UNIVERSITY OF WEST BOHEMIA
FACULTY OF ELECTRICAL ENGINEERING

DEPARTMENT OF APPLIED ELECTRONICS
AND TELECOMMUNICATIONS

in cooperation with
ESIEE PARIS

DIPLOMA THESIS

Bc. Matěj Kubička 2014

This page is intentionally left blank.

UNIVERSITY OF WEST BOHEMIA
FACULTY OF ELECTRICAL ENGINEERING

DEPARTMENT OF APPLIED ELECTRONICS
AND TELECOMMUNICATIONS

in cooperation with

ESIEE PARIS

DIPLOMA THESIS
Road-vehicle navigation

Author: Bc. Matěj Kubička
Supervisor: Prof. Hugues Mounier

Paris, 2014

Statement

I hereby submit for review and defense my diploma thesis. I declare that I prepared this thesis independently using professional literature and resources listed in the list, which is part of this thesis. I also declare that software used to prepare this thesis was obtained legally.

Note that part of this thesis was converted into conference paper and submitted at IEEE Intelligent Transportation Systems Conference, 2014.

In Paris on June 30, 2014 ....

Abstract

Kubička, M. "Road-vehicle navigation". Laboratory of signals and systems (CNRS), 2014, 103 p., supervisors: Prof. Hugues Mounier, Dr. Arben Cela, Dr. Silviu-Iulian Niculescu.

Aim of this diploma thesis is to develop navigation system for terrestrial vehicles traveling on known routing network. It is but one part of larger project that deals with next-generation optimal routing. In context of this project the navigation system is required to provide precise positioning information on a road-network map, hence the name "road-vehicle navigation".

The thesis starts with thorough analysis of the problem. Satellite navigation, inertial navigation and other navigation aids are analyzed. Then, solution proposal is drafted from this analysis. It reflects performance benefits versus factors such as availability, price and complexity.

With the proposal at hand the thesis continues by solving major cornerstones of the road-vehicle navigation problem. First, embedded system with various sensors is presented. Its printed circuit board design and microcontroller software is described in detail. Then, novel map-matching algorithm is proposed and tested with simulations on real data. Finally, an application for Android smart devices that implements road-vehicle navigation system prototype is presented.

Keywords

Integrated navigation system, satellite navigation, inertial navigation, odometers, map-matching, road network, OBD-II, Bluetooth, GNSS, GPS, GLONASS, INS, MEMS, accelerometers, gyroscopes, magnetometer, barometric pressure sensor, feature matching

Anotace

Kubička, M. "Road-vehicle navigation". Laboratoř signálů a systémů (CNRS), 2014, 103 s., vedoucí: Prof. Hugues Mounier, Dr. Arben Cela, Dr. Silviu-Iulian Niculescu.

Cílem této diplomové práce je vyvinout navigační systém pro pozemní vozidla cestující po předem známé síti cest. Jedná se o část většího systému, který řeší problém optimálního plánování cesty s ohledem na stav sítě cest a stav různých zdrojů energie uvnitř hybridního vozidla. V kontextu tohoto projektu je třeba mít navigační systém schopný určit polohu vozidla podél sítě cest s vysokou přesností.

První část této práce je věnována analýze problému, který řeší. Satelitní navigace, inerciální navigace a další techniky jsou nejprve prozkoumány odděleně a poté jsou zkoumány možnosti jejich integrace v jednom navigačním systému. Výstupem této analýzy je nástin optimálního řešení, ke kterému tato i následující práce budou směřovat.

V dalších částech této práce se věnuji vývoji kritických součástí navigačního systému v kontextu provedené analýzy. Nejprve je navržena deska plošných spojů s přijímačem pro satelitní navigaci, s inerciální navigací, OBD-II rozhraním a dalšími senzory. Poté je popsán software s drivery ke všem hardwarovým součástím desky plošných spojů. Dále je představen nový algoritmus pro párování navigačních dat se známou sítí cest a otestován v simulaci s reálnými daty.

Finální část této práce popisuje aplikaci pro systém Android která realizuje prototyp "road-vehicle" navigačního systému za pomoci párovacího algoritmu a zmíněné desky plošných spojů.

Klíčová slova

Integrovaný navigační systém, satelitní navigace, inerciální navigace, odometrie, map-matching, feature matching, mapa sítě cest, OBD-II, Bluetooth, GNSS, GPS, GLONASS, INS, MEMS, akcelerometr, gyroskop, magnetometr, barometrický měřič nadmořské výšky.

Résumé

Kubička, M. "Road-vehicle navigation". Laboratoire des signaux et systèmes (CNRS), 2014, 103 p., superviseurs: Prof. Hugues Mounier, Dr. Arben Cela, Dr. Silviu-Iulian Niculescu.

Cette étude a pour but de developper un systéme de navigation pour les vehicules terrestres dans le traffic routier. Il s'agit d'un module d'un grand projet qui traite la routage optimal concernant l'état du réseau routier et le statut des différentes sources d'énergie dans le véhicule hybride (de la prochaine generation). Dans le cadre de ce projet, le système de navigation est requis de fournir des informations de positionnement précis le long du réseau routier, d'où provient le nom navigation du véhicule routier.

La thèse commence par une analyse approfondie du problème et de sa modélisation. La navigation par satellite, la navigation inertielle et d'autres techniques de la navigation sont analysées séparément, puis les possibilités de leur intégration sont étudiés.

Ensuite, une solution optimale à partir de cette analyse est proposée. Il reflète les avantages et les performances par rapport à des facteurs tels que la disponibilité, le prix et la complexité.

Cette étude se poursuit par une proposition de solution du problème de navigation des véhicule routiers. Premièrement, un module de calcul autour d'un processeur et intégrant des divers capteurs est conçue ainsi que son logiciel et les drivers pour toutes les composantes de mesure, de calcul et de communication du circuit imprimé.

Puis un nouvel algorithme pour le couplage des données de navigation avec des réseaux connus est présenté et testé par des simulations sur des données réelles.

Mots clés

Système intégré de navigation, la navigation par satellite, la navigation inertielle, odomètres, map-matching, le réseau routier, les systèmes OBD-II, Bluetooth, GNSS, GPS, GLONASS, INS, MEMS, accéléromètres, gyroscopes, magnétomètre, capteur de pression barométrique, feature matching

Contents

List of Figures

List of Tables

List of Listings

This page is intentionally left blank.

Chapter 1

Introduction

This work was conducted in Laboratory of Signals and Systems L2S/Supelec during six month internship at the end of my undergraduate study. The Laboratory of signals and systems (commonly known as L2S) is a research unit of the Centre national de la recherche scientifique (CNRS), École supérieure d'électricité (Supélec) and Paris-Sud University.

Main goal of this work was to conduct thorough analysis of Road-vehicle navigation problem and to propose optimal architecture in terms of performance. Secondary goal was to implement this proposal to some degree. It was never required to implement it completely as the proposal was not designed to be implementable in few months by a single developer.

As part of the analysis I have designed embedded system to run tests on sensors, navigation systems and communication interfaces we were considering with my supervisors to deploy. This prototype system has shown capable to host integrated navigation system as proposed in the analysis. It is described in Chapter 3.

Important part of road-vehicle navigation system is so-called map-matching algorithm. This algorithm matches Earth-referenced position to position on a map and provides map-referenced position. Hence, the map-matching algorithm promotes classical navigation systems to road-vehicle navigation systems, see Figure 1.1.

Please note that although both analysis and implementation was conducted by me, I

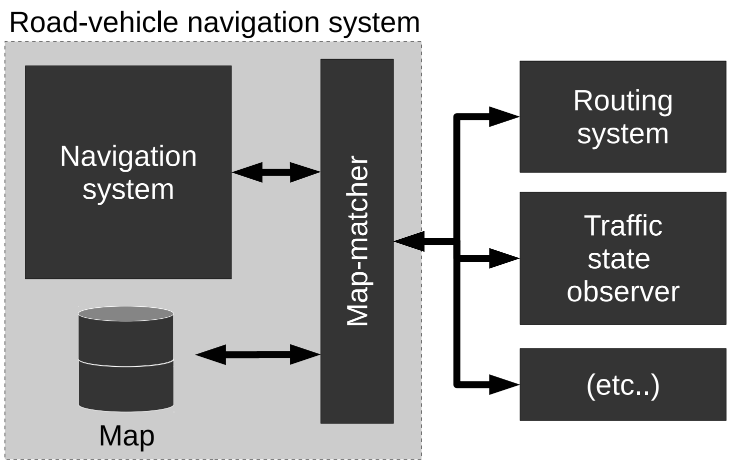


Figure 1.1: Relation between navigation system and road-vehicle navigation system.

was lead by my supervisors who were managing this project. Thorough the thesis text I often refer to authors in plural form to acknowledge it.

Document organization

Chapter 2 is concerned with the analysis, it presents theoretical background and practical test results on various devices that can potentially realize or enhance navigation system performance. Chapter 3 is concerned with design of the embedded system. Printed circuit board is detailed as well as microcontroller's program, peripheral drivers and device drivers. Chapter 4 is concerned with map-matching algorithm, optimal data structures to hold high resolution maps and with "rvn-app" application that implements prototypical road-vehicle navigation system.

Appendices of this document contain description of tools we developed to conduct this work, printed circuit board designs and detailed specification of communication protocol between embedded system and the Android device.

Chapter 2

Problem analysis

This chapter is concerned with technologies that have potential to enhance performance of road-vehicle navigation system we develop. Performance and properties of satellite navigation, inertial navigation and other navigation aids is analyzed separately in sections 2.1-2.4. Section 2.5 is concerned with various map-matching algorithms proposed so far and with analysis of required map-matching algorithm behavior. Then, Section 2.6 lists various ways how navigation systems can be integrated and, finally, Section 2.7 proposes architecture of the road-vehicle navigation system based on this analysis.

2.1 Satellite navigation

Satellite navigation refers to family of systems that use satellites to provide positioning information. These are GPS, GLONASS, Galileo, Compass, Beidou, and IRNSS. Satellite navigation systems are generally referred to as GNSS which is shortcut for Global Navigation Satellite System[1].

GPS is oldest. It has reached full operation capability on January 17, 1994. Two services were provided - first, called C/A code, is freely available for civilian use and second, called P code, is intended for military use. The C/A code was intentionally crippled with selective availability algorithm reducing precision to about 100 meters. Selective availability was disabled on May 1, 2000 enabling horizontal precision of 20 meters. Today, GPS receivers are able to deliver precision in order of few meters. Unfortunately this is never guaranteed as it depends on many environmental factors (visibility, multipath, sun's activity, weather..).

GLONASS is Russian navigation system closely similar to GPS. GLONASS reached full operation capability in 1995, but since 1996 system went into decline losing its ability to provide positioning. The system underwent restoration since 2000, reaching full operation capability in October 2, 2011. It has precision of 5-10 meters since restrictions on its formerly military signal were lifted on May 18, 2007.

GLONASS uses same underlying principles as GPS, but their implementation is different. GLONASS uses FDMA (frequency division) channel access method while GPS

[1]Acronym GLONASS actually translates to GNSS in English. We will use term GNSS to refer to satellite navigation systems in general and GLONASS to refer to Russian system specifically.

Figure 2.1: GPS satellite (block IIA; source: U.S. Air Force)

uses CDMA (code division). This prevents cheap integration of GPS and GLONASS receivers into a single device. First integrated GPS/GLONASS receivers surfaced in 2012. These devices used two separate baseband processors - one for GPS and another one for GLONASS.

Galileo is GNSS system currently developed by European Union. Distinctive feature of Galileo is that it is designed as commercial system (in contrast to GPS and GLONASS which are operated by military). In the time of writing, Galileo is in in-orbit validation phase, having 4 satellites orbiting earth. Full-operation capability is scheduled for 2019. Galileo is on its way to be the first fully operational second-generation satellite system meaning that it will offer global integrity, integrated SBAS (see below) and high chipping rate signals.

BeiDou and Compass are developed by China. BeiDou is different from other systems as it uses two-way communication with geostationary satellites above China. This technology allows only limited number of active users at a single moment, has lower precision than GPS and covers only local parts of Asia. Compass, also known as BeiDou-2, is currently developed as China's truly global GNSS system. It is scheduled for full operational capability by 2020.

IRNSS (Indian Regional Navigation Satellite System) is GNSS system currently developed by India. Seven geostationary satellites are planned to provide independent georanging codes for civilian and military use above Indian peninsula.

Apart of pure GNSS systems listed above there are satellite-based augmentation systems (SBAS) that locally improve performance of GPS (or GLONASS). Currently in operation are EGNOS, WAAS and GAGAN and few others. These systems use a number of geostationary satellites above their area of interest and provide some or all of following services:

- *Geo-ranging "GPS like" signals* that improve satellite visibility in urban areas.
- *Differential corrections*, used to correct errors that affect all receivers in the area covered by SBAS system. These are atmosphere propagation delays and satellite residual clock offsets and drifts.

- *Integrity*, used to validate positioning data and to automatically recover from error states, if possible. This is used in critical applications such as aircraft positioning. It requires the SBAS system to detect and broadcast anomaly in GNSS system within few seconds from its occurrence.

Above Europe we have EGNOS which provides all three services (geo-ranging, differential corrections and integrity). WAAS covers United States of America and GAGAN covers Indian peninsula.

2.1.1 Principles

Many things can be said about satellite navigation but here we restrict ourselves to bare minimum with which we can understand performance characteristics of GNSS systems. Among other things, we completely omit modulation schemes, coding and problems related to signal acquisition and tracking. Instead, we assume that we track signals from j satellites knowing time of transmission from i-th satellite t_{si}, local time of reception t_{ri} and position of the satellite, p_i, when the signal was transmitted. With this at hand we can compute range ρ_i from i-th satellite to receiver

$$\rho_i = (t_{ri} - t_{si})c \tag{2.1}$$

in meters, noting that c is the speed of light. Note that we assume that receiver and satellite clocks are perfectly aligned. Even $1\mu s$ offset would cause error of 300 meters. Clock synchronization in sub-microsecond level is practically impossible to achieve. For this reason ρ_i is called "pseudo-range" to account for the fact that it contain receiver clock offset error.

Imagine we track 3 satellites, then if we ignore the clock offset we can obtain receiver's antenna position simply by computing intersection of 3 spheres centered at p_i (satellite positions) with radius ρ_i (distance satellite-receiver). This yields system of three equations with three unknowns with form

$$\rho_i = ||p_i - p_r|| \tag{2.2}$$

where $i = \{1, 2, 3\}$ and p_r is unknown receiver's antenna position. When fully expanded, equation 2.2 translates to following

$$\begin{aligned} c^2(t_{r1} - t_{s1})^2 &= (p_{1x} - p_{rx})^2 + (p_{1y} - p_{ry})^2 + (p_{1z} - p_{rz})^2 \\ c^2(t_{r2} - t_{s2})^2 &= (p_{2x} - p_{rx})^2 + (p_{2y} - p_{ry})^2 + (p_{2z} - p_{rz})^2 \\ c^2(t_{r3} - t_{s3})^2 &= (p_{3x} - p_{rx})^2 + (p_{3y} - p_{ry})^2 + (p_{3z} - p_{rz})^2 \end{aligned} \tag{2.3}$$

However, the receiver clock error would render results unusable due to not synchronized clocks in receiver and on the satellite. For this reason we treat receiver clock error as fourth unknown in our system of equations, requiring 4 satellites to provide three dimensional position. This is possible because receiver clock offset δ_{rc} is common for all pseudo-ranges.

$$\rho_i + \delta_{rc} = ||p_i - p_r|| \tag{2.4}$$

With less than 4 tracked satellites the system is underestimated and cannot be solved (unless some variable is supplied externally). With exactly 4 tracked satellites the system

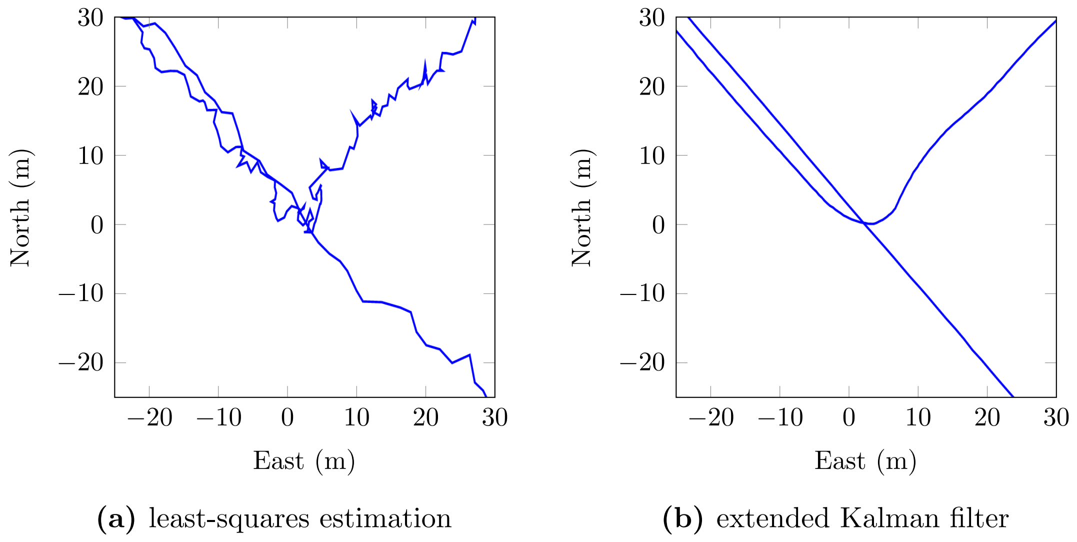


Figure 2.2: Comparison of estimated position with and without filtering (loc: 49.730836, 12.833126).

can be solved exactly. With more than four tracked satellites the solution is overestimated and position can be obtained by means of least squares estimation, although Kalman filter is used in practice. Least-squares solution is often used for RAIM[2] function.

The transmission time, t_{si}, is encoded in the message that satellite transmits. There is high precision clock on board keeping precise time reference. Residual errors in satellite clocks with respect to UTC time reference are observed on ground by GNSS system operator and periodically uploaded to satellites. Satellites do not correct their clocks, but forward the information to receivers. Hence, residual clock error for each satellite is corrected in the receiver.

Satellite's own position is transmitted in a set of parameters known as ephemeris. GPS and Galileo ephemeris contain 16 parameters that describe satellite position on Keplerian orbit. GLONASS satellites transmit position directly in ECEF frame - with position, velocity and acceleration in Cartesian coordinates. This is easier to work with as GPS's ephemeris data have to be transformed into ECEF coordinates, requiring nonlinear equation to be solved. Nonetheless, time of validity for ephemeris data in GLONASS is shorter than for GPS. Note that satellites do not send their current position, but rather some older position that receiver can update to current time.

2.1.2 GNSS receiver architecture

In this section we briefly review main parts of GNSS receiver. Using terminology of [5] we can separate receiver into 4 parts: (1) antenna, (2) baseband processor, (3) ranging processor and (4) navigation processor.

The antenna and baseband processor are classically implemented in hardware[3]. Baseband processor down-converts modulated signal, samples it and separates signals from different satellites using bank of correlators where the signal is compared to locally generated code known in advance (pseudo-random code, each satellite has its own).

[2]Receiver Autonomous Integrity Monitoring

[3]Note that software receivers have baseband processor implemented in software [5].

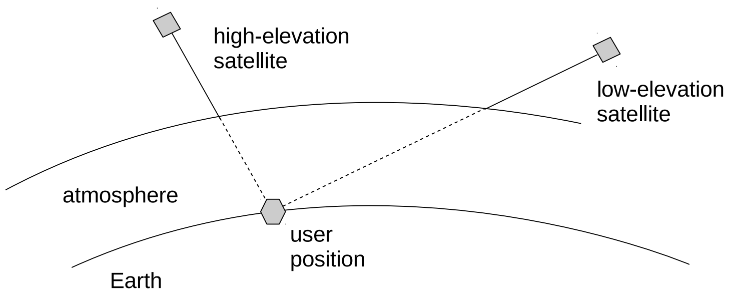


Figure 2.3: Dashed line depicts distance signal travels in atmosphere for low- and high- elevation satellite (image not in scale).

Ranging processor controls the correlator banks. Satellite signal is found when correlation peak is detected (process that searches for the peak is called acquisition) and PLL loop keeps the receiver locked on signal from there on (process called tracking). Ranging processor computes pseudo-ranges, signal-to-noise ratios, demodulates the message data, etc. All this information is then provided to navigation processor which typically implements extended Kalman filter and outputs position. Note that Kalman filter uses pseudo-range rates (ρ_i derivatives) as aid to compute filtered position solution. In practice these can be obtained with the PLL discussed above[4]

2.1.3 Signal strength, line-of-sight

GNSS signals are weak as they fall below noise floor. GNSS receiver has to search for the signal by correlating samples with known PRN codes. GNSS carriers are in L band, many with frequency around 1550 Mhz. This is microwave frequency band meaning that solid objects tend to attenuate the signal. Hence, GNSS carrier cannot pertain to buildings and even foliage have effects on signal strength.

