek" type="SMALLINT"/>
      <field name="carrierPhase" type="DOUBLE"/>
      <field name="pseudorange" type="DOUBLE"/>
      <field name="doppler" type="DOUBLE"/>
      <field name="spaceVehicle" type="SMALLINT"/>
      <field name="measurementQuality" type="SMALLINT"/>
      <field name="signalStrength" type="SMALLINT"/>
      <field name="lli" type="SMALLINT"/>
    </output-structure>
    ...
  </processing-class>
  ...
  <streams>
    ...
    <stream name="data">
      <source alias="source" storage-size="1" sampling-rate="1">
        <address wrapper="ps-backlog">
          <predicate key="deployment">IP_OR_DNS_OF_SENSOR</predicate>
          <predicate key="plugin-classname">
            gsn.wrappers.backlog.plugins.GPSPlugin
          </predicate>
        </address>
        ...
      </source>
      ...
    </stream>
    ...
  </streams>
</virtual-sensor>
```

Es ist wichtig dass die Datentypen unter `<output-structure>` exakt mit der Definition im `GPSPlugin` übereinstimmen.

5.3.3 Datenkonversion in RINEX: csv2rinex.php

Da der Datenimport in die Bernese GPS Software nur über das RINEX Datenformat möglich ist, musste eine Möglichkeit implementiert werden die gemessenen Daten in dieses Format zu exportieren. Zu diesem Zweck wurde ein einfaches PHP-Skript geschrieben. Als Eingabe benötigt das Skript eine Datei welche die gemessenen Datensätze im CSV Format beinhaltet - dies ist mit den meisten Datenbankadministrationstools sehr einfach zu bewerkstelligen. Die CSV-Datei darf keinen Header besitzen sondern muss in der ersten Zeile sogleich mit den Daten beginnen. Das Format einer solchen Zeile ist

```
GPSTIME,GPSWEEK,CARRIERPHASE,PSEUDORANGE,DOPPLER,SPACEVEHICLE,MEASUREMENTQUALITY,SIGNALSTRENGTH,LLI
```

Das PHP Skript `csv2rinex.php` konvertiert die Daten daraufhin ins **RINEX** 2.11 Format [7], welches im Moment das gängigste Format zum Austausch von GPS Daten ist und im Gegensatz zu RINEX 3.0 von den meisten Softwarepaketen - so auch von der Bernese GPS Software - unterstützt wird.

`csv2rinex.php` wird auf der Kommandozeile ausgeführt und erwartet 3 Parameter:

```
> php csv2rinex.php <pfad_zur_csv_datei> <empfaengername> <empfaengernummer>
```

wobei `<empfaengername>` und `<empfaengernummer>` zur Empfängeridentifikation dienen und im Prinzip frei gewählt werden können.

Die aktuelle Version von `csv2rinex.php` kann die dreistelligen Satellitennummern der EGNOS Satelliten nicht korrekt behandeln; dies hauptsächlich weil diese in RINEX Version 2.11 anders als GPS Satelliten behandelt werden. Die gemessenen EGNOS Datensätze müssen also beim Export herausgefiltert werden, am einfachsten geschieht dies wenn nur Datensätze mit Satellitennummern ≤ 32 betrachtet werden.

Zu `csv2rinex.php` muss gesagt werden dass aus zeitlichen Gründen nicht sehr viel Entwicklungs- und Testzeit investiert werden konnte und es sich daher eher um eine Alphaversion als um ein fertiges Tool handelt. Zudem wäre für einen produktiven Einsatz eine Implementierung in Java wohl wünschenswert damit die Konversion der Daten direkt im GSN Workflow abgewickelt werden kann.

6

Betrieb des GPS Systems und erste Testergebnisse

In diesem Kapitel sollen anhand der im Laufe dieser Semesterarbeit durchgeführten Testmessungen Hinweise zum Betrieb des entwickelten Gesamtsystems gegeben werden. Zudem werden die Resultate dieses ersten Tests präsentiert und deren Bedeutung kurz diskutiert.