Note that reflected signals cause positioning errors as they are delayed. This introduces error to pseudo-range computation so only signals in direct line-of-sight between receiver and satellite are usable. See Section 2.1.5 for further information.

2.1.4 Atmospheric errors

GPS satellites are located in medium Earth orbit at altitude of 20,200 kilometers, Galileo satellites are at altitude of 23,222 kilometers and GLONASS satellites at 19,100 kilometers.

Transmitted signals are refracted by free electrons in ionosphere and by gases in troposphere. Induced propagation delays vary with ionosphere's ionization due to sun's activity, with satellite's elevation and also due to weather conditions in troposphere.

There is more refraction when satellites have low elevation with respect to the receiver, see Figure 2.3. Receivers can compensate for that as propagation error is approximately proportional to cosine of elevation angle, see [5] for the formula.

Solar radiation charges ionosphere during the day, causing more refraction than in the night. Induced pseudo-range error can reach 15 meters during the day and 3 meters in the night [5]. Moreover, ionosphere is a dispersive medium so propagation delay is not

[4]Carrier and phase tracking is a complex topic, see [1] for further information.

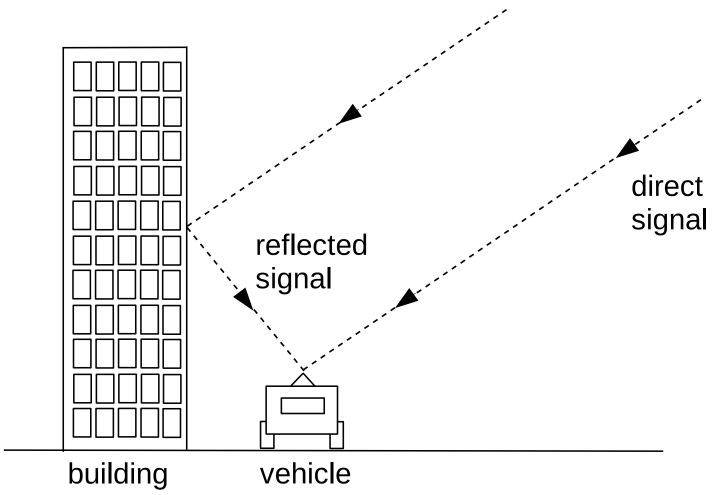


Figure 2.4: Example of multipath interference - signal from same satellite is received twice, reflected signal is delayed with respect to direct one.

uniform on all frequencies. If the receiver tracks signal on more than one frequency, then the propagation delays can be observed and mostly removed from the pseudo-range.

Troposphere is non-dispersive medium so all signals are delayed uniformly, no matter their frequency. It is attributable to the error of about 2.5 meters, mostly due to dry gases in troposphere [5]. Weather fronts (water vapor) have minor effect on troposphere propagation delay as well.

Differential corrections

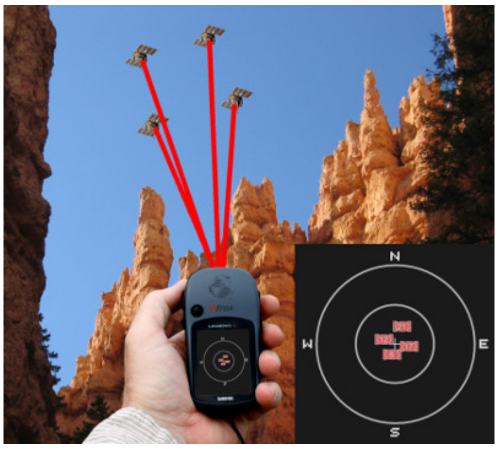
Atmospheric errors are global, meaning that correlation distance is in the order of hundreds of kilometers. Hence, they can be corrected for as they are common for all receivers in a large area. Satellites provide parameters of Klobuchar's ionospheric model that corrects about half of the error.

One have to know local properties of ionosphere for increased precision. This is usually resolved with reference stations that compute the difference between real pseudo-range and observed one. These so-called differential corrections are then supplied to GNSS receivers in radius of hundreds of kilometers from the reference station. Sub-meter accuracy is attainable by subtracting the difference from receiver's own computed pseudo-range.

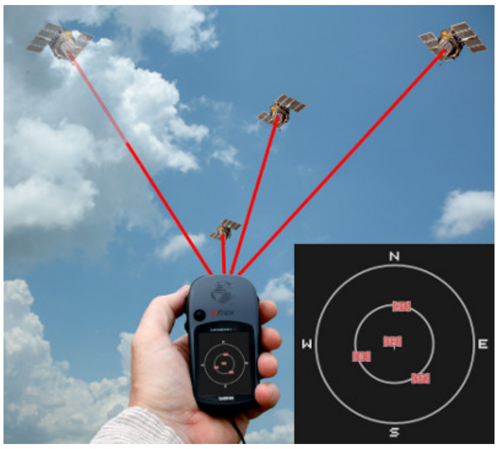
There are various ways how differential corrections are transported to GNSS receivers. SBAS systems upload corrections to geo-stationary satellites that forward the information to receivers. Other technique is to tunnel the differential corrections through the internet. This can be so-called differential GPS (DGPS) or SBAS satellite data stream encapsulated in TCP/IP packets (case of SiSNET service that forwards EGNOS data). Finally, differential corrections are sometimes forwarded by radio beacons transmitting in LF band. This is the case of LORAN terrestrial navigation system that used to forward WAAS differential corrections in its data channel in North America.

2.1.5 Multipath interference

Multipath interference is caused by reflection of GNSS signals from solid objects. In land applications, signals reflect mostly from the land and surrounding buildings, see Figure



(a) bad DOP



(b) good DOP

Figure 2.5: An example of two constellations and related DOP coefficients (source: Wikimedia).

2.4. Consequence is that GNSS receiver receives reflected signals in addition to direct ones. Reflected signals tend to be delayed which introduces error to pseudo-range computation (see equation 2.1).

Direct signals are always right-hand circularly polarized, reflected signal may be both left- and right- hand circularly polarized as single reflection inverts it. Some antennas mitigate multipath interference by rejecting left-hand circular polarized waves.

Moreover, when higher chipping rate[5] is used, then multipath interference is mitigated as well because the difference in observed range from direct and refracted signal must be smaller in order to affect the computation.

2.1.6 DOP errors

Acronym DOP refers to "dilution of precision". Some satellite constellations can enhance effect of errors in pseudo-ranges when combined to single positioning solution. The idea of DOP[6] is to express how errors in pseudo-ranges affect precision of positioning solution. It is unit-less coefficient.

When receiver tracks signals from satellites that are close to each other (from receivers point of view), then the position will be unstable in receiver-satellites direction (see Figure 2.5a). On the other hand, if the satellites are well spaced above the receiver, then the position is more stable (see Figure 2.5b).

DOP parameters had great importance in early GPS receiver designs as these have been able to track only a few satellites at a time. Choice of a set of tracked satellites had effect on performance. Nowadays, receivers are able to track 30+ satellites from multiple systems simultaneously, so they don't need to choose a set of satellites to track - they usually track them all. Hence, dilution of precision errors are mitigated with performance of current GNSS receivers. The problem still exist, though, in situations where only small fraction of the sky is visible.

[5]Term "chipping rate" is not discussed here, see [1] for definition.
[6]It it sometimes called GDOP as Geometric DOP.

2.1.7 Stationary tests

We have tested performance of combined GPS/GLONASS receiver[7] NV08C-CSM from NVS Technologies AG [15]. This device has 2.5-meter horizontal precision (standard deviation) in autonomous mode and 1-meter precision when differential corrections are used.

Two tests were carried out - first was using least-squares position estimator and other used extended Kalman filer. Both tests were stationary meaning that antenna didn't move. Both tests were running non-stop for 24 hours, allowing to see error variation during the day and night. Both GPS and GLONASS satellites circle Earth twice in that period. Note that precise position of the antenna was unknown.

The receiver was set for 10 Hz update rate, giving some 864,000 measurements during the test. It tracked usually between 19 and 24 satellites. Support for SBAS was enabled, although SBAS satellites were not always tracked. We have asked manufacturer to explain this and the answer was that SBAS satellite tracking is implemented as low-priority task. Device's CPU gets overloaded when there are more than 20 other satellites to track.

Test results are shown on Figure 2.7. Unfiltered position shows normal random distribution with standard deviation of 14.4 meters in East direction and 5.02 meters in North direction. See Figure 2.7a for scatter plot and Figure 2.6 for probability mass function.

Test that uses extended Kalman filter to estimate position is shown on Figure 2.7b. Results show random walk of reported position in radius of 2.5 meters around mean (real antenna position is unknown). This matches receiver's specification provided by manufacturer. Figure 2.8 shows in detail measurement obtained during the night. Bounding radius for random walk in this period is significantly smaller, about 1 meter. This indicates that errors in position are generally smaller in the night than in the day. Note that we don't have any proof of that as we would need to compare the data to precise position.

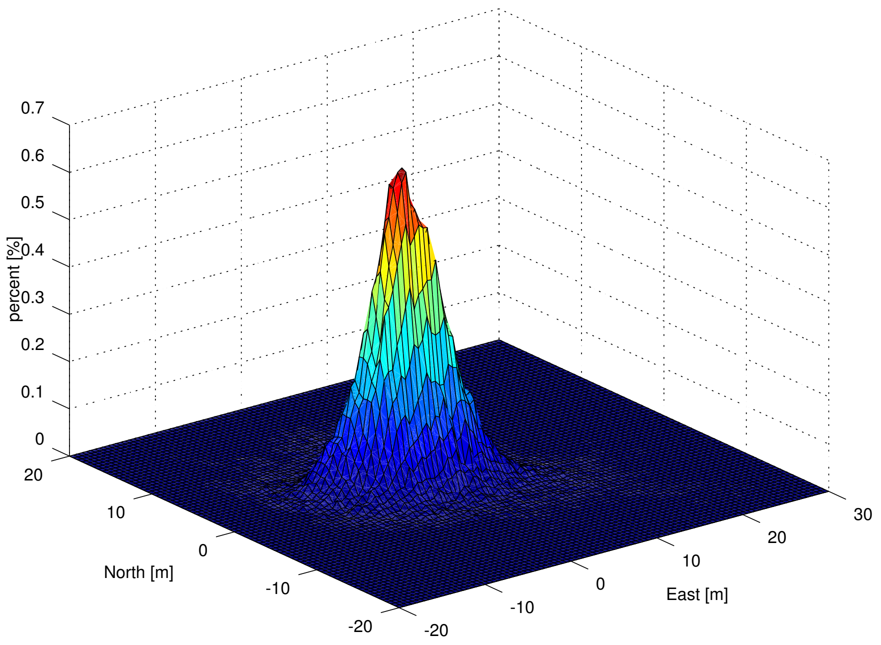
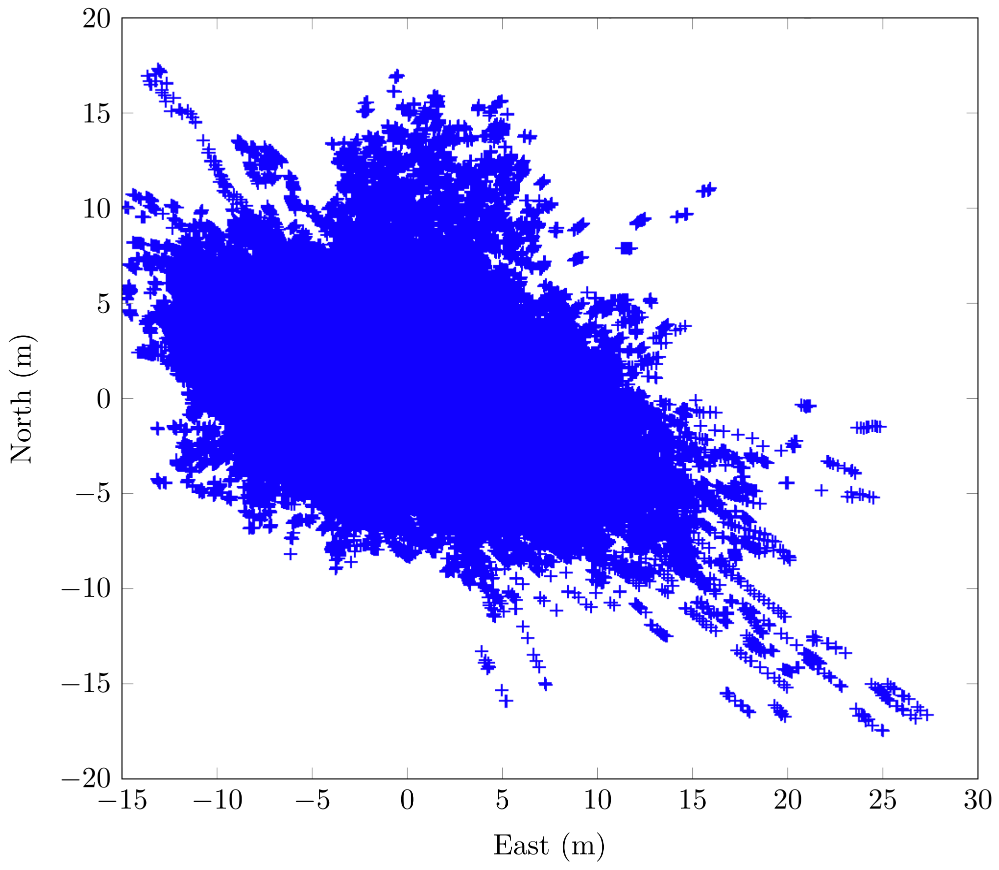
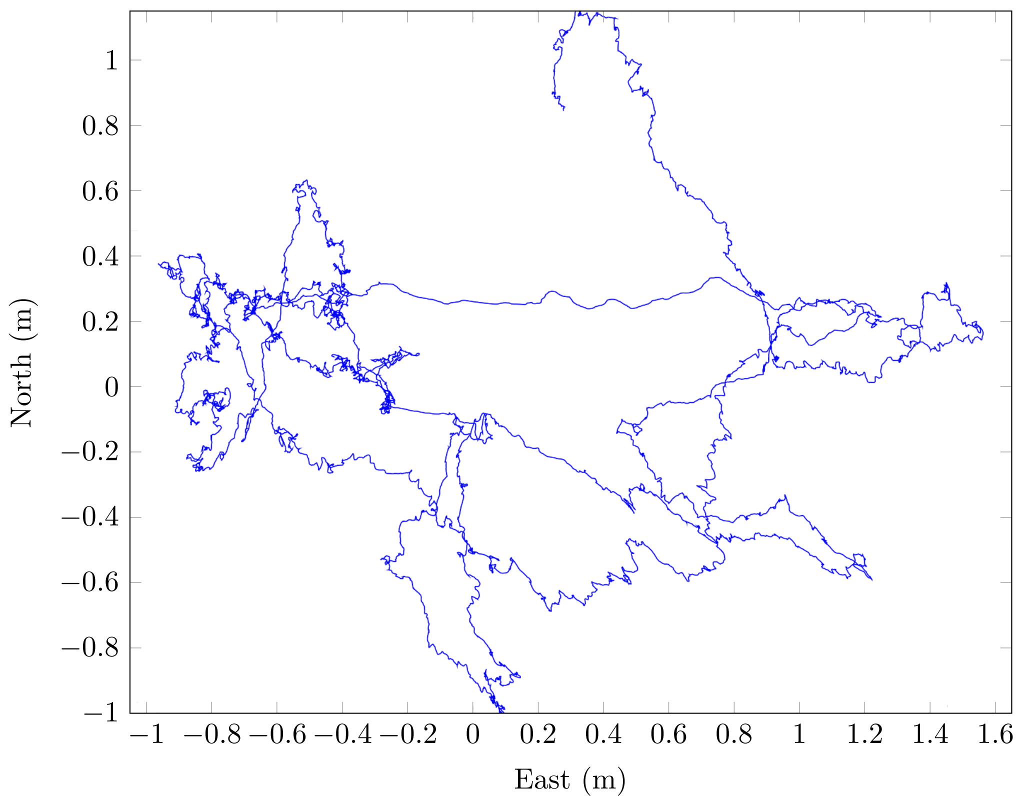


Figure 2.6: Probability mass function of scatter plot on Figure 2.7a.

[7]This receiver can also work with Galileo and BeiDou-2 (with change of firmware).



(a) least-squares position estimator (unfiltered)



(b) Extended Kalman filter

Figure 2.7: Satellite navigation test results (24-hours, stationary).

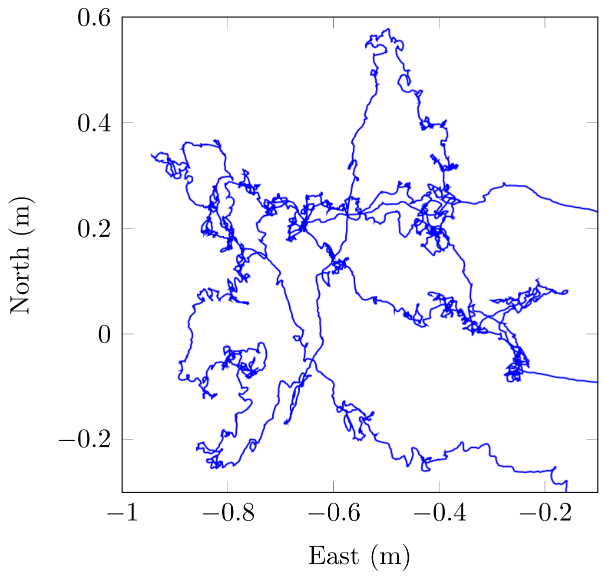


Figure 2.8: position drift in the night

2.2 Inertial navigation

Inertial navigation systems (INS) are navigation aid that uses measurements of acceleration (from accelerometer) and rotational velocity (from gyroscope) to continuously update current position. Note that initial orientation and position has to be set manually due to recursive nature of inertial navigation equations - it updates position based on previous position and current measurements.

Inertial navigation system consists of inertial measurement unit (called IMU) and processing that computes the navigation solution, see Figure 2.9. IMU contains accelerometer and gyroscope triads - 3 accelerometers and 3 gyroscopes.

According to [5], IMUs can be grouped into five broad categories: (1) marine-grade, (2) aviation-grade, (3) intermediate-grade, (4) tactical-grade and (5) automotive-grade. Marine-grade devices drift less than 1.8 kilometers a day, but they cost above 1 million euros. Similarly, price tag for aviation-grade, tactical-grade and intermediate-grade devices starts in thousands of euros per IMU.

On the other hand, automotive-grade IMUs cost a few euros a piece. They are manufactured with MEMS technology which makes them cheap, small and imprecise. They suffer with significantly larger biases and scaling errors than other grades. Their performance is so poor that [5] claims that automotive-grade IMUs are practically useless for navigation even when integrated with other systems.

Some IMUs employ magnetometer triad in addition to accelerometers and gyroscopes. When initial orientation is being determined, one can use accelerometer to find the down vector (assuming the device is not moving). This will fix orientation in two axes, leaving third axis free. Magnetometer indicates direction to magnetic North pole, hence when magnetometer and accelerometer are used together, the orientation can be fully determined. Then, in local frame, Z-axis points to Earth's center of gravity, X-axis points to magnetic North pole and Y-axis is a cross product of the other two, heading to east.

We do not pay attention to magnetometers in this study. Even they are beneficial, they are also very sensitive and they do change performance in different environments.

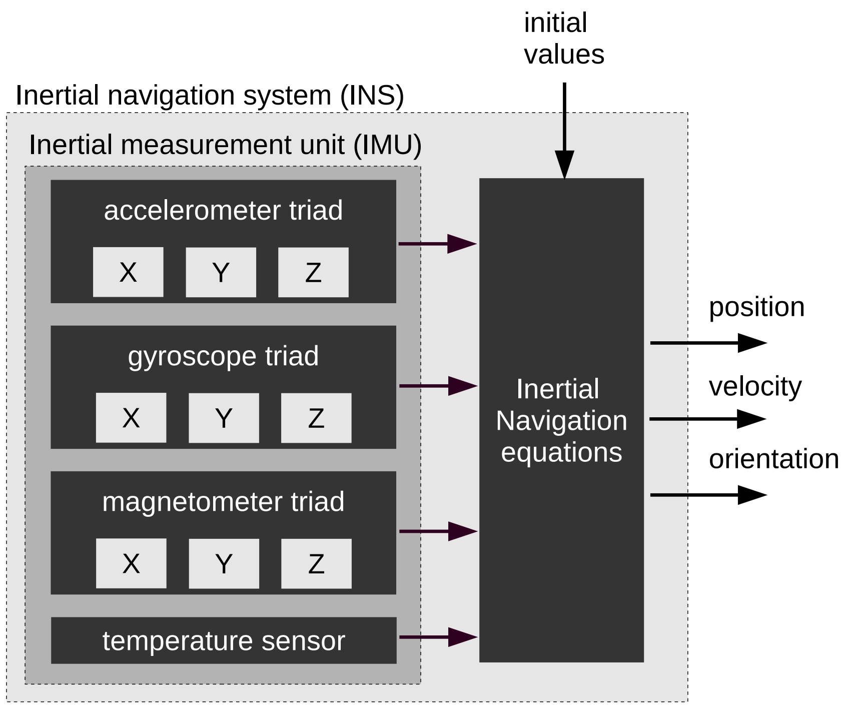


Figure 2.9: inertial navigation system block diagram

2.2.1 Accelerometer

Accelerometers measure force applied to its casing. It is not the same as coordinate acceleration. If the accelerometer is at rest on Earth's surface it will measure acceleration of 1g ($g \approx 9.805m/s^2$) upwards. This is due force that holds the accelerometer in place, not falling. It would measure no acceleration only during free fall.

Conceptually, accelerometer is a proof mass damped on springs. Position of the proof mass is what we measure. MEMS accelerometers work on this principle, with capacitive pick-off being used to measure the position - proof mass is the first electrode and casing is the second. Measured capacity is exponentially related to the gap size between the proof mass and the casing. Damping is provided by residual gas sealed in the device.

Note that a single accelerometer is measuring acceleration in a single direction. In order to measure acceleration in any direction one need three accelerometers positioned orthogonally to each other.

Force measured by accelerometer, f_a, is a sum of forces applied to accelerometer's casing. These are forces that cause movement, f_c, and force f_g that compensates for gravitational acceleration.

$$f_a = f_c + f_g \tag{2.5}$$

Newton's second law of motion says that force is acceleration scaled with mass's weight. Hence, if we assume unitary-weight mass, then force equals acceleration[8] and gravitational force f_g becomes inverted gravitational acceleration vector, $-g$. This is because f_g is the force applied by soil below the accelerometer to keep it in place instead of free-falling.

In order to get pure coordinate acceleration we have to cancel f_g by adding gravitational

[8]Even mass in the device does not have unitary weight, this assumption always holds in our model because accelerometer outputs acceleration, i.e. force applied to unitary-weight mass

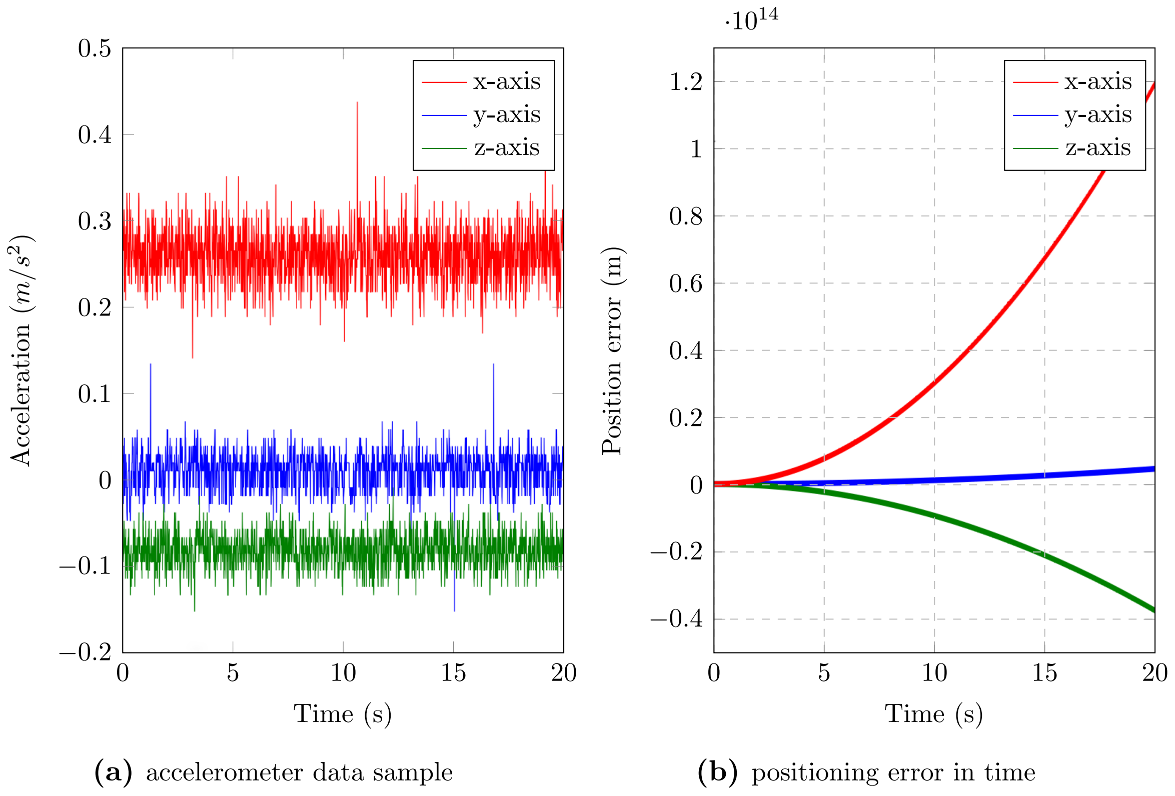


Figure 2.10: Time-series sample of accelerometer data and its conversion to position show extraordinary positioning error within first 30 seconds.

force to the measurement. As both f_g and gravitational force have same magnitude and opposite direction they cancel each other when summed. Hence, coordinate acceleration $\hat{a}$ is obtained as follows (assuming proof mass with unit weight, giving $\hat{a} = f_c$):

$$\hat{a} = f_a + g \tag{2.6}$$

Note that we need to know orientation of the vehicle in order to find vector g. This is why gyroscope is always deployed together with accelerometer. Moreover, magnitude of gravitational acceleration, g, is not universally same everywhere. Nonetheless, we can model it as constant because errors of MEMS accelerometers overbound departure of standard gravity from its mean value.

Finally, there is an easy way to remove f_g from accelerometric measurements that does not require orientation. Assuming device is stationary or accelerating slowly, then gravitational force is close to the DC component in frequency domain of the signal. So, one can apply high-pass filter to the samples which will remove the DC component. Although it is easy to implement, it also distorts the measurements and works only when acceleration changes slower than filter's cut-off frequency.

Error sources

Accelerometers are devices with a number of error sources. Zero-bias, scaling factor error and cross-coupling errors are typical. Zero-bias (or simply bias) is departure from 0 when there is no force applied to the sensor. Biases are by far the largest pain with MEMS accelerometers due to inherent double integration process when acceleration is converted

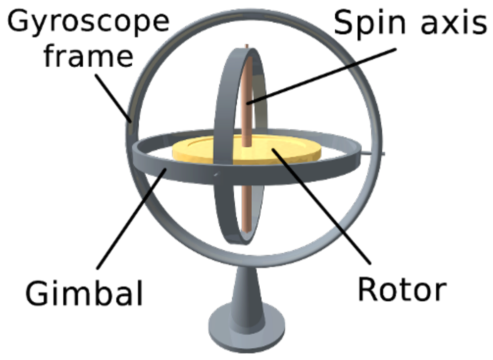


Figure 2.11: gimballed gyroscope (source: Wikimedia)

to position. By integrating small constant-valued bias we get linear slope, meaning that after first integration the bias error adds linearly in time. Consequently, velocity error is not bounded. Moreover, second integration (of the linear slope) will cause exponential growth of the positioning error.

In order to get some sense of how grave this problem is, we have made test with uncalibrated accelerometer inside a cell phone. The device indicated position change in X-axis of about $1.2 \cdot 10^{14}$ meters in 30 seconds even it actually didn't move at all, see Figure 2.10b. It is roughly 1000 times the distance from Earth to Sun!

Scale factor error is multiplicative error of measured value with respect to true forces applied to accelerometer casing.

Cross-coupling errors occur due to orthogonal misalignment of the three measurement axes. Two axes that are not precisely orthogonal are partially sensitive to forces in the same direction. Cross-coupling errors cause scale factor error as well.

2.2.2 Gyroscope

Classical gyroscope is a device that measures orientation. It uses gimballed platform and a spin wheel that rotates with high speed, see Figure 2.11. The spin axis remains upright while spinning due to its angular momentum even when the casing is moving.

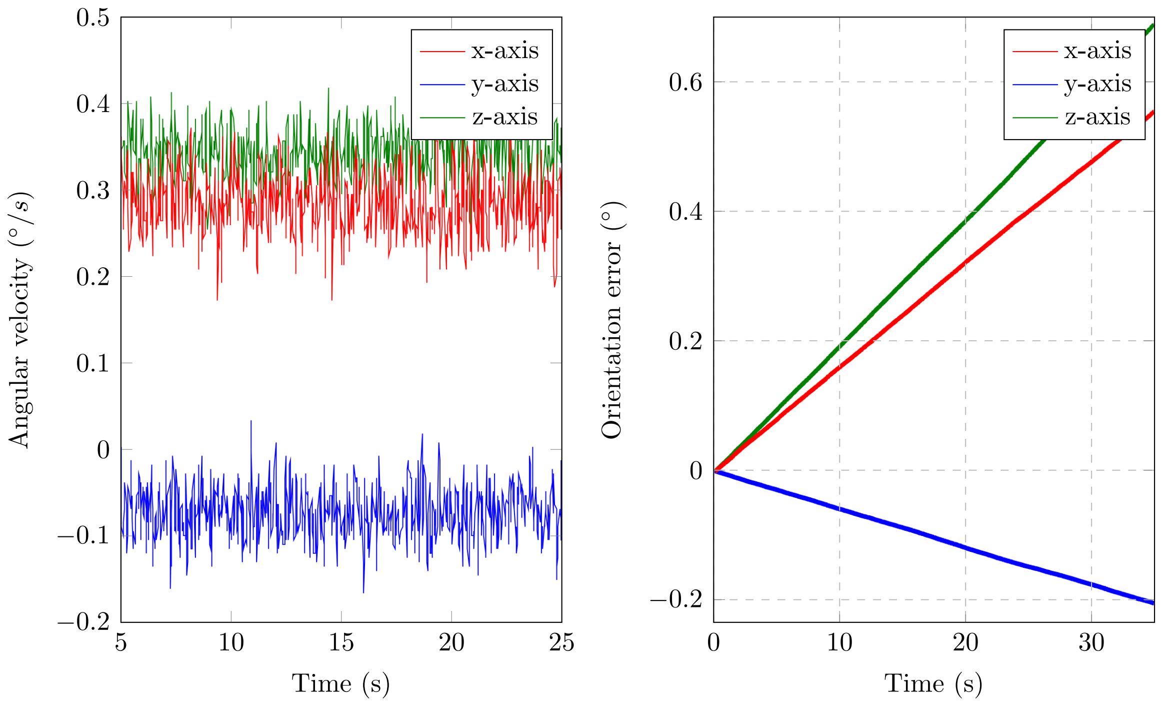
MEMS gyroscopes measure angular velocity (ω, rate of change in orientation) hence they are not gyroscopes in classical sense. The physical principle is based on Coriolis force - vibrating beam tends to continue vibrating in the same plane while its support rotates.

Three sensors in mutually orthogonal orientations are necessary to measure rotation of all three axes (pitch, yaw and roll). Bias, scale and cross-coupling errors apply and can be corrected with same model as for accelerometers, see Section 2.2.3.

Figure 2.12 shows gyroscope test results (when stationary). We can observe similar biases like with MEMS accelerometers. Nonetheless, the problem is not as severe because error in orientation grows only linearly in time. Tested gyroscope shown worst-case error of $0.6°$ in 35 seconds, see Figure 2.12b.

Determining orientation from gyroscope measurement

Orientation (also called attitude) can be represented in various ways: as rotation matrix, Euler angles or quaternions. In here, we use rotation matrix as it is simplest. Rotation



(a) MEMS gyroscope output sample **(b)** orientation error in time

Figure 2.12: Time-series sample of gyroscope data and derived orientation error propagation in time.

matrix is sometimes referred to as "coordinate transformation matrix", but here we avoid using this term as it can be used interchangeably for both rotation and translation[9].

Lets suppose we have current orientation expressed as rotation matrix $C_r(t)$ and we wish to update it with new angular velocity measurement $\omega = (\omega_x, \omega_y, \omega_z)$. Then, we can compute updated orientation $C_r(t+\tau)$ using

$$C_r(t+\tau) = C_r(t)e^{\Omega\tau}; \quad \Omega = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \tag{2.7}$$

Where τ is gyroscope measurement update period. By expressing (2.7) as truncated power series we get simplest form of orientation update:

$$C_r(t+\tau) \approx C_r(t) \begin{pmatrix} 1 & -\omega_z\tau & \omega_y\tau \\ \omega_z\tau & 1 & -\omega_x\tau \\ -\omega_y\tau & \omega_x\tau & 1 \end{pmatrix} \tag{2.8}$$

This version applies to inertial (ECI) frame only, version for ECEF and local frames can be found in [5] together with detailed derivation of (2.8).

2.2.3 Error models

Lets assume we have acceleration measurement $\hat{a}$ and angular velocity measurement ω. Model that would correct for bias, scale and cross-coupling errors is essentially the same

[9]Translation is achievable using homogeneous coordinates

for both accelerometers and gyroscopes.

Lets note scale factor coefficient $\alpha = (\alpha_x, \alpha_y, \alpha_z)$, bias $\beta = (\beta_x, \beta_y, \beta_z)$ and cross-coupling coefficients m_{xy}, m_{yz} and m_{xz}. Then, acceleration error can be expressed with following formula

$$\hat{a} = \begin{pmatrix} 1+\alpha_x & m_{xy} & m_{xz} \\ m_{xy} & 1+\alpha_y & m_{yz} \\ m_{xz} & m_{yz} & 1+\alpha_z \end{pmatrix} a + \beta \tag{2.9}$$

Where $\hat{a}$ is measured acceleration and a its real counter part. This model requires 9 parameters - three for bias, three for scaling and three for cross-coupling. Note that random noise is ignored. Correction formula can be found simply by rearranging (2.9).

$$a = \beta - \hat{a} \begin{pmatrix} 1+\alpha_x & m_{xy} & m_{xz} \\ m_{xy} & 1+\alpha_y & m_{yz} \\ m_{xz} & m_{yz} & 1+\alpha_z \end{pmatrix}^{-1} \tag{2.10}$$

Scale and cross-coupling coefficients can be determined once and used from there on. When comes to biases, our experimentation has shown that they undergo random walks. Literature [5] suggest that there are 3 types of bias variations: (1) device-to-device variation that can be corrected in factory, (2) run-to-run variation and (3) in-run variations. Latest manifests as slow bias drift with correlation time in minutes. Bias dependence on temperature was observed as well, perhaps this has major impact on run-to-run and in-run bias variations. Practical implementations should model temperature dependence as well. Typically, it suffices to estimate correction coefficients on two distinct temperatures and use linear fitting to estimate correction coefficients on any temperature.

Bias correction is not easy due to run-to-run and in-run variations. Nonetheless, it remains crucial due to exponential growth of positioning errors originating from biases. Applications often estimate bias whenever possible. For example, biases can be estimated with Kalman filter that is activated when vehicle is stationary. This is sometimes called zero-velocity updates (ZVUs).

A method that estimates all 9 parameters for accelerometer is presented in [4]. Authors use Kalman filter and a special platform that can put device in any orientation. They have shown that by putting IMU in some 30 approximately known orientations the filter is capable to estimate all 9 parameters.

2.3 Odometers

Cars were traditionally fitted with vehicle speed sensor in transmission shaft. Nowadays, each wheel has its own speed sensor and some hi-end cars are equipped with steering angle sensor as well.

With information from all four wheel speed sensors one can determine vehicle speed and yaw (turning rate). For appropriate model see [5], it provides precise vehicle speed and low-resolution yaw. This is because yaw is determined from difference in wheel speed between left and right front wheels. As these two wheels copy different radiuses, the outer wheel moves slightly faster. If we take into account that wheel speed sensor has finite resolution together with the fact that wheels rotate only with slightly different speeds,

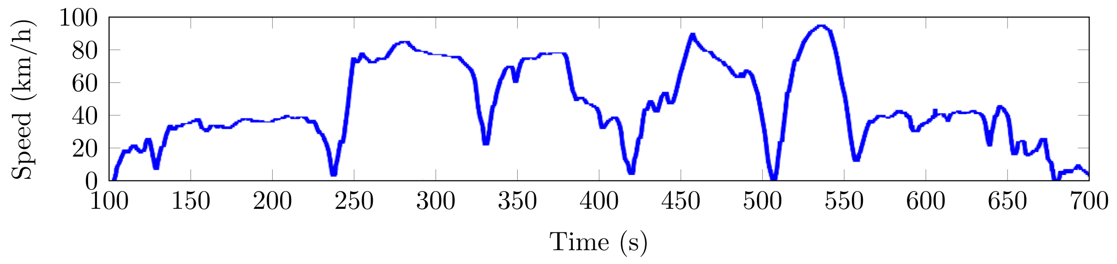


Figure 2.13: velocity profile

consequence is that yaw inherits low resolution. Note that with steering angle sensor, it should be possible to obtain high precision positioning in horizontal plane.

Wheel speed sensor measures angular velocity of the wheel. So, wheel radius is required in order to convert it to speed. This is not constant, though, because of tire wear and pressure. Induced error can be observed with Kalman filter that uses GNSS to estimate scale factor that corrects the velocity, see Section 2.3.1

Odometry-related information can be read through standard OBD-II diagnostic interface. Nonetheless, only vehicle speed is available through legislated OBD. Wheel speed sensors on each wheel and steering angle sensors might be available as well, but only through manufacturer-specific access codes. In practice, these have to be reverse engineered for a specific vehicle in use.

Legislated OBD-II services are specified in ISO 15031-5 ("Road vehicles - Communication between vehicle and external equipment for emissions-related diagnostics - Part 5: Emissions-related diagnostic services"). Vehicle speed is available via PID (parameter identifier) 0x0D under service $01 - request for current powertrain data. Its value is 8-bit unsigned integer with range from 0 to 255 km/h and resolution of $\approx$ 0.28 m/s (1 km/h). See Figure 2.13 for sample data obtained through OBD-II interface.

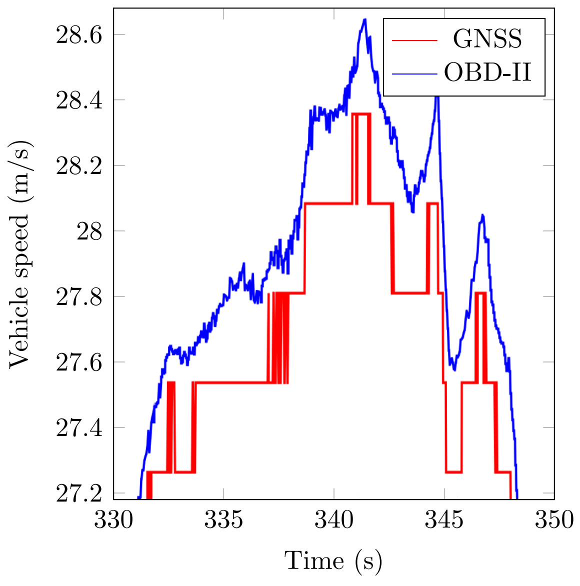
2.3.1 Estimating scale error in vehicle speed

As was stated above, wheel speed sensor precision suffers with scaling error due to slightly variable wheel radius. This is multiplicative error that changes slowly over the lifetime of a tire. It can be expressed simply as

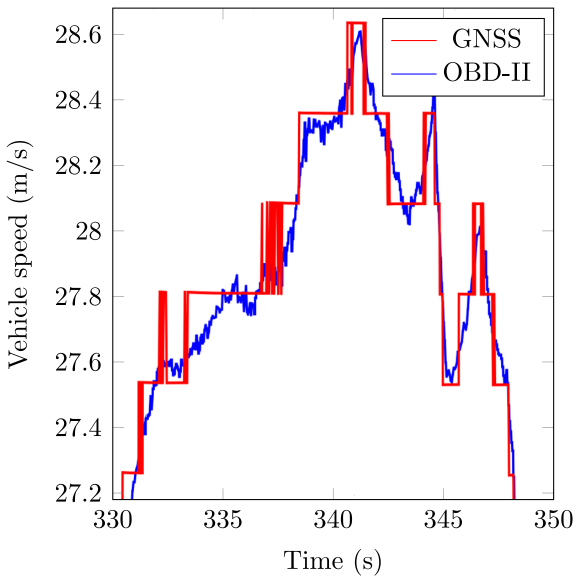
$$s = \delta_w \hat{s} \tag{2.11}$$

where s is real speed, $\hat{s}$ is speed read via OBD-II and δ_w is scale correction coefficient. We can use Kalman filter to continuously estimate δ_w based on difference between speed reported by OBD-II and satellite navigation. Speed based on GNSS is noisier, especially when the vehicle is moving slowly, but it doesn't share same error modes as odometry-based sensor making it a good complement for this use case.