6.1 Beschreibung des Systemtests

Die einzelnen entwickelten Komponenten wurden im Laufe der Entwicklung natürlich immer wieder getestet - einen wirkliche Bestätigung für die korrekte Funktion eines Systems kann aber nur durch einen globalen Systemtest erhalten werden, welcher den gesamten Datenfluss von der Messung bis zur Auswertung der Daten beinhaltet - und zwar unter Umständen welche denjenigen im Feld möglichst ähnlich sind.

Zu diesem Zweck wurden zwei Prototypen der GPS Empfängerstation gebaut. Diese bestehen aus einer standardmässig ausgerüsteten PermaSense Basebox - also mit Mikrotik WLAN Router, Gumstix Embedded PC, TinyNode sowie den nötigen Modifikationen am Baseboard um die Stromversorgung des Gumstix und der USB-Ports vom TinyNode aus zu steuern[1]. Der GPS Empfänger kann an einen beliebigen USB-Port angeschlossen werden und für die Verbindung zur GPS-Antenne wurde eine der in die Box integrierten Antennendurchführungen verwendet.

Die beiden Station wurden auf dem Dach des ETZ Gebäudes bei der bereits bestehenden PermaSense Testinfrastruktur im Abstand von ca. $5m$ montiert. Durch das bereits vorhandenen WLAN Netz mussten diesbezüglich bis auf die korrekte Konfiguration der in die Box integrierten WLAN Router keine weiteren Vorkehrungen mehr getroffen werden.

[1] dies betrifft primär den Transistor Q27 welcher nicht Teil der standardmässigen Bestückung des Baseboards ist

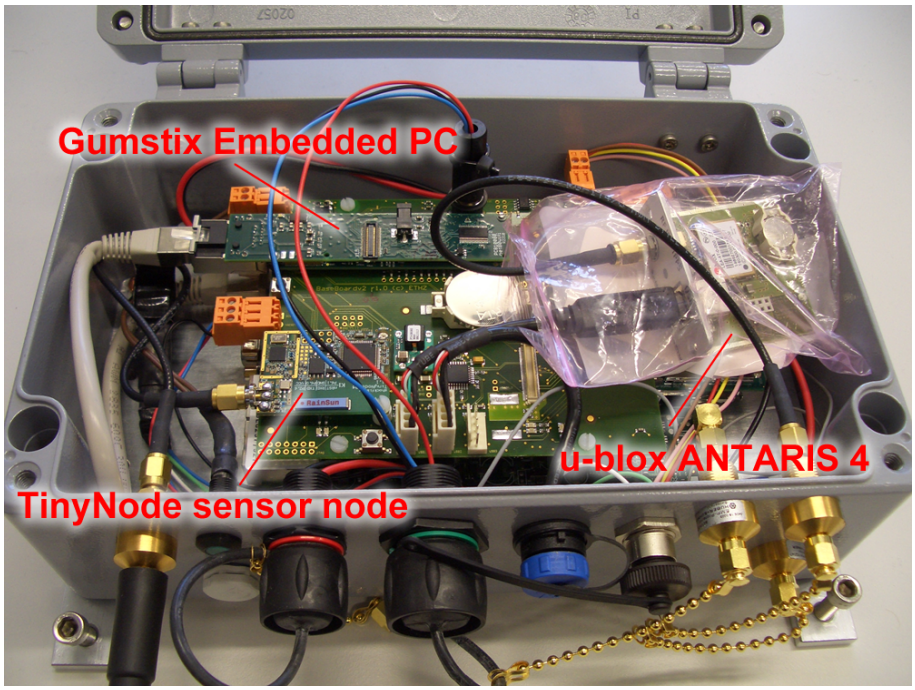
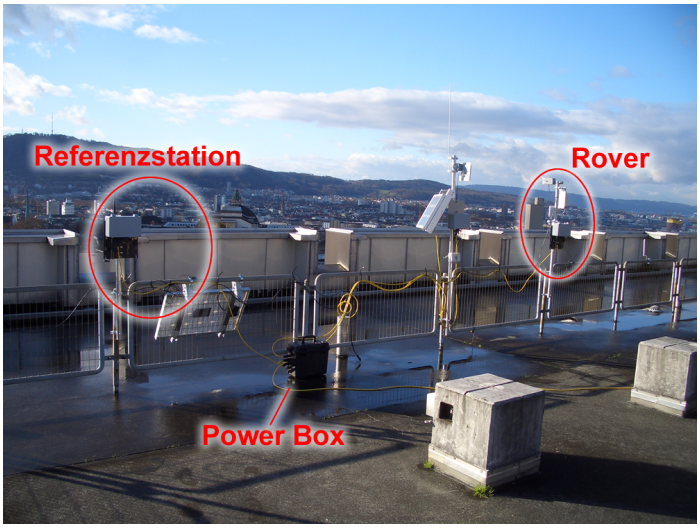


Abbildung 6-1
Die GPS Empfängerstation



(a) Der Versuchsaufbau auf dem Dach des ETZ Gebäudes

(b) Detailansicht einer Empfängerstation

Abbildung 6-2

Das System wurden auf folgenden Betriebsmodus konfiguriert (siehe auch Abschnitt 6.2):

- Es wurde jeweils 15min in 5s Intervallen gesamplet
- Die Messungen wurden im Schedule jeweils auf die volle und auf die halbe Stunde geplant.