Kalman filter [26] implementation is simple as we estimate state of a single parameter δ_w. Hence, all matrices and vectors reduce to scalars. State matrix A reduces to 1 and process noise covariance matrix to 0, taking into account that δ_w is virtually constant. Measurement matrix H reduces to current speed reported by OBD-II and measurement vector z becomes current speed observed by satellite navigation system. Finally, measurement covariance matrix reduces to variance of velocity as reported by GNSS receiver. Note



(a) before correction (detail)



(b) after correction (detail)

Figure 2.14: Comparison of vehicle speeds before and after correction.

that Kalman's measurement model ($z_k = Hx_k$) becomes our error model (2.11) in this setup. See Listing 2.1 for implementation in MATLAB.

We have tested this filter on real test drive data, it estimated $\delta_w = 1.0115$. As you can see on Figure 2.14 the filter clearly corrected gap between GNSS and OBD-II speed measurements. Note that GNSS data were not filtered to avoid Kalman filter cascading. See Figure 2.15 for comparison of performance with filtered and not filtered GNSS data.

```
data = dlmread("hlupenov-stribro-obd2+gps.csv", "\t");
vel_obd = data(2:end,2)*(1/3.6);
vel_gnss = data(2:end,11)*(1/3.6);
vel_gns_var = (max(data(2:end, 19:21)')').^2;
time = data(2:end,1);

A = 1;
R = vel_gnss_var;
H = vel_obd;
z = vel_gnss;
B = Q = 0;
P = K = xm = delta_sf = u = Pm = zeros(size(data, 1)-1,1);

P(1) = 1;
for k=2:size(data, 1)-1

    % projection
    xm(k) = A*delta_sf(k-1)+B*u(k-1);
    Pm(k) = A*P(k-1)*A'+Q;

    % correction
    K(k) = Pm(k)*H(k)'*inv(H(k)*Pm(k)*H(k)'+R(k));
    delta_sf(k) = xm(k)+K(k)*(z(k)-H(k)*xm(k));
    P(k) = (1-K(k)*H(k))*Pm(k);
end
```

Listing 2.1: implementation of δ_w estimator

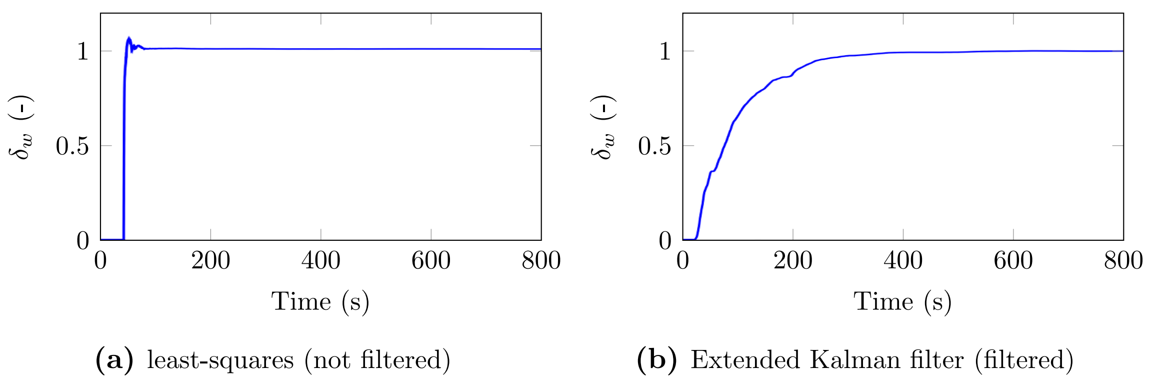


Figure 2.15: Dependence of δ_w estimator performance on GNSS data filtering.

2.4 Barometric altitude meter

Barometric altitude meter uses barometer to measure ambient air pressure, p_a. Height above mean sea level, h, can be determined using US Standard Atmosphere 1976 model. It requires reference atmospheric pressure p_r, reference temperature t_r (in Kelvins) and reference height h_r. These can be supplied from a station nearby, if available. Otherwise standard values $p_r = 101325\text{kPa}$ and $t_r = 288.15\text{K}$ at $h_r = 0$ are assumed.

$$h = \frac{t_r}{0.0065}\left[\left(\frac{p_a}{p_r}\right)^{\frac{-1.86615}{g}} - 1\right] + h_r \tag{2.12}$$

This model is valid for heights limited to about 11 kilometers. Note the gravitational acceleration, g. In simplest case one can use $g = 9.80665$, otherwise gravitational model such as EGM96 (Earth Gravitational Model 1996) can be used if higher precision is required. Then, g becomes function of latitude and longitude.

Barometric pressure vary with weather causing biases of several hundred meters in standalone version. It is due to departure of reference ambient pressure, p_r and temperature t_r at mean sea level from the assumed values. If reference is used, then precision depends on distance from the reference station and on age of information it provided.

Sonic booms cause measurement outliers, hence outlier removal and data validation techniques are important for robust applications. Measured pressure can also undergo significant changes in short time when weather front is passing by in vicinity of the sensor. These changes can be corrected only with differential data from reference station.

Modern MEMS barometers have resolution of about 2 Pascals. Sigma-delta ADC converts with high oversampling factors are used. It allows sub-meter precision for the price of slow operation, typically about 1 Hz.

The height information can be used as an aid to GNSS. By supplying height externally to receiver's navigation processor we can compute position fix with only three instead of four satellites. Note that this works only if height is referenced to EGM96 geoid, not to standard ellipsoid model. The difference between them can be as much as 100 meters.

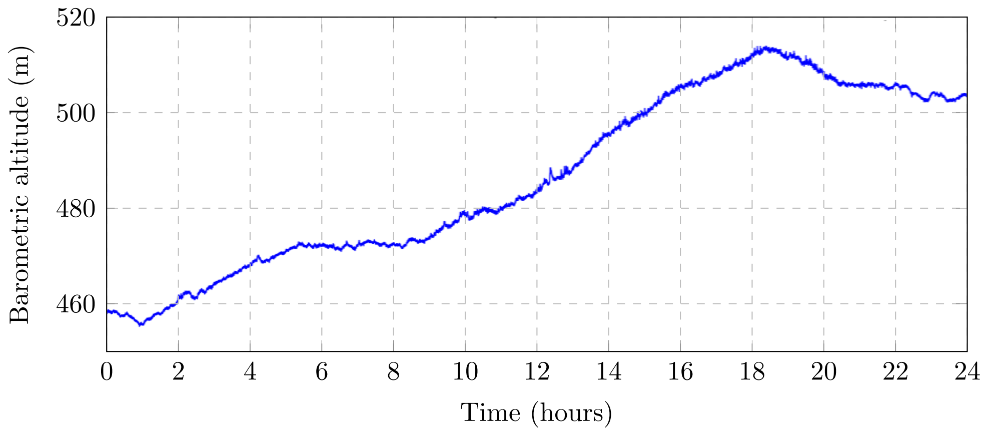


Figure 2.16: Altitude measurement sample (midnight to midnight).

2.5 Map-matching algorithms

By map-matching we mean problem of converting information provided by vehicle's navigation system onto position on a map. By navigation system we mean any system (satellite based, inertial-based, integrated, filtered, unfiltered, etc..) that provides three dimensional position in ECEF frame (optionally with other information such as heading and velocity).

In context of road-vehicle navigation, the map-matcher is instrumental as it converts Earth-referenced position to position on a known road network see Figure 2.17.

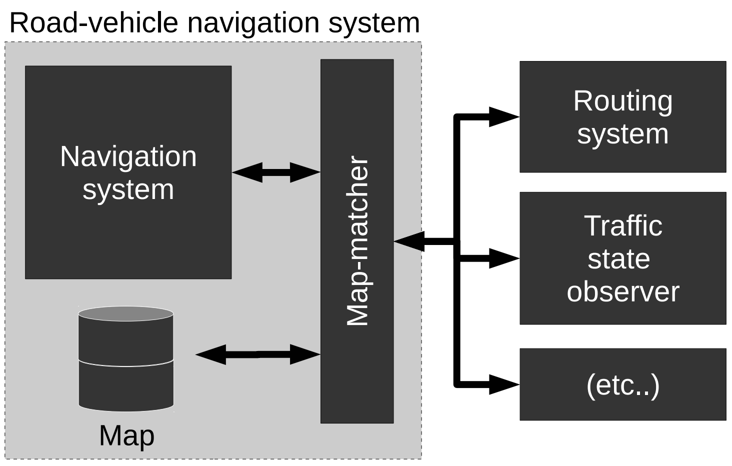


Figure 2.17: Map-matcher as a gateway between low-level and high-level subsystems.

We assume to have only minimal representation of the road-network. It shall be specified by graph-like structure with only position in terms of latitude and longitude assigned to each node. The altitude coordinate is omitted as we consider only classical 2D maps for matching.

2.5.1 Related work

To our best knowledge there is no work published so far that would analyze the problem of map-matching for road navigation in general sense. Indeed, there are over 30 map-matching

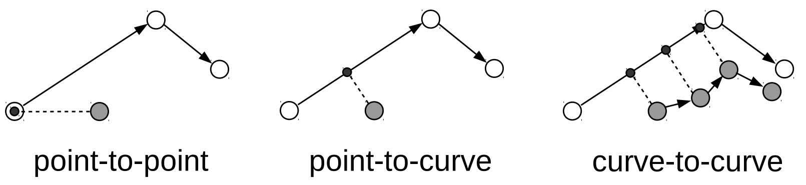


Figure 2.18: Different principles of geometric algorithms. Grey circle is measured position, white circles are road network nodes and black dots are matched points on the road network.

algorithms published so far but they mostly relate to experiments in specific environment and reports often don't consider their general applicability [3].

Comprehensive review of map-matching algorithms published between 1989 and 2007 is given in paper by Quddus et al. [3]. Author categorizes the algorithms into four groups: geometric, topological, probabilistic and advanced.

- Geometric algorithms [13] are generally based on geometric closeness. Simplest approach is so-called point-to-point matching that, given a position from navigation system, searches for closest node on the road network graph. More advanced approach is point-to-curve matching that searches for closest point along the edges of the road-network (given that edges represent so-called road centrelines). Finally, curve-to-curve matching compares shape of the road network with shape of position trace. This overcomes shortcomings of previous "point-to-something" approaches as it takes into account past measurements. Nonetheless, it is sensitive to outliers. Differences between the three approaches are depicted on Figure 2.18.

- Topological algorithms [12] take into account both road network geometry and topology. Hence, connectivity, contiguity and, in some cases, direction of the links is taken into account. This has been shown to give better results than pure geometric algorithms as the solution is more constrained.

- Probabilistic algorithms [14] use uncertainties related to positioning system measurements to devise error regions (circles, ellipses, spheres or ellipsoids). This is a powerful technique, but it relies on the positioning system to provide realistic uncertainties. It is not always the case [5]. The algorithm use error regions to identify all road segments on which the vehicle can be traveling and computes related probabilities for each. Most probable (or most-likely) road segment is identified as correct one. Probabilistic algorithms have been shown to perform well.

- Advanced algorithms [7, 9, 6] relate to methods that use Kalman filter, particle filters, fuzzy logic, belief theory, etc. Fuzzy logic algorithms have been told to perform slightly better than probabilistic ones [6] but as these algorithms are rule based, performance in varying environments is questionable. Algorithms that use Kalman filter require the underlying positioning system to provide unfiltered data as one of the main assumptions of Kalman (normally distributed zero-bias errors) is breached otherwise. It is feasible to use filtered data but uncertainties have to be boosted to avoid stability issues and consequently the Kalman gain gets smaller and filter converges slower.

Although the categorization above provides concise roadmap for different methods proposed so far, algorithms usually combine ideas from different categories trying to get the best from each.

Methods that use multiple hypotheses stem from research conducted mainly in 1970's for multiple target tracking. Pyo et. al [7] used Reid's algorithm [8] for multiple hypothesis multiple target tracking and restated it as single target tracking problem. Although Reid's algorithm was designed to track air vehicles (airplanes, missiles) with radar measurements and on-board transponders, Pyo has successfully shown that it can readily be used for map-matching.

Interesting results were reported by Gustafsson et. al [9] who used particle filters (recursive Bayesian estimation) to find current position based solely on a map, approximate starting position and vehicle's transnational velocity. They report that the filter performs comparably with satellite navigation in rural areas and better than satellite navigation in urban areas. Note that achievable precision with GPS C/A code at the time of writing of their paper (2002) was considerably lower.

2.5.2 Map-matching problem analysis

We consider road-network represented as directed graph. Each node have assigned position in known frame and each directed edge represents centreline of linear road segment between position of the two nodes it connects. Structure of the graph together with the position of the nodes constitute a map.

For map-matching purposes we neglect any additional information such as street name, type of the road, etc. Furthermore we do not make any distinction between junctions and waypoints on curved road. Note that as the graph is directed, information whether we are on one-way or two-way road is inherently integrated in the directed graph structure.

Maps are classically described by means of curves and arcs. Nonetheless, arcs in maps are piece-wise linear. By dropping any diversification between waypoints that shape the arcs and junction nodes that connect different roads, we encoded curvature of the road directly into the structure of the graph, leaving only linear road segments in the map.

Navigation system is, for the purposes of this text, any system that periodically reports current position and related uncertainty in ECEF frame. In the case where supplementary outputs are reported (velocity, heading), then we require them to have their uncertainty reported as well.

Map-matching is a process of matching positioning data to points along line segments described by directed edges of the road network graph such that discrepancies between reported position and position on the map are minimized. x Hypothesis is, for the purposes of this text, any sequence of contiguous edges on the graph without any restriction on its length.

Rest of this section is concerned with various conclusions of our analysis. Each claim is described in detail and defended.

Use only minimal set of assumptions

Map-matching algorithm that depends on many assumptions or ad-hoc rules can not be robust as the road-network and positioning system properties vary with changing environment. For example, satellite based navigation systems are well known to perform poorly in urban areas where multipath is common and direct satellite-antenna visibility is limited. Yet, many proposed algorithms [10, 11, 12] define fixed-sized uncertainty radius around measurements and assume that it bounds all the errors.

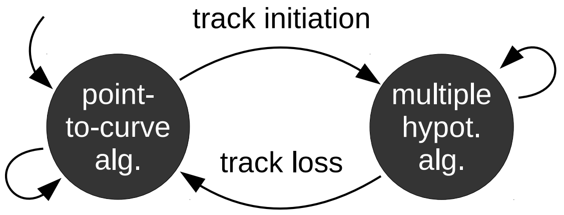


Figure 2.19: State machine that governs which algorithm is active enables fast failure recovery.

One has to be careful to keep assumptions he makes at minimum to sustain robustness. Optimal map-matching in this sense would not require any tuning.

Maps are only models of reality that change (due to road constructions, closures, etc..) and even if they would not, the graph-like model is useless in 2D areas like parking lots, for example. Hence, one cannot assume that maps are error free. On the other hand, errors in maps cannot be effectively modeled. Apart of simple errors such as discretization of curved road by piece-wise linear arc or bias of the road centrelines, most painful errors stem from various simplifications (i.e. missing road segments) and artifacts (i.e. road segment takes place in a map instead of a parking lot that actually lays there). These cannot be effectively modeled and so we cannot infer realistic uncertainties about the errors inside the map.

Fast failure recovery

All feature-matching systems give occasional false matches [5]. With map-matching this emerges in form of occasionally taken wrong turn (usually only for a short period of time; it is typical for Y junctions) and by track loss. Hence, algorithms shall be able to recover from track loss as quickly as possible.

With multiple-hypothesis algorithms this problem is remedied with two map matching algorithms existing in parallel [10, 12]. First algorithm is geometrical (point-to-curve). It is active only when no hypothesis is available - during initialization and track loss. It identifies possible road segments on which the vehicle might be traveling. When at least one road segment is identified, then the main algorithm takes control. It can be implemented as simple state machine, see Figure 2.19.

Problem arises when using point-to-curve algorithms. They don't cooperate well with graph representation of the road network as closeness criteria on graph are not geometric. The algorithm have no choice but to search first for nodes in vicinity (since position along the linear road segments constitute infinite set). Then, having the list of candidate nodes, the algorithm can create a list of candidate edges by looking at their adjacency's. Finally, then the algorithm can apply its geometrical closeness criteria to the list of edges in vicinity. However question remains how large is the search radius for nodes that would give certainty that a complete set of edges in vicinity is found. Indeed, it might happen that some edges will remain hidden. Imagine situation on Figure 2.20. The long, straight edge is close to reported position but still remain hidden for the point-to-curve algorithm because search radius for the nodes wasn't large enough.

Typical remedy for this problem is artificial threshold parameter which was empirically tested to perform well with the map, algorithm, data structures, and computational capacity at hand. Better solution is to make the search radius as large as longest linear road segment in the map. That gives certitude that no road segments remain hidden. When the longest road segments are unfeasibly long, then redundant nodes can be added on these

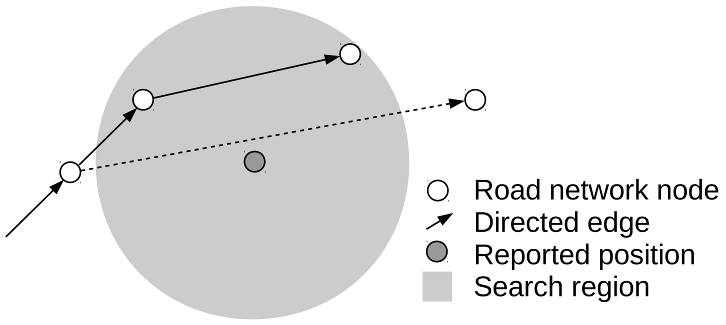


Figure 2.20: The dotted line remains hidden for point-to-curve algorithm because its enclosing nodes are not in vicinity of the measurement. This situation results with incorrect match.

segments at a little cost.

Recursive nature

Having recursive instead of iterative algorithm is desirable as it off-loads the main processor and enables the algorithm to run on smaller and cheaper embedded system.

One of the main advantages of multiple hypothesis technique is that it is recursive in nature - new hypotheses are easily generated by branching already existing ones along directed edges of the road network graph (by taking advantage of its topology).

Maximum likelihood estimation is also more desirable than classic probabilistic approach as likelihood functions that add error terms can be easily designed recursive.

No filtering

We claim that there should be no filtering applied on input data. This is so because navigation system is always in better position to filter the positioning data than map-matcher. Filtering data in map-matcher data practically means that assumptions have to be made about underlying positioning system. That is undesirable for generally applicable algorithm.

Often, map-matching algorithms try to enhance data provided by positioning system with the map. Nowadays this looses its significance as cheap satellite navigation systems are capable of sub-meter precision in some conditions. Putting this together with the fact that many errors in maps cannot be statistically modeled drives us to conclusion that, for common cases, classical maps are not suitable to enhance position. An exception to this would be when positioning data are known to drift (which is the case with inertial navigation for example).

Mind the uncertainty

Robust algorithms shall consider uncertainty regions around reported position. These regions are usually modeled as circles or ellipses (spheres, or ellipsoids) so simple geometry can be applied to detect intersections with road segments. They provide convenient way for choosing search regions. The size of these regions can be computed in a way that it overbounds the measurement error with probability above given level (assuming normally distributed errors).

Uncertainties can be obtained from RMS position errors reported by GNSS receiver. RMS errors are simply statistical measures of magnitude of the errors. They relate to mean and variance with:

$$x_{rms} = \sqrt{E[x]^2 + \sigma_x^2} \tag{2.13}$$

Taking into account that Kalman filter and least squares estimator (both can be used by GNSS receiver) are zero-bias estimators, its easy to see that RMS error of the position equal to its standard deviation σ_x.

Where uncertainties are not available, they can be often modeled. Moreover, where navigation system integrates measurements from different sources, the integration is usually carried out with error-state Kalman filter. Such filter directly reports residual uncertainty.

2.6 Integration architectures

Number of navigation systems and navigation aids were discussed so far, but possibilities of their integration into one system were mentioned only marginally. This section discusses various ways how these systems can be combined together.

Satellite navigation systems provide Earth-referenced positioning with sustainable long-term accuracy, but it cannot be relied upon to provide continuous navigation solution. On the other hand, inertial navigation systems are able to provide continuous navigation solution, have high update rate and high short-term precision. Nonetheless, its accuracy degrades in time as errors in inertial navigation equations are integrated in time. System that combines satellite navigation with inertial navigation can overcome these drawbacks and provide continuous high-precision navigation solution.