Das Experiment dauerte vom 10.12.2009 17:00 UTC bis am 11.12.2009 10:17 UTC. Am 11.12.2009 07:20 UTC - also in einer Pause zwischen zwei Messperioden - wurde die Rover-Antenne um 13cm verschoben um eine Verschiebung des Roverstandortes zu simulieren.

Als GSN Server und somit Backend für das Experiment wurde ein einfacher Laptop mit lokaler MySQL Datenbank benützt. Daraus wurden die Daten anschliessend als CSV Datei exportiert und mit `csv2rinex.php` ins RINEX Format konvertiert. Diese Dateien schickte ich per Email an Dr. Philippe Limpach vom Institut für Geodäsie und Photogrammetrie, welcher sich freundlicherweise dazu bereiterklärt hatte die Daten mit der Bernese GPS Software auszuwerten. Die Ergebnisse des Experiments werden unter 6.3 näher diskutiert.

6.2 Systemkonfiguration

6.2.1 Konfiguration des GPS Empfängers

Der GPS Empfänger wird am einfachsten mit der mitgelieferten Software u-center konfiguriert. Der Chip muss so konfiguriert werden dass er standardmässig keine Nachrichten von selbst ausgibt, da das `GPSPlugin` für PSBacklog jede Nachricht vom Empfänger pollt.

Folgende Schritte sind im u-center nötig:

- Sicherstellen der Verbindung zwischen u-center und GPS-Empfänger (Das Verbindungssymbol erscheint in grün).
- Öffnen der Konfigurationsansicht mit "View"→"Configuration View".
- Auswahl von "CFG (Configuration)", danach die Option "Revert to default configuration" und Bestätigung mit "Send". Dies lädt die Standardkonfiguration auf den Empfänger.
- Auswahl von "PRT (Ports)", dann als Target "USB". Sowohl für "Protocol in" als auch für "Protocol out" muss "0 - UBX" gewählt werden. Bestätigung der Einstellung mit "Send". Dies bewirkt dass auf dem USB Port nur per UBX Protokoll kommuniziert wird.
- Auswahl von "MSG (Messages)". Für alle Messages in der Dropdown Box muss das Feld bei "USB" deaktiviert sein. Jede Deaktivierung muss einzeln mit "Send" bestätigt werden. Dies stellt sicher dass der Empfänger von sich aus keine Nachrichten sendet, sondern nur wenn eine Nachricht aktiv gepollt wird.
- Auswahl von "RATE (Rates)". "Time Source" muss "1 - GPS Time" sein. Bestätigung mit "Send". Dies stellt sicher dass alle Empfänger synchron und auf die GPS Zeit abgestimmt messen.
- Auswahl von "CFG (Configuration)". "Save current configuration" muss aktiviert sein; Bestätigung mit "Send". Damit wird die aktuelle Konfiguration persistent gespeichert. Ansonsten wäre der Empfänger nach einem Verlust der Stromversorgung wieder im alten Konfigurationszustand.

Der GPS Empfänger sollte nun für den Einsatz bereit sein.

6.2.2 Konfiguration des Gumstix Embedded PC

Diese Anleitung geht davon aus dass der Gumstix Embedded PC frisch mit dem neusten System-Image programmiert wurde.

- `/etc/plugins.cfg` muss angepasst werden. *Nur* das `ReverseModePlugin` muss unter `[plugins]` geladen werden. Das Konfigurationsfile für das beschrieben Experiment sieht wie folgt aus:

```
[plugins]
plugin1 = ReverseModePlugin

[options]
tos_source_addr = serial@/dev/ttyS2:57600

reverse_mode_schedule = /etc/schedule.cfg
reverse_mode_connection_wait = 60
reverse_mode_configuration_wait = 10
reverse_mode_service_wakeup = 13:23
reverse_mode_service_uptime = 120
reverse_mode_shutdown_offset = 20

gps_device = /dev/ttyACM0
```

  Für eine Erklärung der einzelnen Konfigurationsparameter wird auf Abschnitt 4.5.1.3 verwiesen.

- Der Inhalt von `/etc/schedule.cfg` für den Systemtest hatte folgende Form:

```
00:00 GPSPlugin:interval=5,measTime=900
00:30 GPSPlugin:interval=5,measTime=900
01:00 GPSPlugin:interval=5,measTime=900
01:30 GPSPlugin:interval=5,measTime=900
02:00 GPSPlugin:interval=5,measTime=900
...
23:00 GPSPlugin:interval=5,measTime=900
23:30 GPSPlugin:interval=5,measTime=900
```

  Eine detaillierte Erklärung des Schedule Formats findet sich in Abschnitt 4.5.1.1.

  Nach einer Änderung direkt im Schedule muss die Datei `/etc/schedule.cfg.parsed` - welche eine für Python optimierte Darstellung des Schedules beinhaltet - gelöscht werden. Ansonsten wird mit dieser Version weitergearbeitet. Wenn ein neuer Schedule via Webupload auf die Station geladen wird so kümmert sich das `ReverseModePlugin` selbst darum.

- Die Datei `/etc/reversemode.cfg` muss angelegt werden, z.B. mit `> touch /etc/reversemode.cfg`. Dies bewirkt dass der Gumstix nach Beendigung von PSBacklog automatisch heruntergefahren wird.

- Beim Bootvorgang müssen die Init-Skripts `/etc/init.d/mikrotik` und `/etc/init.d/psbacklog` ausgeführt werden.

6.2.3 Weitere Konfiguration

- Der TinyNode muss mit der Applikation `BBv2PowerControl` programmiert werden.
- Für jede Empfängerstation muss ein entsprechend angepasster virtueller Sensor gemäss Listing 5.1 im `virtual-sensors` Verzeichnis des GSN Servers angelegt werden. Ein entsprechendes Template befindet sich auch im Unterverzeichnis `ethz` mit dem Namen `Rooftop_GPS_Ref.xml`.

6.3 Diskussion der Ergebnisse

6.3.1 Auswertung der GPS Messungen

Für jedes 15 minütige Messintervall wurde von Dr. Philippe Limpach bei der Datenauswertung eine Positionslösung für den Rover bestimmt. Diese sind in Abbildung 6-3 als grüne Dreiecke dargestellt. Alle Lösungen *vor* der Verschiebung der Roverantenne wurden zudem noch zu einer Gesamtlösung aggregiert (blaues +), alle Lösungen *nach* der Verschiebung wurden ebenfalls miteinander kombiniert (rotes +).