The integration algorithm is typically based on Kalman filter. Inertial navigation system is always referential, GNSS is used to correct it. This is because INS provides uninterrupted navigation solution while GNSS-based position availability depends on signal availability. Open-loop and closed-loop implementations are possible. When open-loop is used, the integration algorithm estimates INS position error, uses it for correct the output, but do not feed back the errors to INS. With closed-loop approach, the corrections are fed back to INS where they are used to correct inertial navigation solution itself. This allows to estimate bias and scaling errors as well as linearization-induced errors in inertial navigation equations. Where low-grade INS is used, open-loop implementation is not suitable because raw INS navigation solution would quickly become useless.

Literature [5] describes four approaches for GNSS/INS integration. These are uncoupled, loosely coupled, tightly coupled and deeply coupled architectures. The difference is in how "deeply" is inertial navigation embedded in GNSS receiver. Most simple is uncoupled architecture. It uses GNSS receiver to reset INS-based position on periodical basis. This provides reliable position only when higher-grade INS system is used and therefore is not suitable for our needs.

Loosely coupled architecture (Figure 2.21) can work with any GNSS receiver and any INS system, which is its main advantage. It is an example of high level integration where GNSS and INS are two separate subsystems.

Main problem with loosely coupled architecture is that it cascades Kalman filters (first is in the navigation processor, see Section 2.1.2, and second in the integration algorithm). One of the assumptions Kalman filter makes about its inputs is that they are not time-

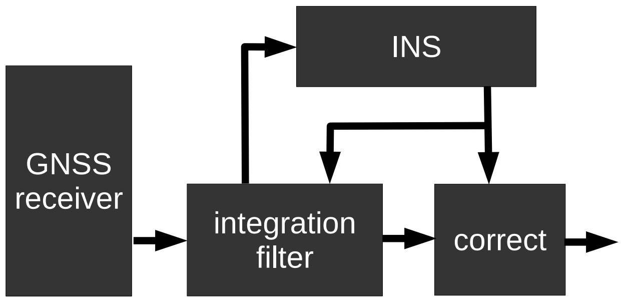


Figure 2.21: loosely coupled integration architecture

correlated. Filtered data are, in nature, time-correlated so inputs of the second filter do not meet this assumption. Consequently, Kalman gain must be lowered by increasing related uncertainties, otherwise the filter might run into stability problems. Lower Kalman gain means that the filter will be slower.

Filter cascading can be overcome by using least square estimator in navigation processor instead of Kalman filter. This, however, solves nothing as unfiltered positioning solution is noisy (see Figure 2.2a). Hence, uncertainties have to be boosted anyway causing sub-optimal performance.

Tightly coupled architecture replaces navigation processor in GNSS receiver with specialized processor that incorporates INS data into the main fusing filter. This avoids problems related to filter cascading, but it requires specialized GNSS equipment that can provide raw pseudo-ranges and pseudo-range rates from ranging processor. Standard GNSS receivers cannot do that, although, it is possible to buy standalone receivers capable of providing this information alongside its own navigation solution.

Finally, deeply coupled integration replaces both ranging and navigation processors. In practice this means that own specialized GNSS receiver has to be developed. Benefits are substantial, though, as INS can aid with satellite tracking as well as with positioning. Deeply coupled integration architecture is optimal [5].

2.7 Proposed road-vehicle navigation system

In this section we draw a number of conclusions based on our analysis and propose architecture of road-vehicle navigation system that meet required performance goals. See Figure 2.22 for block diagram. Note that in here we provide only high-level outlook of the system we propose, implementation is discussed in following chapters.

2.7.1 Hardware

Originally, whole road-vehicle navigation was supposed to be implemented on Android devices as they are usually equipped with both satellite navigation receiver and inertial measurement unit. Most important observation we made is that specialized hardware will be necessary. Smartphones are not equipped with hardware that is sufficient to create navigation system with guaranteed performance. We observed that especially accelerometers and gyroscopes have low-resolution and unstable update rate (due to non real-time kernel running on smartphone). Moreover, applications have access to GNSS/INS measurements

		min	typ	max
accelerometer	full-scale range	±2g	-	±16g
	sensitivity	0.48mg/LSB	-	0.061mg/LSB
	temperature change	-	±0.75mg/°C	-
gyroscope	full-scale range	±250°/s	-	±2000°/s
	sensitivity	0.061°/LSB	-	0.0076°/LSB
	RMS noise	-	0.06°/s-rms	-

Table 2.1: IMU-9150 performance (1g=9.80665m/s^2)

only through high-level interfaces, having no information about underlying processes such as filtering, on-line calibration, etc..

When comes to satellite navigation, we should use specialized GNSS receiver that can receive both GPS and GLONASS systems in order increase satellite visibility. Furthermore, receiver should provide raw measurements (pseudo-ranges and pseudo-range rates) so we are able to implement tightly coupled integrated navigation system. Finally, it has to provide detailed information about its state[10] and preferably even implement both least-squares and Kalman filter based position estimators.

We found device which is readily available on market and posses properties mentioned above - module NV08C-CSM from NVS technologies AG [15]. It is combined GPS/-GLONASS receiver that features proprietary interface called BINR [17]. This interface provides access to internal information such as raw data from ranging processor, detailed description of all tracked satellites, etc.. Such level of insight into its operation is not possible with standard NMEA-0183 interface. Also, it is possible to feed the device on secondary UART channel with differential corrections in RTCM-SC104 format. There are web services that provide corrections directly in this format so one can open TCP/IP socket and just forward incoming data to the receiver.

Inertial measurement unit based on MEMS has to have high update rate (>50Hz) and high resolution. Literature indicated that MEMS accelerometers and gyroscopes are practically useless for navigation even when integrated with other systems, so we need to use cutting edge device in order to try to prove this claim outdated. We have compared performance of many devices according to their sensitivity and noise characteristics. Device called MPU-9150 from InvenSense had best performance (on paper). It combines 3-axis accelerometer, 3-axis gyroscope and 3-axis magnetometer in a single package. Main properties according to which we compared it with others are listed in Table 2.1.

Odometry-based navigation system is not readily available with common vehicles, only low-resolution vehicle speed can be read through standard interface. This is still valuable information as it can be used as navigation aid in various ways. Most notably, vehicle speed in conjunction with gyroscope can form standalone dead-reckoning navigation system excluding use of accelerometer altogether. This is beneficial because position error stemming from cumulative integration of the vehicle speed has errors growing linearly in time while for accelerometer they grow exponentially (see Figure 2.10b). Moreover, if we add filter to estimate scale error in the vehicle speed (see Section 2.3.1) we can eliminate major error source and improve performance of dead-reckoning navigation system. Note that dead-reckoning system can be integrated with GNSS in the same way as INS (see section 2.6).

Barometric altitude meter cannot improve performance of our system. It is useful only

[10]Non-standard communication interface is required for that.

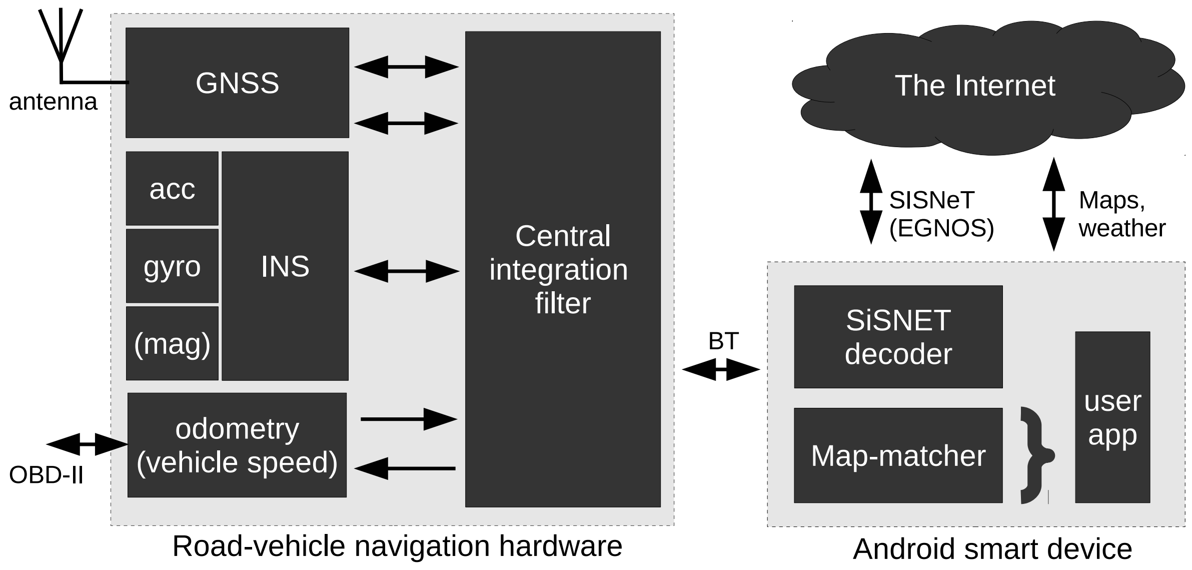


Figure 2.22: road-vehicle navigation system

when referential system provides differential corrections, but it cannot be guaranteed as there is no such network of referential barometers deployed today (to our best knowledge). Bias errors can be as high as several hundred meters without the corrections. This problem can be evaded by taking altitude rate of change. Nonetheless, our analysis have shown that this information has low quality due to slow update rate (1Hz) and measurement noise that is high when differentiated.

There might be a way to improve performance of barometric altimeter as some weather forecasting web services provide information about atmospheric pressure and temperature. Atmospheric pressure is either related to mean sea level or there is altitude information added to the data. It might be possible to use this information to mitigate problem of large measurement bias common with standalone barometric altimeters. This is topic requires further investigation.

2.7.2 Software

Considering our analysis of integration architectures (section 2.6) we chose to build our road-vehicle navigation system on tight GNSS/INS integration. It does not suffer with statistical problems related to filter cascading while it is not necessary to build specialized GNSS receiver to get it working.

We have also considered where to implement inertial navigation equations and integration filter. Originally, we planned to do it on Android device, but our tests have shown that Bluetooth[11] communication channel is quickly overloaded when GNSS raw data and IMU measurements are fed through.

Problems with timing also need to be considered. Android smartphone uses best-effort scheduler meaning that it is not real-time capable. For this reason, all measurements we send to the Android device must be time-stamped. This allows to compare age of the data between each other, but as the Android device and our hardware does not run on synchronized clocks we cannot devise age of the data in Android device. This is a problem

[11]We use Bluetooth to connect our hardware with Android phone, see Chapter 3.

particularly for prediction phase of the Kalman filter.

Clock synchronization is not easily achievable between the two devices due to non-realtime behavior of Android device and unknown behavior of Bluetooth stack. Nonetheless, our hardware can easily align its time reference with UTC using GNSS receiver. Unfortunately, Android devices are not precisely synchronized with UTC time reference (even if the device carry all necessary hardware).

Consequently, we have to implement inertial navigation equations and integration filter in the embedded system. However, the map-matching algorithm can be implemented on Android as it only requires latest position to provide most up-to-date position on a map. Considering that map-matching algorithm requires substantial road-network database, it is easier to implement it on Android device.

We also had to decide what maps and mapping resources to use. We have chosen to use OpenStreetMaps because of high resolution, friendly licensing and simple data format. Precision of this map is not guaranteed as it is developed by open community, but our analysis of map-matching algorithms have shown that error-less maps do not exist anyway, so our algorithm need the ability to cooperate with them whatever map we use.

Finally, we should make use of differential corrections to improve overall performance. SiSNET service provides EGNOS data stream over the internet, which is optimal as it allows us to make use of SBAS services even when EGNOS satellite is not tracked by the receiver.

Chapter 3

Embedded system

Previous chapter finished with a proposal of road-vehicle navigation system based on thorough analysis. One of the main conclusions was that specially designed embedded system will be necessary to collect and filter navigational data. This chapter is concerned with design of that embedded system.

See Figure 3.1 for block diagram of the printed circuit board (PCB). Subsystems and their interconnection is shown. Note that signal names are identical to the ones used in schematic (see Appendix C, so the diagram can be understood as more abstract version of schematic showing how different subsystems are connected to each other. These subsystems can be organized in three broad categories:

- *Sensors* that collect navigation-related information. Inertial measurement unit MPU-9150, barometric altitude meter MPL3115A2 and GNSS receiver NV08C-CSM belong to this category. Note the barometric altitude meter - we have deemed it useless in our analysis yet here we use it. It is so because this board was designed to run the tests presented in analysis as well.
- *Communication interfaces* that exchange data with other devices. Bluetooth, CAN-bus and USB belong to this category. Bluetooth is used to interface with Android device, CAN-bus provides means to communicate with the vehicle (OBD-II interface) and USB is there as a backup in cases when Bluetooth bandwidth proves insufficient.
- *Microcontroller* that collects, filters and forwards measurements. We chose device named AT91SAM7X512 from Atmel featuring ARM7 processor. It has 512kB of Flash EEPROM memory, 256kB of SRAM and all peripherals we need (CAN, I2C, USB, JTAG, DBGU[1] and two UARTs).

We have developed peripheral drivers for CAN, I2C, DBGU, UART and built device drivers for MPU-9150, MPL3115A2, NV08C-CSM and OBD-II on top of them. Peripheral drivers were designed optimally in terms of processor load. I2C interface got asynchronous drivers with simple message prioritization and the two UARTs with DBGU use direct memory access (DMA) to transfer their payload. Device drivers were designed as minimalist implementations of functionality we require from connected devices. We've avoided implementing any unnecessary features.

[1]DBGU is actually used as third UART

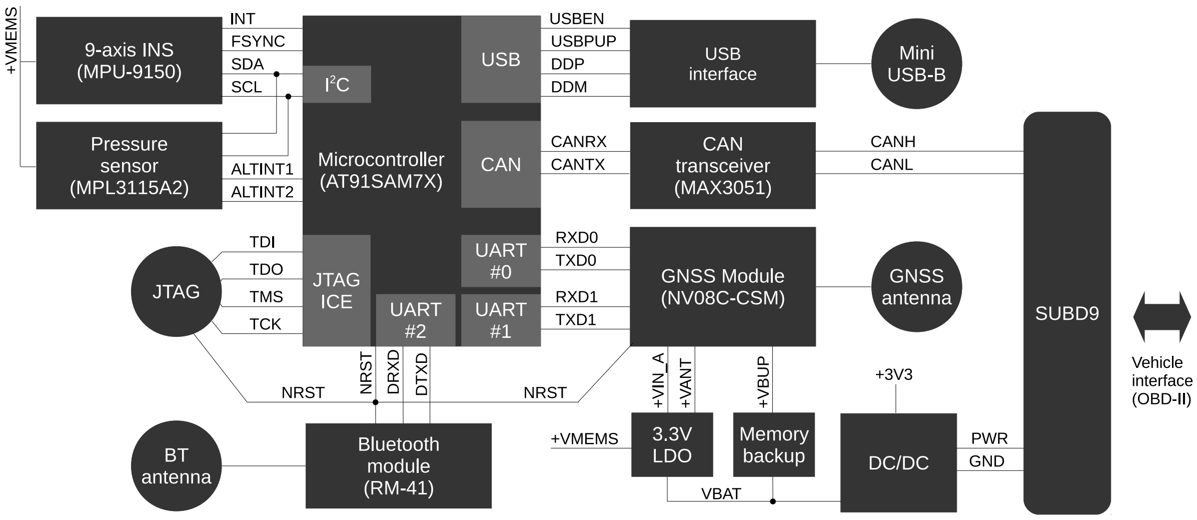


Figure 3.1: PCB block diagram

3.1 Printed circuit board design

The printed circuit board (also reffered to as PCB, or simply board) has 103 × 53 × 9 millimeters. It is classic 2-layer PCB featuring 1.5mm FR4 dielectric with 18μm copper layer on both sides. Green solder mask and white printing is also both sided. Contact pads were chemically gilded to avoid corrosion. See Figure 3.2 for finished board.

The device needs DC voltage in range from 5V to 15V. Small overvoltage will cause increased current draw due to transient voltage suppression diode, high overvoltage and reversed polarity will short the circuit and blow 2A fuse.

It features three buttons and five LED diodes. The buttons are used for system-wide reset, to erase program memory and to put Bluetooth module to factory reset. Three LED diodes are used by Bluetooth module to indicate mode, whether it is connected and to indicate ongoing data transfer. One LED is used by the microcontroller to indicate health (so-called heartbeat LED) and the last LED indicates state of NRST signal routed thorough the board to all resetable circuits.

Three connectors are available. SUBD-9 (also called CANNON-9) is used to connect to the vehicle. The power is routed through there together with CAN-bus (if available). SMA connector for external GNSS antenna and JTAG connector are available as well.

3.1.1 Dealing with interference

There are three microwave signals routed on the board - 2.45Ghz signal from Bluetooth antenna, 1.55Ghz signal from GNSS antenna and USB data lines clocked on 480Mhz. Also, accelerometers, gyroscopes, magnetometers and barometer make use of high resolution analog-to-digital converters. Hence, reducing all kinds of interference is critical if the board is to deliver without spurious failures.

We have deployed number of different techniques to minimize it. Some are concerned with differential-mode and common-mode noises propagated on copper traces of the board. Other techniques are used to minimize EMI interference.

We have created "copper pour islands" under each of the subsystems mentioned above (microcontroller, sensors, communication interfaces) and made sure they are connected to

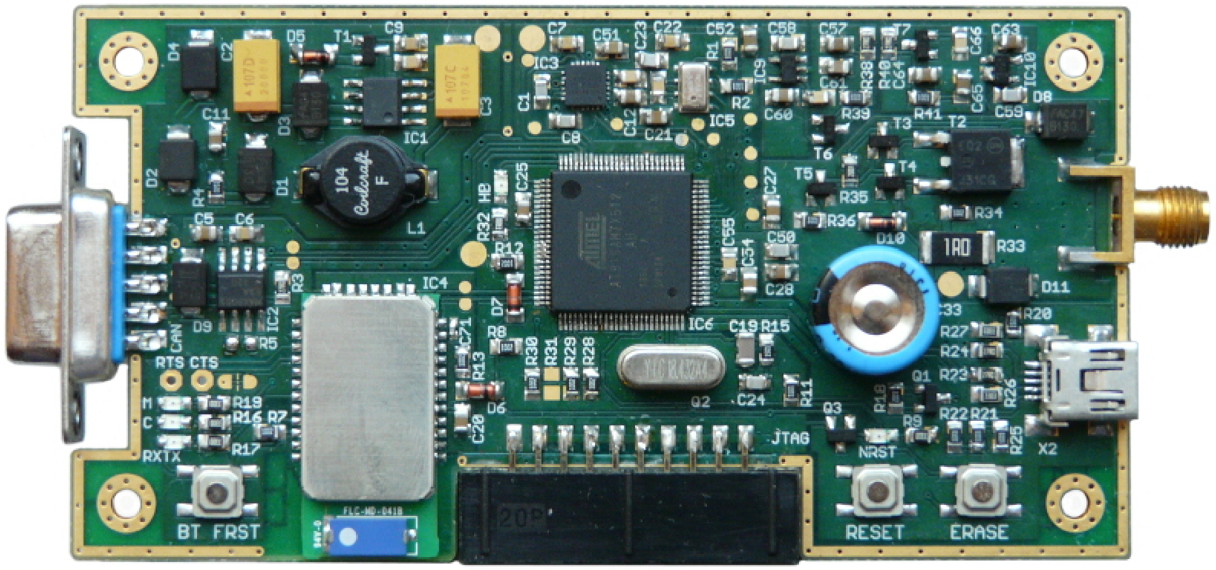


Figure 3.2: Front side of the printed circuit board.