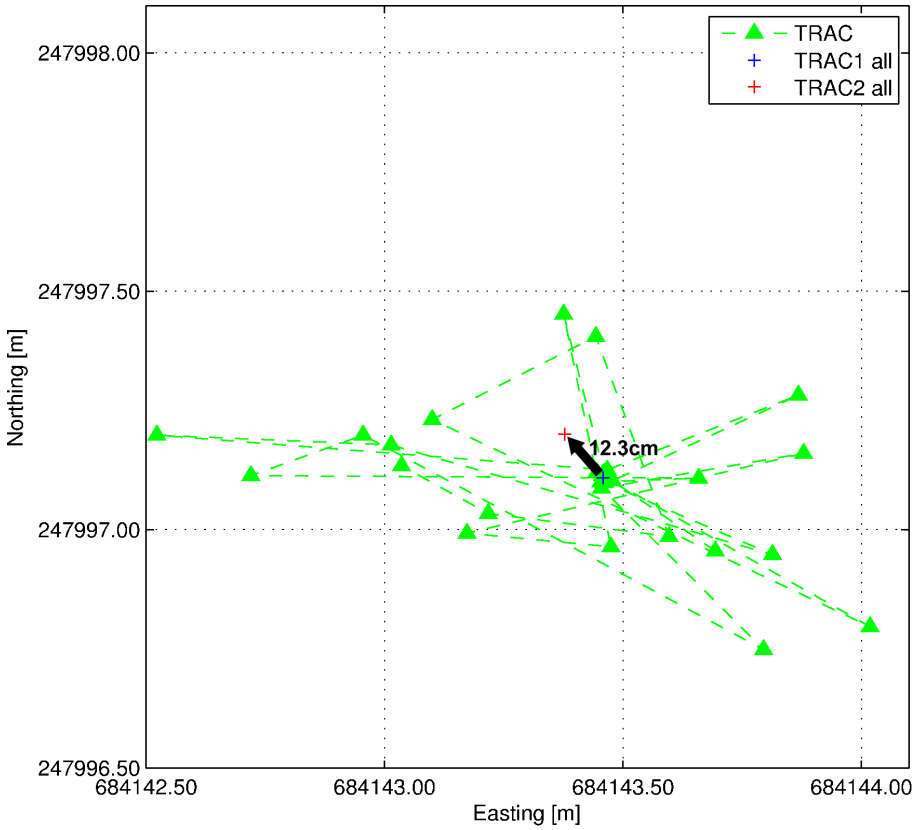


Abbildung 6-3
Resultate der Positionsbestimmung

Mit einem Blick auch die Skala der Grafik wird schnell klar, dass die gewünschte Genauigkeit im Zentimeterbereich bei weitem nicht erreicht wurde. Der Streube-

reich der einzelnen Positionslösungen hat einen Durchmesser von gut $1.5m$ in Ost-West Richtung und $0.75m$ in Nord-Süd Richtung. Wirft man jedoch einen Blick auf die aggregierten Lösungspunkte, so sieht man dass der gemessene Abstand dazwischen mit $12.3cm$ sehr nahe an der tatsächlichen Verschiebungsdistanz von $13cm$ liegt.

Die genauere Betrachtung von Abbildung 6-4 zeigt dass der Fehler in der Positionsbestimmung keineswegs rein zufällig ist: "Schlechte" Lösungen weisen meist in allen Dimensionen grosse Abweichungen von der gemittelten Position auf, andere Lösungen wie z.B. zwischen 20:00 und 22:00 Uhr liegen über längere Zeit ziemlich exakt auf der selben Stelle.

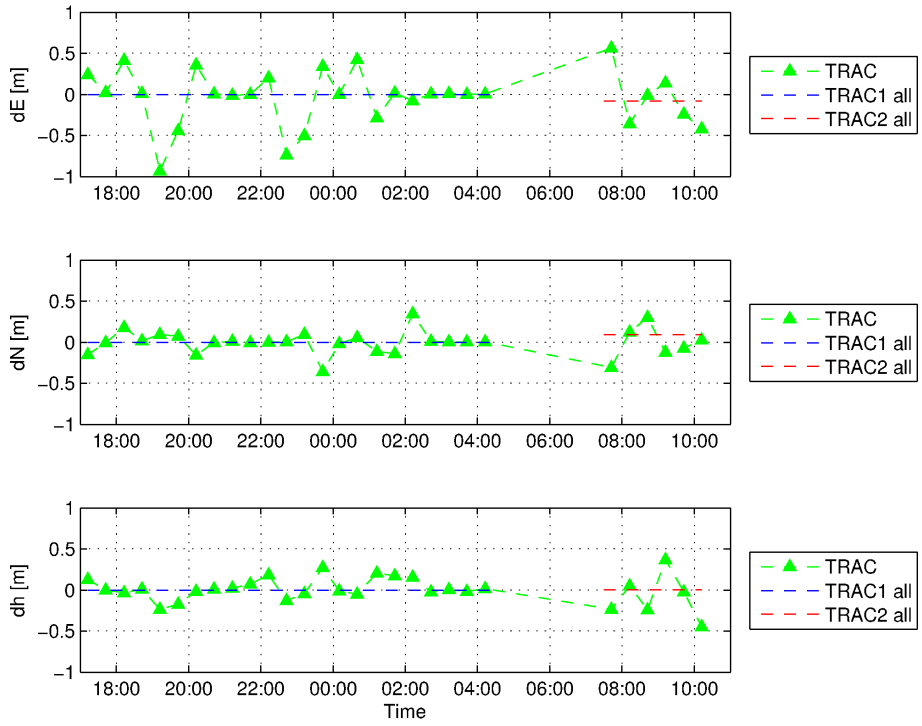


Abbildung 6-4
Resultate der Positionsbestimmung in topozentrischer Darstellung

Es ist schwer zu sagen woher diese grossen Abweichungen bei einzelnen Lösungen herrühren, eine wahrscheinliche Möglichkeit wäre z.B. eine ungünstige Satellitenkonstellation in diesem Zeitraum.