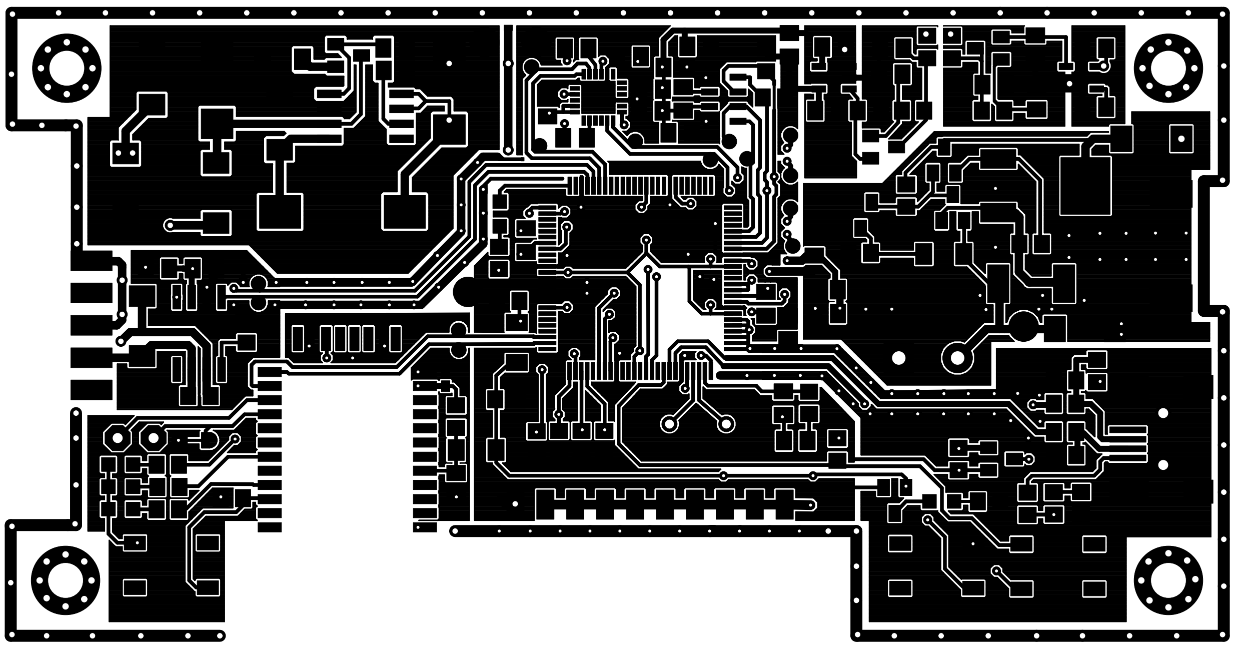
power lines (+3V3, GND) at one point only. We have put at least three MLCC capacitors (10nF/C0G, 100nF/X7R, 10μF/X7R) to the point where the power lines are connected to the copper pour. Noise have no choice but to pass through this place where we can effectively eliminate it with filtering capacitors. This way noise from one part of the board cannot escape to another.

Main power supply is based on step-down converter. These circuits tend to generate noise on their switching frequency which can couple to both +3V3 and ground. Consequently, part of the noise is common for both lines. In order to avoid it we didn't allow GND power line to pass through this noisy area and connected it to power supply in one place only. Best place for that is right under the output capacitor using a via to other side of the board, where the GND line is routed. Note that the via has some inductance as well. It improves performance because it behaves as a small choke.

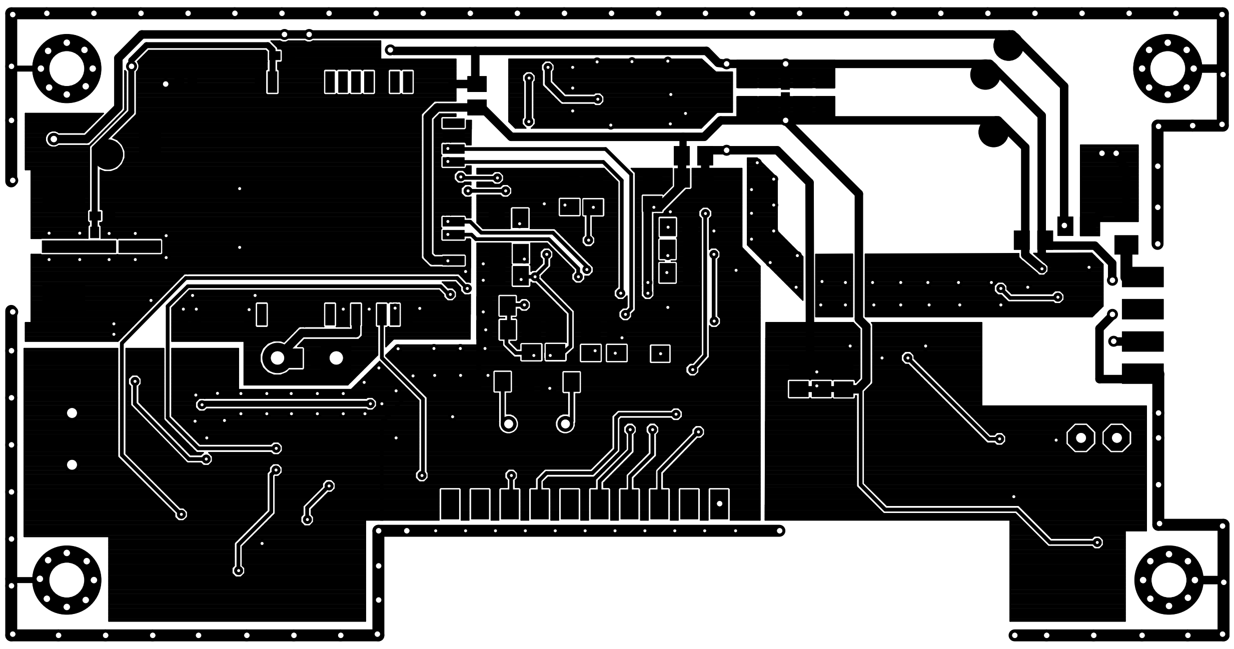
In order to further improve our guards against noise we placed vias under both pads of the output capacitor. On the other side we placed five ceramic capacitors ranging from 22pF to 10μF to filter differential-mode noise (noise coupled to one of the two power lines, not both).

Result is that we get very clean output even switching power supply is used. Nonetheless, there are high-resolution ADCs on board which require highly stable voltage to avoid coupling of noise to their digital output. So, we designed two low-noise power supplies to provide analogous parts of the board with separate power. First is for inertial measurement unit and barometer, second for analogous part of GNSS receiver and its active antenna.

We placed guard ring around the board. It is a copper trace around its edge connected to ground. Important is that ground is connected at a single place that is as close as possible to power connector. In principle, guard ring works as trap for interference radiating out of the board. The trace is both sided with frequent vias to minimize impedance between top and bottom sides.

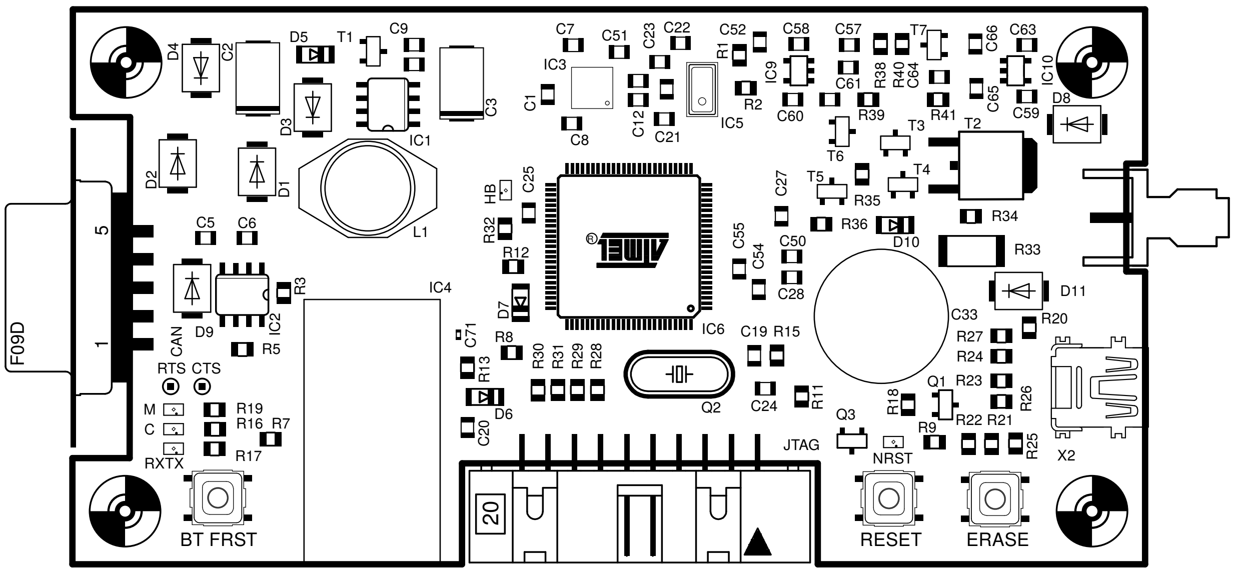


(a) top side

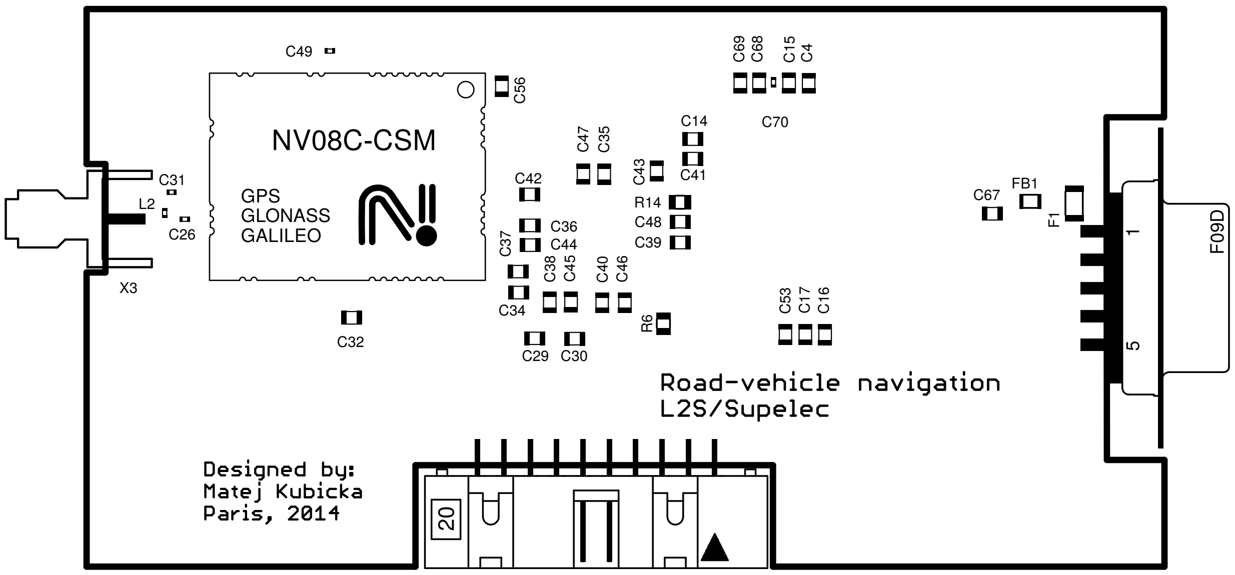


(b) bottom side

Figure 3.3: PCB copper traces (not in scale)



(a) top side



(b) bottom side

Figure 3.4: PCB parts placement (not in scale)

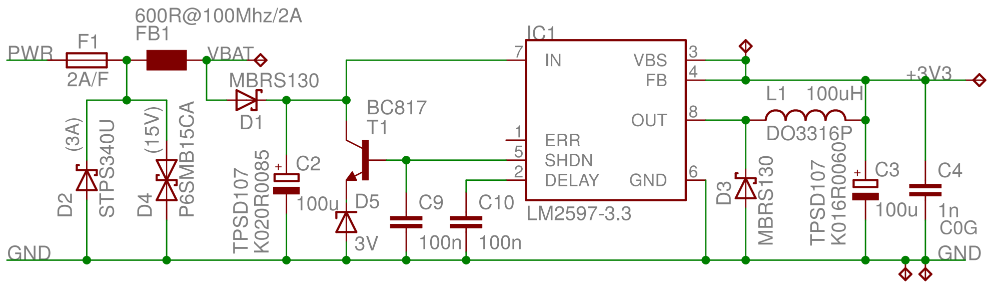


Figure 3.5: power supply for digital circuitry

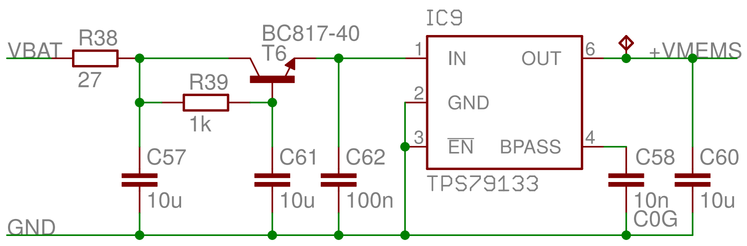


Figure 3.6: power supply with low-noise properties

3.1.2 Power supplies

As was mentioned above, the board uses three power supplies. Main power supply provides power to digital circuitry and the other two are supplementary for sensitive analogous circuitry.

Vehicle battery is connected through the SUBD-9 connector. In schematic we use names PWR and GND for the two nets. Firstly, there is SMT fuse (rated 2A; very fast) on PWR line for protection. Transient voltage suppression diode, D4, and reverse-polarized diode D2 follow. Diode D4 is used to suppress transients and to limit maximum applied voltage to ±16V. When the voltage is higher than that the D4 opens and short-circuits the power lines. As D4 is bidirectional, we also use reverse-polarized diode D2 to short-circuit the power lines in case voltage is reversed. In effect, if applied voltage is not in range from -0.7V to about 16V, then the protection circuitry shorts PWR and GND together, causing the fuse to blow.

Main power supply is a step-down converter with properties listed in Table 3.1. It is classical design with switcher, inductor, input/output capacitors and flyback diode. We use LM2597-3V3 switcher from Texas Instruments, 100μF tantalum input/output capacitors, 100μH power inductor and 1A Schottky diode. Both input and output capacitors were selected so they have very low equivalent series resistance. Also, the power inductor is rated for double current than required to avoid inductance loss due to saturation and to minimize heat losses.

The other two power supplies are identical in design. First powers the MEMS devices, second GNSS antenna and baseband. It is important that they both provide clean and stable voltage.

In heart of these power supplies lays linear voltage regulator. These devices tend to generate noise as small variations in their voltage reference are multiplied by feedback loop and propagated to the output. Even linear regulators are usually less noisy than switched circuits, this is still a problem to address when aiming for low noise design. We have used linear low drop-out regulator TPS79133 from Texas Instruments. This device has extra pin for bypass capacitor that, if used, stabilizes its internal voltage reference.

Input voltage	5-15V
Output voltage	3.3V
Maximum load	500mA
Output voltage ripple	15mV
Switching frequency	150kHz

Table 3.1: main power supply properties

On the input side of the voltage regulator we placed RC circuit and so-called "capacitor multiplier" circuit. The RC circuit works as low pass filter with 3.3kHz cut-off frequency. Ferrite bead FB1 behaves as inductor for frequencies below 100Mhz, so it might happen that noise in that range is amplified by LC circuit formed by FB1 and input capacitors on these secondary power supplies. Both ferrite beads and ceramic capacitors have high quality factor which allows them to create gain peaks on their resonant frequency. The series resistor in the RC circuit effectively damps the peaking by diminishing quality factor of the LC circuit.

The capacitor multiplier helps to remove ripple voltage. In principle, this circuit multiplies capacity of C61 with current gain of transistor T6. Note that the capacity isn't actually multiplied, only impedance of this circuit resembles impedance of much larger capacitor than actually used.

3.1.3 Microcontroller

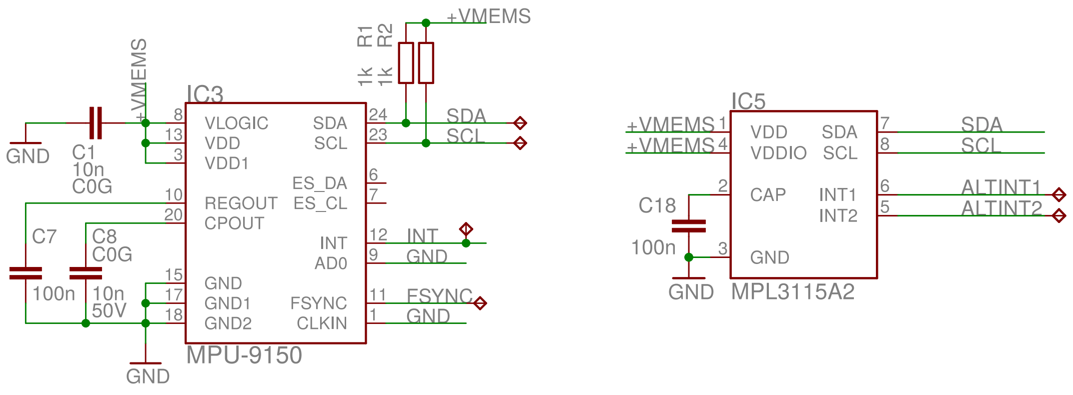
Microcontroller is the heart of this embedded system. Its task is to collect measurements, filter them and exchange processing and servicing information with devices connected either via Bluetooth or USB.

We have chosen AT91SAM7X512 microcontroller from Atmel. This device has 512kB of Flash EEPROM memory, 256kB of SRAM and ARM7TDMI processor from ARM. Hence, there is plenty of memory and computational capacity available. The device also features CAN, USART, DBGU, USB and I2C peripherals that we need to connect to all on-board sensors and interfaces. Only missing feature is so-called FPU unit that would enable hardware support for floating-point numbers. Unfortunately, ARM7 cores were never equipped with this.

It became apparent in late stages of our analysis that the integration filter will have to be implemented in this microcontroller (see Section 2.7). Implementing Kalman filter with fixed point arithmetic is problematic due to numerical instability of the solution - covariance matrices deteriorate in time because of accumulated rounding errors. These matrices need to stay positive definite, otherwise the filter will fail. Often, they have to be stored in squared-root form and squared on each iteration in order to make sure that even severely deteriorated covariance matrix is always positive definite. Of course, it is possible to implement Kalman filter using floating point arithmetic even on fixed point processor, but it will be slow as there is no hardware support for the floating point operations. New versions of this system would be better off using microcontroller with floating point support, or even better, with DSP (digital signal processor) replacing the microcontroller.

Used microcontroller has PLL to generate main clock frequency. We clock the device on 48.092MHz using 18.432Mhz crystal oscillator[2]. The clocking frequency might be changed

[2]USB peripheral requires this clock frequency to operate



(a) inertial measurement unit **(b)** barometric altitude meter

Figure 3.7: MEMS sensors (filtering capacitors not shown)

in software, but one has to be careful as there is filter for PLL outside the chip that might not comply with changed clock frequency, see [18]. Maximum clock frequency is 55Mhz, although one can overclock it (risking device damage and/or shorter lifetime).

It also supports standard JTAG boundary scan and JTAG Embedded ICE (in-circuit emulator). We don't make use of the boundary scan so this feature is permanently disabled, but we make heavy use of Embedded ICE for software development and debugging.

NRST resetting signal from JTAG is routed thorough the board to reset the microcontroller, GNSS receiver and Bluetooth transceiver. Note that when the reset button is pushed it takes NRST signal to active state.

Future versions should route NRST signal to enabling inputs of the voltage regulators used in secondary power supplies (Figure 3.6). This will reset the remaining circuitry when NRST is asserted.

3.1.4 MEMS sensors

Both MEMS devices are connected on separate power supply (see Section 3.1.2) and communicate with main microcontroller on shared I2C bus. They are both I2C slaves while the microcontroller acts as I2C master.

Inertial measurement unit MPU-9150 is multiple-chip-module (MCM) with two silicon dies in single LGA package. First die accommodates accelerometer and gyroscope, second is the 3-axis magnetometer. These are, in fact, two separate devices - first die is normally sold as MPU-6150 from InvenSense and second as AK8975 from Asahi Kasei Microdevices. They are internally connected with separate I2C bus. First die (MPU-6150) acts as master, second as slave. Other slave devices can be connected via ES_DA and ES_CL pins. Note that it is possible to bypass the secondary bus programatically. Doing it would enable direct access to the magnetometer.

This device also prides itself with feature named "DMP processor". It seems to be some sort of specialized digital signal processor. Unfortunately, there is very little information available about it. Official documentation [23] don't mention this feature apart of admitting its existence. There is application note available with instructions on how to upload given binary file to the device. When uploaded, it enables number of motion detection features which are not available otherwise. Nonetheless, there is no documentation, no source code, and no development tools. Consequently, it is useless for our purposes.