All dies lässt darauf schliessen dass mit entsprechender Parametrisierung auch mit diesem System - welches im Unterschied zu dem in Kapitel 2 vorgestellten Messsystem am Dirru Gletscher nicht rund um die Uhr misst - genaue Positionslösungen erreichbar wären. Jedoch müssen in dieser Hinsicht mehr Erfahrungen gesammelt werden da im Moment noch ziemlich unklar ist welchen Einfluss Änderungen an der Messdauer und der Samplingrate auf die erreichte Präzision der Messungen haben.

6.3.2 Energieverbrauch

Im gesamten Entwicklungsprozess war auch immer wieder der Energieverbrauch des Systems ein Diskussionspunkt. Abbildung 6-5 stellt die Anteile am Verbrauch im Zeitverlauf schematisch dar.

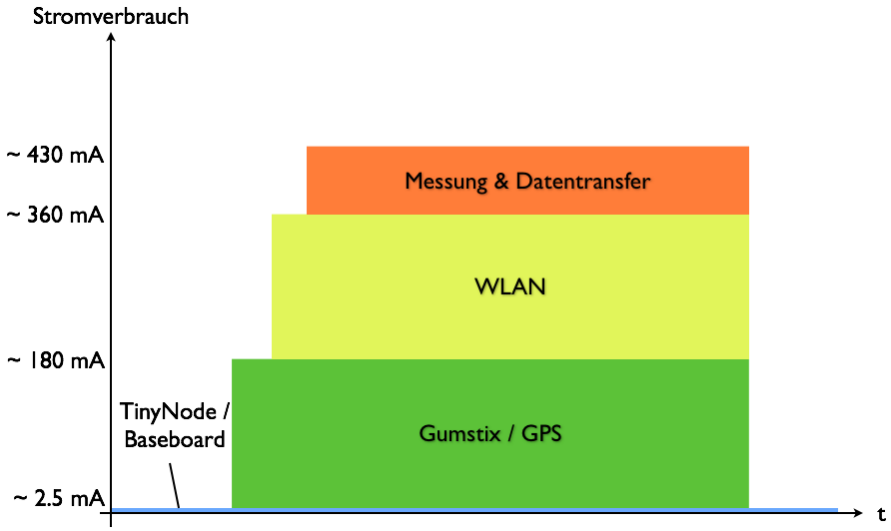


Abbildung 6-5
Aufteilung des Stromverbrauchs einer Empfängerstation

Es zeigt sich dass die Station im Vollbetrieb ca. *170 mal mehr Strom verbraucht* als wenn nur der TinyNode läuft und die restliche Stromversorgung unterbrochen ist. Der Betriebsmodus in welchem der TinyNode die restliche Stromversorgung kontrolliert stellt also eine enorm effiziente Energiesparmassnahme dar - das System bleibt über das Dozer Netzwerk erreichbar und somit jederzeit "online", jedoch mit verhältnismässig winzigem Aufwand an Energie.

6.3.3 Übertragene Datenmengen

Abhängig von der Anzahl sichtbarer Satelliten fallen pro Sample und Empfänger unterschiedlich grosse Datenmengen an. Pro sichtbarem Satellit und Sample beträgt die ans Backend übertragene Nutzdatenmenge $30Byte$. Wird mit einer durchschnittlich sichtbaren Satellitenzahl von 9 Satelliten gerechnet, so ergibt dies eine Datenmenge von ungefähr $270Byte$ pro Sample. In vorliegendem Experiment müssen also pro Positionierunglösung $12 \cdot 15 \cdot 270Bytes = 47.5kB$ an Daten übertragen werden - eine Datenmenge welche das Dozer Netzwerk zur Zeit sicherlich an seine Grenzen bringen würde. Weiter haben wir ja in Abschnitt 6.3.1 festgestellt dass die im Systemtest gemessenen Daten für eine zentimetergenaue Positionierung nicht ausreichen - es ist also in einer realen Messung durchaus mit einem höheren Datenvolumen zu rechnen.

Eine Möglichkeit zur Verringerung des zu übertragenden Datenvolumens wäre eine gewisse Prozessierung auf der Empfangsstation selbst, bevor die Daten ins Backend übertragen werden. Dieser Ansatz bedarf aber eine eingehenden weiteren Untersuchung.

6.3.4 Allgemeine Bemerkungen

Zu guter Letzt sei noch erwähnt dass der Systemtest erst im zweiten Anlauf brauchbare Ergebnisse lieferte. Der erste Test welcher vom 08.12.2009 bis 09.12.2009 dauerte schien zwar technisch korrekt zu funktionieren, jedoch stellte sich später heraus dass die gemessenen Daten nicht auswertbar waren. Grund dafür war dass der eine GPS Empfänger nicht korrekt konfiguriert war und daher die Messzeitpunkte der beiden Empfänger nicht synchronisiert waren. Der zweite Anlauf verlief - bis auf den Stromausfall welcher in Abbildung 6-4 zwischen 04:00 und 07.30 Uhr klar als Messlücke erkennbar ist - störungsfrei. Ob sich das System auch unter Feldbedingungen als zuverlässig erweist wird sich im weiteren Projektverlauf von PermaSense erweisen müssen.

7

Fazit

Zu Beginn dieser Arbeit wurde in Bezug auf die vorgestellte Anwendung von GPS zur Überwachung von Massenbewegungen im Gelände die Problemstellung und somit der Fokus dieser Arbeit formuliert. Diese Problemstellung setzt sich im Wesentlichen zusammen aus

- der Konzeption eines energiesparenden und zuverlässigen Betriebsmodus für ein online GPS System zur Überwachung von Massenbewegungen im Gelände. Dabei sollte der grundlegende Mechanismus auch auf andere experimentelle oder datenintensive Sensoraufgaben einfach übertragbar sein.
- der Implementierung dieses Konzepts in einem lauffähigen Prototypen.

In einer ersten Literaturrecherche konnten zwar einige ähnliche Projekte und Arbeiten ausfindig gemacht werden ([9], [6], [14]), jedoch basieren diese Konzepte alle auf GPS Zweifrequenzempfängern und kommen ohne Datenverbindung zum Backend aus. Einzig in der Masterarbeit von Herwig Lanzendörfer [12] wurden Einzelfrequenzempfänger eingesetzt (notabene ebenfalls die ANTARIS 4 Evaluation Kits), jedoch beschäftigt sich diese Arbeit mehr mit dem Aspekt der Vergleichbarkeit von Einzelfrequenz- zu Zweifrequenzempfängern und stellt Fragen zum Betrieb der Sensoren in den Hintergrund. Sowohl zentimetergenaue Messung mit Einzelfrequenzempfängern als auch nicht-kontinuierliche Messungen in dieser Präzision sind also beides relativ neue Konzepte, zu welchen noch kaum Erfahrungswerte existieren.

Das entwickelte Betriebskonzept - lokaler Scheduler auf dem Gumstix Embedded PC; Steuerung der Stromzufuhr durch den low-power optimierten TinyNode - hat sich sowohl in Laborversuchen als auch im abschliessenden Systemtest auf dem Dach des ETZ Gebäudes bewährt und als zuverlässig erwiesen. Unter Freifeldbedingungen muss sich dieses Konzept erst noch beweisen, insbesondere wird es interessant sein herauszufinden wie hoch die Verfügbarkeit des Systems *tatsächlich* ist - also ob die Dozer-Verbindung des TinyNodes tatsächlich einen quasi-online Status des Systems garantieren kann. Auch für Konzepte wie das "Daily Service Window"

wird sich wohl erst in der realen Anwendung zeigen ob sie den erhofften Gewinn an Kontrolle und Überwachbarkeit des Systems mit sich bringen. Sollte sich das System aber auch unter diesen "erschwerten Bedingungen" bewähren, so wird sich eine Einbindung von anderen Sensoren in das Kontroll- und Kommunikationssystem mittels des PSBacklog `ReverseModePlugin` relativ einfach bewerkstelligen lassen.