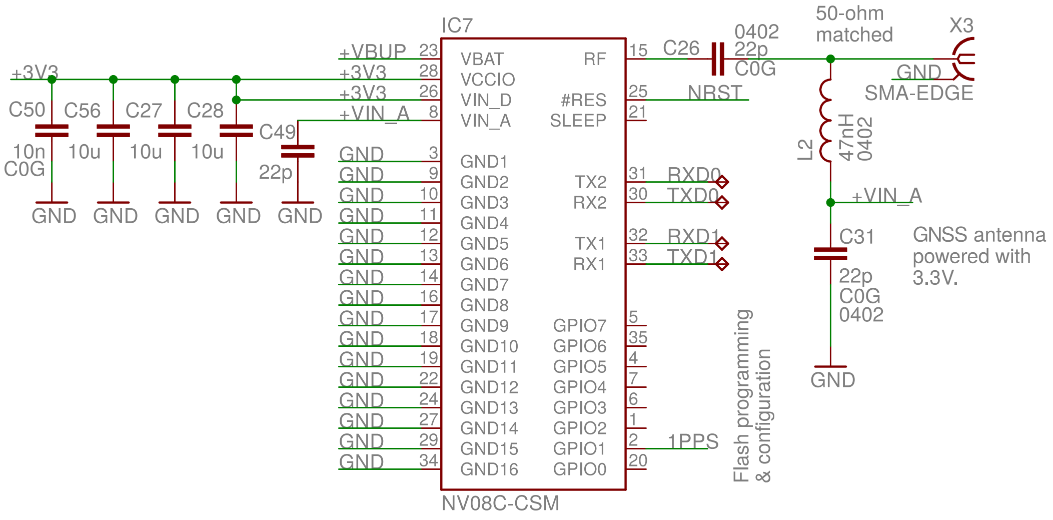


Figure 3.8: GNSS receiver

Communication between the MPU-9150 and main microcontroller is interrupt driven. MPU9150 asserts its INT line when new data are available. When main microcontroller detects the change it will start I2C-read command to retrieve the data. This will automatically clear the interrupt source and de-assert the interrupt line.

Figure 3.7b shows schematic connection of barometric pressure sensor MPL3115A2 [24]. This device converts pressure to height above mean sea level as described in Section 2.4. It is enclosed in 8-pin LGA package from noncorrosive metal with a hole for pressure sensing. Signals SDA and SCL belong to I2C bus. The ALTINT1 and ALTINT2 are interruption signals connected to main microcontroller. Capacitor C18 is required for proper operation.

Communication with the barometer is interrupt driven as well - when new data are available the device asserts ALTINT1 line and microcontroller reads the data in response. ALTINT2 is not used at the moment.

3.1.5 GNSS receiver

NV08C-CSM is standalone GNSS receiver module, for its connection with other parts of the board see Figure 3.8. It is small printed circuit board covered with metallic shield that leaves only SMT soldering pads outside. No blocking capacitors or extra filtering is necessary according to documentation so we added only 10μF capacitors on power inputs to lower source impedance. Capacitor C50 is used as placeholder in case we need to add filtering capacitor later.

GNSS antenna connects on SMA connector X3. Power supply is provided for the antenna, maximum allowed current consumption is about 70mA. DC bias on the RF line is separated from the GNSS receiver with capacitor C26.

The NRST signal is system-wide reset as discussed in Section 3.1.3. Signal 1PPS generates short pulse when one-second boundary is passed. It allows alignment of local wall-clock time with UTC time reference. Signals RXD0/TXD0 and RXD1/TXD1 are two UART channels used to communicate with microcontroller. Channel 0 is preset to communicate using standard NMEA-0183 protocol, channel 1 uses manufacturer's proprietary BINR protocol. Optionally, channel 1 can be switched for RTCM-SC104 protocol.

Common pain with GNSS receivers is long time to first fix. It might take several minutes

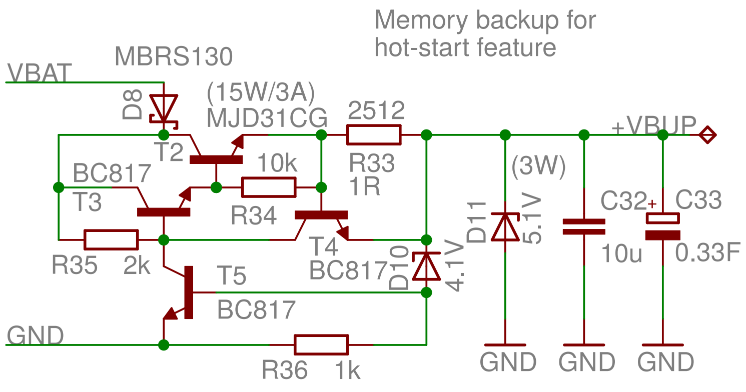


Figure 3.9: memory backup for GNSS receiver

for the receiver start up. For this reason we use memory backup circuit that preserves its memory across resets and short power outages, see Figure 3.9. In heart of this circuit lays 330mF backup capacitor (C33, double layer "gold" capacitor). This type of capacitor have outstanding capacity with otherwise terrible properties. First thing to know about it is that it does not behave like capacitor - more like network of many small capacitors in parallel, each having random resistance in series added to it. The "small" capacitors inside with small series resistance will charge almost immediately causing the device to seem charged (it will quickly report full voltage across its contacts). Nonetheless, it is not the case as the rest of the "small" capacitors with large series resistors will be charging slowly. In our experiments the capacitor have charged in first 5 seconds of operation to about 40%. Rest was charging for several hours using minimal currents.

The circuitry around C33 is to provide correct voltage and to limit the current. Maximum voltage for this type of capacitor is 5.5V, while all power supplies provide only 3.3V. Hence, we used battery voltage (≈12V) and designed basic voltage regulator with current limiting capability for its charging and protection.

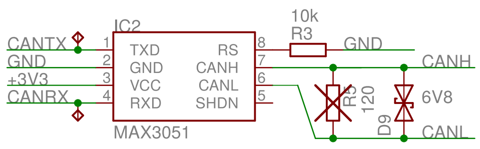
3.1.6 Communication interfaces

The system can communicate with other devices using Bluetooth, USB and CAN bus. Bluetooth is used to connect with host system, USB is used as backup channel and CAN bus is used to communicate with the vehicle.

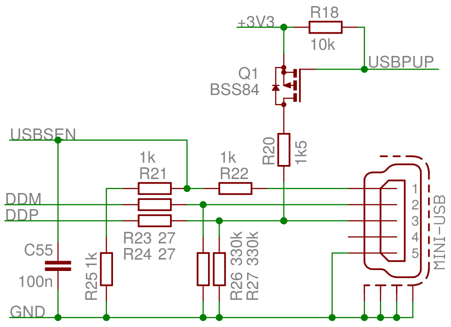
Bluetooth module RN41 from Microchip is used to implement Bluetooth interface. It is standalone system that doesn't require any extra circuitry. Only power supply and UART channel needs to be connected. The device works as transparent cable replacement using standard serial port profile (SPP). Data received on UART are enclosed in Bluetooth packet and transmitted wirelessly. Similarly, data received over the wireless channel are forwarded to the UART. Hence the main microcontroller communicates with the Android device using its UART channel having no idea that the data are actually transmitted wirelessly.

CAN is supported by the microcontroller, but it implements only CAN protocol layer, not the hardware layer. Hardware layer requires powerful bus drivers that does not fit on a VLSI die with microcontroller. Hence, external CAN transceiver is necessary. We chose MAX3051 from Maxim Integrated, see Figure 3.10a.

Signals CANH and CANL connect the CAN bus. Note the crossed resistor R5, it is used as placeholder for termination resistor. The transient voltage suppression diode D9 is



(a) CAN transceiver



(b) USB interface

Figure 3.10: CAN and USB interfaces

used for protection against high transients that can damage the transceiver. Resistor R3 on RS pin is used for slope control. The microcontroller and the transceiver use CANTX and CANRX signals to communicate. CANTX is driven by the transceiver according to bus state. CANRX is driven by the microcontroller to instruct the transceiver whether it should force dominant state on the bus or not.

USB 2.0 bus is supported by the microcontroller. Schematic diagram on Figure 3.10b follows recommendations of application note “AT91SAM7X and AT91SAM7XC Microcontroller Series Schematic Check List” from Atmel, see [18]. Signals DDM and DDP are USB data lines, they connect to microcontroller. Note that DDM is inverted version of DDP. Signal USBSEN is used to sense voltage on the USB bus, USBPUP is used to select between high speed and full speed.

MiniUSB-B slot is used instead of classical USB-B as this one is smaller. DDM and DDP traces are routed as short as possible and shielded from the rest of the board. We followed recommendations from application note “USB 2.0 Board Design and Layout Guidelines” from Texas Instruments, see [19].

3.2 Low-level drivers and support libraries

This section is concerned with libraries and drivers that help to accommodate application in the microcontroller. Hence, everything mentioned here relates only to the microcontroller and its peripherals. Device drivers and the application are described later. We haven’t used any support library[3] and wrote the code from scratch using only newlib (incarnation of standard C library).

Even we currently don’t use any support library, our intention is to use RTOS in future. What we need is not full-fledged RTOS but merely simple real-time threading library with support for EDF scheduling.

We have analyzed applicability of following real-time operating systems: FreeRTOS, FreeOSEK, TOPPERS-OSEK, nxtOSEK, eCos, SHaRK and ERIKA Enterprise. FreeRTOS is well-known to work, well-documented, but while it calls itself free, the documentation is available for a fee. FreeOSEK is one of many OSEK implementations, but it seems unfinished and forgotten (last update 2010). TOPPERS-OSEK is a good candidate, but the documentation is in Japanese. nxtOSEK (based on TOPPERS-OSEK) is used in Lego NXT bricks. We use similar microcontroller so it is possible to use it with

[3]Suitable library would be AT91lib provided by Atmel. We were not satisfied with its quality.

few modifications. Nonetheless, nxtOSEK is not just RTOS - it combines a number of things together. Furthermore, eCos and ERIKA Enterprise operating systems are too big and complex, while we were aiming for something light and clean. Finally SHaRK RTOS seems outdated, which is a pity as it is only RTOS from the list capable of EDF scheduling.

We chose FreeRTOS because it is best match for our needs. It is not fully to our satisfaction, though, as FreeRTOS uses simple round-robin scheduling with support for preemption (instead of EDF scheduler we wish to deploy). The program on microcontroller is designed in such a way that FreeRTOS can be easily integrated to it. For the moment we use a number of alarms that run specified routine when expired. This is by no means optimal, but we decided to stick with this simple solution for the moment.

3.2.1 Bottom halves

Bottom halves are used to defer time-consuming processing that should be done within interrupt for later. All interrupt sources are blocked when a single interrupt is serviced. Hence, it is critical that interrupt servicing routines finish as quickly as possible, otherwise system responsiveness will suffer. One solution is to use nested interrupts, but this often cause more problems than benefits. Alternative solution is to use bottom halves.

The concept (and name) of bottom halves is taken from Linux kernel. Linux separates interrupt handler to two parts - top half and bottom half. Top half is interrupt service routine in classical sense. When it is finished the processor re-enables interrupts and, normally, it would return to the point where program was interrupted. The idea of bottom halves is to do interrupt's lengthy processing after interrupts are re-enabled but before the execution returns to the program. What we get by implementing bottom halves is means to run lengthy tasks within interrupt while other interrupts (of all priorities) are not blocked.

Our implementation of bottom halves is simple. Internally it uses bit-field to collect pending requests and an array of function pointers to handling functions. Its API features following routines:

- `void bh_init()` - Initializes internal data structures, must be called during system startup.
- `int bh_register()` - Used to register new bottom half. It takes function pointer in form of `int (*h)(unsigned long)` as its only argument. It shall point to bottom half handler. Handle to allocated bottom half is returned.
- `void bh_request()` - Registers request to run bottom half. It takes two arguments. First is handle to the bottom half (must have been previously returned by `bh_register`). Second argument is going to be passed to bottom half handler when invoked.
- `void bh_process()` - Executes all pending requests. It may never be called from interrupt context as it blocks until all requests are serviced.

Note that bottom half handler must always return `BH_OK` in case of success, otherwise the request will not be cleared and the handler will be called again on next invocation of `bh_process()`.

Current implementation does not process bottom halves right after interrupt handler. There is no need to run it immediately as we don't use RTOS yet. Function `bh_process()` is called from main program loop instead. When we integrate RTOS to our system we will have to implement it as part of interrupt servicing. It can be done by calling `bh_process` right after global interrupt mask is re-enabled and before execution is returned to the point of interruption. Note that it is not as straightforward as it might sound because one has to be careful not to loose context of the program before interruption.

3.2.2 Alarms

Purpose of alarms is to call given function when time expires. We use periodic interval timer (PIT) on the microcontroller to count time. It generates interrupts periodically on each millisecond. The interrupt handler simply increments global variable `time` on each invocation. This realizes simple timer with resolution of one millisecond.

We use alarms to call functions that implement tasks. There are 8 alarms available, each can be programmed separately. Every time new alarm is configured the code will sum current time with given delay to get expiration date. Then, the expiration date is registered together with the pointer to function that shall be called when timer expires. Note that the alarm can be restarted from within function it called upon expiration. In this way one can achieve periodical behavior. Alarms API consists of following functions:

- `void pit_init()` - Initializes the timer and related data structures.
- `void pit_setup_alarm()` - Sets up new alarm. It takes three arguments: first is alarm identifier (valid values are from 0 to 7), second is waiting interval in milliseconds and third is the function pointer that is called upon timer expiration. Its prototype is simply `void (*hook)(void)`. Note that it is possible to setup only single-shot alarms.
- `void pit_restart_alarm()` - Restarts disabled alarm. Its only argument is alarm identifier as interval and function pointer values are copied from previous setting.
- `void pit_process_alarms()` - This routine checks whether any of the alarms expired and, if so, disables it and calls given handling function. Currently, we call this routine from main program loop in order to service the alarms as frequently as possible. This shall be implemented as bottom half of the PIT interrupt when we integrate RTOS with our system.
- `unsigned long get_system_time()` - Returns value of `time` variable. It takes no arguments.

3.2.3 Timestamps

The millisecond timer described above provides useful reference for local timekeeping but we needed higher precision for the integration algorithm. Each measurement has to be timestamped in order to know age of information it carries. Furthermore, the integration algorithm requires all its inputs to refer to same instant in time. This is not feasible with different sensors running on different clocks. Solution is to have a model that can estimate measurements at any time[4] by fitting past measurements on a curve.

Modern processors are usually equipped with so-called timestamp register. It is 64-bit free-running counter that increments on each rising edge of system clock. Unfortunately, ARM7 processor doesn't have this feature. Our solution uses three serialized 16-bit timers in pulse generation mode. Input of first timer is connected to system clock divided by five (9.618 Mhz), output is connected to input of second timer. Similarly, output of second timer connects to input of third timer. Having each timer set to generate rising edge on overflow we get 48-bit free-running counter similar to timestamp register available in other processors. First timer overflows every 6.8 milliseconds, second timer every 446.55 seconds and third timer in about a year.

Reading correct value from the three timers at once is tricky. We need to read three registers one after another meaning that this operation is not atomic. Consequently, when we read state of any of the three timers the other two might overflow in that very moment. If that happens then reported time might be shifted either by 6.8 milliseconds or 446.55 seconds. For solution see function `read_timestamp_raw()` in Listing 3.1 below. We start by reading the timing registers from highest significant bytes to lowest. Then we read again values of upper and middle timers to verify that they didn't overflow sometime during the process. This is repeated until no overflow is detected. Note that probability that one of the 3 timers overflows while we read their value is below 0.1%.

```
static void read_timestamp_raw(int *l, int *m, int *h)
{
  int m1,h1;

  /* the timestamp will be spoiled if irq strikes here */
  do {
    *h = TC_CV(TC2);
    *m = TC_CV(TC1);
    *l = TC_CV(TC0);
    m1 = TC_CV(TC1);
    h1 = TC_CV(TC2);
  } while(m1!=*m||h1!=*h);
}
```

Listing 3.1: function `read_timestamp_raw()`

There are two functions in the API that return most current timestamp. First is `get_timestamp()` and second is `get_timestamp_isr()`. Former disables the interrupts before calling `read_timestamp_raw`, latter doesn't do that as it is meant to be invoked from interrupt context (where the interrupts are disabled already).

Note that it is not allowed for the function `get_timestamp()` to be called from interrupt context.

[4]small number approximation applies

3.2.4 I2C peripheral driver

We use I2C bus to communicate with the inertial measurement unit and barometric altimeter. Protocol and signaling is handled by I2C peripheral[5]. We merely needed to write hardware abstraction layer that would provide reasonable application interface. Note that the microcontroller acts as I2C master only, I2C slave is not supported.

Main requirement from this driver is that it runs asynchronously. It means that both send and read requests cannot block the processor until the transfer is finished. The I2C bus is much slower than processor so blocking calls would cause tremendous performance regression. Unfortunately, the peripheral is not asynchronous in nature and it does not support DMA. Nonetheless, it is possible to implement the driver as state machine that executes asynchronous transfers through interrupt-based notifications.

Our implementation uses prioritized queue of I2C transmission requests. The state machine takes request with highest priority, executes it and when done continues with next highest priority request until there is none left. Transmission handling is done asynchronously, but there is function `i2c_do_requests` that has to be called periodically for house keeping and starting new requests. Currently we call this routine from main program loop but with RTOS it should be implemented as bottom half.

There are two data structures involved. First is `i2c_device` that is used to specify properties of connected devices (such as device address and internal addressing mode). Second is `i2c_request`. This structure gathers all information about the request necessary to carry it out. Normally the requests are prepared during system startup and used repeatedly to perform read or write operations later. Declaration of both data structures is in Listings 3.2 and 3.3.

```c
/* used to describe a device */
struct i2c_device {
  unsigned int address;
  unsigned int iadrsz;
};
```

Listing 3.2: `i2c_device` data structure

Structure `i2c_device` contains only device address and specification of internal address size. For example, EEPROM memories often use 3-byte internal addressing while sensors use only 1-byte addressing as they dont have that much memory to address.

```c
/* used to specify a request */
struct i2c_request {
  struct i2c_device *dev;
  unsigned char *data;
  /* flags can be retrieved with i2c_get_finished(), i2c_get_dir(),
  i2c_get_bh(), and i2c_get_prio() macros */
  unsigned int flags;
  unsigned int iadr;
  unsigned int size;
};
```

Listing 3.3: `i2c_request` data structure

[5]Strictly speaking the microcontroller does not support I2C - it supports another bus called TWI (two-wire interface) which is I2C compliant.

Structure `i2c_request` contain many fields, see Listing 3.3. Pointer `*dev` specifies the device for which the request is ment to. Pointer `*data` points to location in memory where received data bytes should be copied to in case of read request or where the data shall be copied from in case of write request. Field `flags` specifies number of different things such as priority, transfer direction and others. Notably, it encodes identifier of bottom half that shall be invoked when the request is finished. Furthermore, field `iadr` is the internal address in the device and field `size` contain number of bytes to transfer.

Note that location in memory where `*data` points to cannot be accessed until the request is finished. For this reason we notify the application by invoking bottom half specified in `flags` field upon finishing. This is convenient as I2C driver is not used directly by the application but by device drivers that handle devices connected on I2C bus. They need to be notified as soon as possible in order to increase system responsiveness. Bottom half is fastest way to do it.

Finally, there is blocking interface existing in parallel with non-blocking equivalent described so far. Non-blocking interface is difficult to use for single-shot transfers, but it is well suited for repetitive transmissions during normal operation. Blocking interface is easy to use for device configuration purposes, but it would slow down the microcontroller during normal operation. List of all interface functions follows:

- `void i2c_init()` - Initializes I2C peripheral and internal data structures.
- `int i2c_write_blocking()` - Performs I2C write command and does not return until the transfer is finished. It takes four arguments: first is pointer to `i2c_device` structure, second is address in the device where the data shall be stored, third is pointer to the data and last argument specifies how many bytes shall be transferred. Returns non-zero value upon success.
- `int i2c_write_1blocking()` - Wrapper of `i2c_write_blocking()` that transfers only a single byte. Instead of pointer to data it takes directly single value that will be stored on given address in the device.
- `int i2c_read_blocking()` - Performs I2C read command and does not return until the transfer is finished. It takes same arguments as `i2c_write_blocking()`, but in this case `data` pointer points to place in memory where to store the data, not where to read them.
- `unsigned char i2c_read_1blocking()` - Wrapper function that simplifies use of `i2c_read_blocking()` when only single data byte needs to be read from given address in the device. It takes two arguments: first is pointer to `i2c_device` structure and second is internal address where to read from. Retrieved data byte is returned.
- `struct i2c_device *i2c_create_device()` - Returns initialized structure `i2c_device`. It takes two arguments: first is device address and second is size of its internal address in bytes. Put zero if the device does not support internal addressing.
- `struct i2c_request *i2c_create_request()` - Creates asynchronous I2C request and returns pointer to it. It takes 6 arguments in following order: pointer to valid `i2c_device`, priority, internal address, read/write flag, transfer size and bottom half handle. For the prototype see header file `i2c.h`. Note that this function

does not actually request data transfer, it just creates the data structure necessary for that.

- `struct i2c_request *i2c_create_read_req()` - This is actually a macro that expands to `i2c_create_request()` with read/write flag set to read. It takes same arguments as `i2c_create_request()` without the read/write flag.
- `struct i2c_request *i2c_create_write_req()` - Also a macro that expands to `i2c_create_request`, only the read/write flag is fixed on write.
- `void i2c_nonblocking()` - Requests non-blocking transfer. It takes two arguments: pointer to `i2c_request` structure and pointer to data where to copy from (or copy to).