In Bezug auf die durchgeführten GPS Messungen ist der Verbesserungsbedarf der bisher erreichten Genauigkeit offensichtlich. Wie jedoch schon in Abschnitt 6.3.1 dargelegt sind durchaus Hinweise erkennbar dass mit veränderter Parametrisierung der Messungen wohl deutlich präzisere Resultate erzielt werden könnten. Mit dem im Verlaufe dieser Arbeit entwickelten Messsystem steht dafür aber auf jeden Fall eine mächtige Plattform zur Verfügung, mit welcher ohne übermässigen Aufwand verschiedene Parametrisierungen getestet und ausgewertet werden können.

Literaturverzeichnis

[1] *ANTARIS Protocol Specification*. `http://www.u-blox.com/en/download-center.html`, 2003.

[2] ABERER, K., M. HAUSWIRTH und A. SALEHI: *The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks*. Techn. Ber., 2006. Sumitted to ACM/IFIP/USENIX 7th International Middleware Conference.

[3] BEUTEL, J., S. GRUBER, A. HASLER, R. LIM, A. MEIER, C. PLESSL, I. TALZI, L. THIELE, C. TSCHUDIN, M. WOEHRLE und M. YUECEL: *PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes*. In: *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, S. 265–276, Washington, DC, USA, 2009. IEEE Computer Society.

[4] BURRI, N., P. VON RICKENBACH und R. WATTENHOFER: *Dozer: ultra-low power data gathering in sensor networks*. In: *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, S. 450–459, New York, NY, USA, 2007. ACM.

[5] FERRIÈRE, H. D., L. FABRE, R. MEIER und P. METRAILLER: *TinyNode: a comprehensive platform for wireless sensor network applications*. In: *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, S. 358–365, New York, NY, USA, 2006. ACM.

[6] GILI, J. A., J. COROMINAS und J. RIUS: *Using Global Positioning System techniques in landslide monitoring*. Engineering Geology, 55(3):167 – 192, 2000.

[7] GURTNER, W.: *RINEX: The Receiver Independent Exchange Format Version 2.11*. `ftp://ftp.unibe.ch/aiub/rinex/rinex211.txt`, 2007.

[8] HAREL, D.: *Statecharts: A Visual Approach to Complex Systems*. In: *Science of Computer Programming*, S. 231–274. Elsevier Science Publishers B.V., 1987.

[9] HARTINGER, H. und F. BRUNNER: *Development of a monitoring system of landslide motions using GPS*. In: *Proc. 9th FIG Symp. on Deformation Monitoring*, S. 29–38, 2000.

[10] KELLER, M., J. BEUTEL und L. THIELE: *Mountainview - Precision Image Sensing on High-Alpine Locations*. In: *6th European Workshop on Sensor Networks (EWSN 2009), Cork, 11-13 Feb 09*, 2009.

[11] KÖHNE, A. und M. WÖSSNER: *Wie funktioniert GPS. Alles Wissenswerte*. `http://www.kowoma.de/gps`, 2008.

[12] LANZENDÖRFER, H.: *Zum Einsatz von low-cost GPS-Empfängern für kontinuierliches Monitoring eines Rutschhanges*, 2007.

[13] SIEWIOREK, D. P. und R. S. SWARZ: *Reliable computer systems (3rd ed.): design and evaluation*. A. K. Peters, Ltd., Natick, MA, USA, 1998.

[14] SQUARZONI, C., C. DELACOURT und P. ALLEMAND: *Differential single-frequency GPS monitoring of the La Valette landslide (French Alps)*. Engineering Geology, 79(3-4):215 – 229, 2005.

[15] ZOGG, J.-M.: *GPS Compendium - Essentials of Satellite Navigation*. `http://www.u-blox.com/de/tutorials-links-gps.html`, 2009.