Note that some I2C slave devices does not support internal addressing. This can be indicated by setting internal address size (`iadrsz` field in `i2c_device` structure) to zero. Then, field `iadr` in structure `i2c_request` is ignored.

3.2.5 USART and DBGU peripheral drivers

We needed three UART channels - two for GNSS receiver and one for Bluetooth. The microcontroller is equipped with two USART channels that can be configured as standard UART. Furthermore, there is DBGU peripheral meant for debugging and testing purposes that is equipped with standard UART as well. We used DBGU as third UART channel for communication with the host system over Bluetooth and the two USARTs to exchange data with GNSS receiver.

Both USARTs and the DBGU are equipped with their own DMA channel (called peripheral data channel - PDC). The DMA is critical as it allows to copy data between UART and SRAM automatically without assistance of the processor. This is optimal from processor load point of view. One can program the DMA to store received data to SRAM automatically. Similarly, it is possible to program the DMA channel to transmit specified part of SRAM memory over UART.

We don't access USART and DBGU peripherals directly, instead we use the DMA. Each UART channel has associated 4kB reception and transmission circular buffers. Received data are automatically stored to the reception buffer. The application has to read content of the reception buffer as often as possible to avoid data overriding. Similarly, if the application wish to transmit data it only needs to copy them to the top of the transmission buffer and to notify DMA about number of copied bytes.

The application interface for the USART peripherals consists of following functions:

- `void usart_init()` - Initializes both USART peripherals and internal data structures. This has to be called prior to `usart_setup()`.
- `struct usart *usart_setup()` - Configures USART peripheral. It takes three arguments. First is base address of the peripheral to use (USART0 is at 0xFFFC0000 and USART1 at 0xFFFC4000). Second argument is baud rate, it is used to compute baud rate generator settings. Third argument is for flags that are forwarded to USART mode register (see datasheet [20]). Normally it can be left zero.

  It returns pointer to structure `usart` that is used as a handle to specify which peripheral to use when calling `usart_send()` and `usart_get()` functions.

- `void usart_send()` - Used to send data over UART. It takes three arguments: first is pointer to structure `usart` that specifies which USART peripheral shall be used. Second argument is pointer to buffer with data and last argument is a number of bytes to transfer. This function is not blocking until the transmission is finished, it will only copy the data to the transmission buffer and program the DMA to do the job.
- `int usart_get()` - Copies data from reception buffer to given location in memory and returns number of bytes copied. It takes two arguments: first is pointer to structure `usart` and second is pointer to address in memory where the data shall be copied to.

As you can see, the interface is simple as all DMA programming and parallelism-related issues are handled internally. Note that interrupt sources USART0 and USART1 have to be handled by routines `usart0_irq()` and `usart1_irq` for the driver to work correctly.

The DBGU driver implements both low-level DMA handling and high-level communication protocol used to communicate with the host device. The protocol is described in Section 3.4.1. Here it suffices to say that it uses a number of different data objects. Each data object have assigned unique identifier and pointer to function that handles its reception. DBGU driver application interface consists of following functions:

- `void dbgu_init()` - Initializes DBGU peripheral. Its only argument is requested baud rate.
- `void dbgu_setup_do()` - Configures new data object (DO). It takes two arguments: data object identifier (valid values are from 0 to 15) and pointer to function with prototype `void (*hook)(unsigned char *data, int size)`. This function is called upon reception of data object.
- `void dbgu_send_do()` - Sends data object. It takes three arguments: first is data object identifier (values from 0 to 15), second is pointer to memory location where the message is stored and last argument is message size in bytes.
- `void dbgu_process_rx()` - Processes reception buffer and invokes data object handlers accordingly. This function has to be called periodically to process the reception buffer. It takes no arguments.

3.2.6 CAN peripheral driver

CAN peripheral is asynchronous in nature. There are 8 mailboxes in hardware, each can contain a single message. The mailboxes can be configured either to hold received messages according to acceptance criteria or to hold messages that are due for transmission.

Each reception mailbox have acceptance criterion that, if met, allow received message to be copied into the mailbox. The message is dropped if no mailbox have compliant acceptance criteria.

The criterion consists of matching identifier and its acceptance mask. When new message is received then its identifier bits are multiplied with acceptance mask. Similarly, matching identifier is multiplied with acceptance mask as well. If the two masked identifiers are identical, then the message is accepted. This process is repeated for all receiving mailboxes.

CAN bus bit timing scheme is complex so the programmer has to compute the values manually and provide them to `can_init()`. CAN driver application interface consists of following functions:

- `void can_init()` - Initializes CAN peripheral. It takes 5 arguments: first is baud rate prescaler (divides system clock) and then follows phase1, phase2, synchronization jump width (SJW) and propagation delay. These parameters are specified in time quanta. One time quantum has period of baud rate prescaler divided by system clock (see [21] for details on CAN bit timing).
- `int can_setup_tx_mailbox()` - Configures mailbox for transmission. It takes three arguments: message identifier, flag whether the identifier has standard or extended size and flag whether the message is remote request or not. Note that message data are not given at this point. Returns handle to the mailbox.
- `int can_setup_rx_mailbox()` - Configures mailbox for reception. It takes four arguments. First two are acceptance criteria (message identifier and acceptance mask). Third argument specifies whether received message is remote request or not. Last argument is bottom half identifier that shall be invoked when a message gets accepted.
- `void can_send()` - Used to send a CAN message. It takes three arguments: first is mailbox handle, second pointer to memory where the message is stored and last is message size. Note that CAN message have length limited to 8 bytes.
- `unsigned char can_get_tec()` - Returns value of transmission error counter.
- `unsigned char can_get_rec()` - Returns value of reception error counter.

3.3 Device drivers

This section is concerned with design of device drivers. They use services of peripheral drivers to communicate with devices on PCB.

3.3.1 MPU-9150 and MPL3115A2 drivers

Both devices are connected to microcontroller using I2C bus. They are I2C slaves. As slave devices cannot start communication on I2C bus they use separate signal to inform the microcontroller that new data are ready.

Listing 3.4 contains excerpt from MPU-9150 driver. Function `mpu9150_setup` is used to initialize the device and underlying I2C driver. It takes single argument - a pointer to notifier function that shall be called when new measurements are available. Note that it creates `i2c_request` named `req_meas`. This request is preset to read accelerometric and gyroscopic measurements when executed. Also, it is further configured to execute `mpu9150_bh()` bottom half when transfer is finished.

The MPU-9150 device is configured to generate edge on its INT output when new measurements become available. This signal is routed to pin PB10 on the microcontroller. It is assumed that every time when rising edge appears on PB10 then function `mpu9150_irq` is called. This function will execute transfer request `req_meas` and return. When the request is finished the bottom half `mpu9150_bh` is invoked automatically by underlying I2C driver. As bottom halves are always executed in program context it can call notifier function to inform the application. Currently, we forward directly raw data read from MPU-9150. It contains 7 floating point variables encoded in 16-bit numbers. These are three accelerometric measurements, three gyroscopic measurements and temperature.

```c
static unsigned char raw[15];
static struct i2c_device *mpu9150;
static struct i2c_request *req_meas;
static void (*notifier_fcn)(unsigned char *raw);

void mpu9150_irq() {
  /* request the i2c transfer when rising edge on PB10 is detected */
  i2c_nonblocking(req_meas, &raw[0]);
}

int mpu9150_bh() {
  /* transfer is finished, raw[] has the data, notify the application */
  notifier_fcn(&raw[1]);
  return BH_OK;
}

void mpu9150_setup(void (*notifier)(unsigned char *raw)) {
  /* supposes that i2c is initialized and that mpu9150_irq is called
  when rising edge appears on IO pin where INT is routed to. */
  int bh = bh_register(mpu9150_bh);
  mpu9150 = i2c_create_device(0x68, 1);
  req_meas = i2c_create_read_req(mpu9150, I2C_GET_IMU_DATA_PRIO,
    INT_STATUS, 15, bh);
  notifier_fcn = notifier;

  /* configure the device */
  i2c_write_1blocking(mpu9150, PWR_MGMT_1, 0x00); /* go active */
  i2c_write_1blocking(mpu9150, SMPLRT_DIV, 5); /* max sample rate */
  i2c_write_1blocking(mpu9150, CONFIG, 0x02); /* dlpf mode 2 */
  i2c_write_1blocking(mpu9150, GYRO_CONFIG, 0x08); /* +-500 deg/s */
  i2c_write_1blocking(mpu9150, ACCEL_CONFIG, 0x08); /* +-4g */
  i2c_write_1blocking(mpu9150, INT_PIN_CFG, 0x30);
  i2c_write_1blocking(mpu9150, INT_ENABLE, 0x01);
}
```

Listing 3.4: MPU-9150 driver (simplified)

As you can see the driver is simple. It does not allow any flexibility in configuration as there is no need for universality (for the configuration see Table 3.2). Also, it does not implement any features we don't use at the moment. Future versions of this driver should convert retrieved data to floating point variables with correct units. We don't do that as the integration filter is not implemented yet - only thing we do with the data is that we forward them using Bluetooth to host device.

MPL3115A2 driver works on same principle. The device generates rising edge on ALTINT1 line connected to pin PA13 on the microcontroller (see schematic in Appendix C). The driver expects function `mpl3115a2_irq` to be called in response to that. This routine will execute non-blocking I2C transfer to read current pressure. The application is informed through bottom half that is called automatically when the transfer is finished. Function `mpl3115a2_setup()` is available to initialize the driver.

Sampling rate	166Hz
Gyroscope full-scale range	$\pm 500°/s$
Gyroscope sensitivity	$0.0152°/s$
Accelerometer full-scale range	±4g
Accelerometer sensitivity	122.07mg

Table 3.2: MPU-9150 configuration

3.3.2 NV08C-CSM driver

The GNSS receiver communicates with the microcontroller using two UART channels. First channel is set for NMEA-0183 protocol [22] and second for BINR protocol [17]. We implement both but use mainly BINR as it provides floating point numbers directly in IEEE 754 format. The NMEA channel is currently used only to command the receiver to stop reporting on that channel.

Application interface of the driver consists of two functions. First is `gps_setup()`. It configures the GNSS receiver. Single argument is required - pointer to notifier function that is used to inform the application when new data are available. Second function is `gps_process_incoming()`. It processes USART reception buffers on both channels. No arguments are necessary, but it needs to be called periodically.

Internally, function `gps_setup` configures the device according to Table 3.3 and requests messages in Table 3.4 to be sent periodically by receiver.

Data sent using BINR protocol are validated with cyclic redundancy check (CRC) that uses standardized CCITT-16 seed. Whenever new message is received the function `gps_process_incoming()` decodes it and checks the CRC checksum. If the message is one of those listed in Table 3.4, then the application is informed through the notifier callback.

Update rate	10Hz
Enabled systems	GPS, GLONASS, SBAS
Maximum acceleration	10m/s^2
BINR baudrate	230.4kBd

Table 3.3: GNSS receiver configuration

41h	Course angle and current speed
52h	Visible satellites
88h	Position, velocity and time vector
4Ah	Ionosphere parameters
64h	DOP and calculated uncertainties
F5h	Raw data (pseudo-ranges, pseudo-range rates, doppler)
F7h	Extended ephemeris of satellites

Table 3.4: messages sent by GNSS receiver

3.3.3 ISO 15031 support (OBD-II)

Support for communication with the vehicle over OBD-II cannot be called device driver as there is no device to control. We periodically read a single value from a vehicle's ECU connected on CAN bus through diagnostic port. This value is vehicle speed encoded as 8-bit unsigned integer.

Physical and link layers are based on CAN bus (ISO 11898). Network and transport layers are described in ISO 15765 (Road vehicles - Diagnostics on Controller Area Networks) and OBD-II services are subject of ISO 15031-5 (Road vehicles - Communication between vehicle and external equipment for emission-related diagnostics).

We need to send periodically a request for vehicle speed. According to ISO 15031, this request turns out to be a single CAN message. The vehicle shall respond in defined manner using only a single CAN message as well. Hence, for our limited needs we don't need to implement these standards, but only send periodically predefined message and check form of the response (if it comes). Message identifier for the request is `0x7DF` and the message consists of 3 bytes `0x2`, `0x1`, `0xD` followed by five zeroes. First byte specifies number of following bytes, second byte specifies requested service (Service $1 - Request current powertrain diagnostic data) and `0xD` is parameter ID of vehicle speed sensor.

Expected response can have identifiers in range from `0x7E8` to `0x7EF`. The message shall start with three bytes `0x3`, `0x41` and `0xD` followed by vehicle speed encoded in a single byte. First byte (value 3) specifies number of following bytes, second byte (`0x41`) says that the message is response to Service $1 request and third byte is the parameter ID.

Periodical reading of vehicle speed is based on alarms. Function `obd2_setup()` configures the CAN driver and the alarm. It takes pointer to notifier function as its single argument. Similarly like with other peripheral drivers, the notifier is used to inform the application that new data are ready. Vehicle speed is passed as its argument.

3.4 Embedded application

The application running on microcontroller continuously collects measurements from inertial measurement unit, from barometric altimeter, from vehicle (its speed) and from GNSS receiver (both positioning solution and raw data). For that it uses the device drivers described above.

Current version does not implement the integration filter. Its design and implementation is not in scope of this thesis.

3.4.1 Communication with host system

By host system we mean Android device connected via Bluetooth that runs navigational application. We refer to it as "master" while embedded system is called "slave". Master requests services, slave provides them.

Used communication protocol is described in Appendix B. Data are transferred by means of messages called data objects. Different data objects carry different information. For example, one data object can contain satellite navigation-based position, another latest inertial measurements, etc.. There can be up to 16 different data objects in use. Each data object can carry a message of 4 kilobytes.

We distinguish between process data objects (PDOs) and service data objects (SDOs). There is no actual difference in message format, but it allows to distinguish whether the data contain processing or configuration information. Process data objects contain most

ID	name	description
1	SDO1_MASTER_ALIVE	Sent periodically by master to inform slave of its presence. Message contains list of requested services that the slave shall provide.
2	SDO2_ALIVE_STATS	Sent periodically by slave to inform master of its presence. Message contains device state, diagnostic and statistic information.
3	SDO3_WEATHER	Weather-dependent corrections for barometric altitude meter. Sent by master.
8	PDO8_IMU	Latest inertial measurements.
9	PDO9_ALT	Latest altitude measurement.
10	PDO10_OBD	Latest vehicle speed.
11	PDO11_GPS	Forwards GNSS message.
12	PDO12_GPS_RAW	Forwards GNSS message that contain raw measurements (pseudo ranges, pseudo-range rates, etc..).
13	PDO13_INTEGRATED	Current position, velocity and related uncertainties.

Table 3.5: list of process/service data objects

up-to-date measurements while service data objects are used for configuration and reporting. List of all data objects is in Table 3.5, for their definition see Appendix B.

The slave is idle after system start. He is waiting for the master to report his presence by transmitting `SDO1_MASTER_ALIVE` data object. When slave receives this message it will consider master connected for next 2 seconds. Within this period master should report again, otherwise slave will consider him as disconnected when timeout expires. This is used for automatic detection of connection failures. Master has to periodically reset the 2 second timeout which will keep slave informed whether master is listening. When master drops out, slave will switch into disconnected mode and stop transmitting.

When master is connected the slave forwards requested measurements to him. Also, he sends his `SDO2_ALIVE_STATS` object every 500ms and blinks with heartbeat LED fast. When in disconnected mode, slave keeps collecting the measurements, but it does not forward them and the heartbeat LED blinks slowly.

The slave provides services to the master. There is one service that is provided always (when connected) and five that are optional. Which optional services are activated is selected by master in his `SDO1_MASTER_ALIVE` data object. As this message is periodic, master has to select each time which services he requests.

The slave transmits `PDO13_INTEGRATED` with period 200ms. It is the integrated positioning solution supposedly calculated by integration filter. As the filter does not exist yet, we forward position obtained from satellite navigation.

The other five optional services forward raw measurements from sensors for offline processing and analysis. Their names are `SERVICE_IMU`, `SERVICE_ALT`, `SERVICE_OBD`, `SERVICE_GPS` and `SERVICE_GPS_RAW`. Each service has associated PDO that is used to forward new measurements to the master.

3.4.2 Diagnostic features

Each information source (MEMS sensors, OBD-II and GNSS receiver) is monitored by imposing deadlines on its update period. These deadlines are designed to detect sensor

variable	**description**
`checksum_errors`	counts number of dropped data objects
`missed_irqs`	counts missed external interrupts
`can_unknown_messages`	counts unrecognized CAN messages
`rbuf_level`	DBGU reception buffer watermark
`tbuf_level`	DBGU transmission buffer watermark
`gps0_rbuf_level`	GNSS channel 0 reception buffer watermark
`gps0_tbuf_level`	GNSS channel 0 transmission buffer watermark
`gps1_rbuf_level`	GNSS channel 1 reception buffer watermark
`gps1_tbuf_level`	GNSS channel 1 transmission buffer watermark
`looptime`	looping-time watermark
`system_time`	system time, in milliseconds
`services`	bit flags for currently enabled services
`stability`	bit flags that indicate stability
`health`	bit flags that indicate healthiness

Table 3.6: in-run statistics and other diagnostic information (extract)

failure when it goes silent. If the sensor fails to provide update before the deadline, then it is flagged as both "unstable" and "not healthy". If the sensor wakes up again later and starts working normally, then we drop the "not healthy" flag, but it will be remain flagged as "unstable". In this way we can easily observe reliability of the sensors. This functionality is implemented with task `health_observer` that runs with period of 70 milliseconds.

Furthermore, `health_observer` also checks whether the system missed any external interrupts. Both interrupt signals (INT and ALTINT1, see schematic in Appendix C) are level-triggered while the microcontroller has edge-triggered interrupts on I/O lines. The edge might potentially go unnoticed which would cause issuing sensor to go silent. To detect this, task `health_observer` checks state of both interrupt lines and recovers from error if interrupt was missed.

The program also collects numerous in-run statistics. These can be used to locate problems. For example, one of the things we observe is number of checksum errors in communication. One can expect that these errors are rather rare, but if the count grows unnaturally fast, then it might indicate problem with the transmission lines or a problem with involved algorithms. Hence, these statistics are most valuable during development to observe whether the device behaves as expected.

Collected statistics are mostly concerned with counting error conditions, watermark levels of buffers, etc. Watermarks are buffer load maxima observed in some period of time. Simply said, it indicates how close it came to overflow. Similarly, we use looping time watermark which is maximum time it took the program to finish single iteration of its main loop.

See Table 3.6 for list of in-run statistics program generates. Data structure that holds them is declared in `stats.h`. Various accessing functions are available in `stats.c`. Snapshot of these statistics is also part of `SDO1_ALIVE_STATS` data object.

Chapter 4

Android application

This chapter is concerned with implementation of host system. In general sense this should be Java library that provides road-referenced position to higher-level services such as route planner for example. In specific case of this project we also needed presentation layer that would show both integrated position and map-referenced position on a map so we can validate behavior of the map-matcher.

As a result, we designed application "rvn-app" that runs map-matching algorithm and shows the output on a map. Additionally, it logs measurements for offline analysis.

Section 4.1 deals with road-network graph representation. We analyze various ways to deal with graphs that feature millions of vertices and present the best solution. Section 4.2 describes the map-matching algorithm and Section 4.4 the rvn-app application.

4.1 Road-network representation

We have experimented with various ways to store road-network graph. As was defined in Section 2.5, road network graph is directed graph $G(V, E)$ where each vertex V has assigned position in terms of latitude/longitude and each directed edge represents road between the two vertices. By taking position of the two vertices connected by an edge we can construct line segment between them that represents road "centreline". Direction of the edge defines driving direction.

This model allows to encode complete road-networks, their curvature and allowed driving directions. Although not optimal in terms of data set size, this is minimalist and most simplest representation which can fulfill needs of any road-network. Other authors use layered representation with list of known roads where each road has its own list of geographic points. Even this representation is somewhat intuitive, it is more structural than analytic. Our model is simple, elegant and allows to apply graph theory on it.

No matter the representation, these graphs are large. For example road-network graph of Czech Republic has 4 millions vertices and about 10 million indices (directed edges). Important question that remains to be answered is how to store it. One solution is to use graph databases that are designed to handle this type of data structures, but we didn't find any free implementation on Android.

Our first attempt was to store the graph as separate lists of vertices and indices to binary file. This leads to format where vertices have assigned their geographical position and indices refer to positions in array of vertices. First few bytes encode number of vertices and indices in following lists. Then, list of vertices is stored, having each vertex is represented by two floating point numbers, latitude and longitude. List of indices follows where each edge is represented by two integer numbers - indexes of starting and ending vertices in the vertices list.

We wrote utility program named "osmer" that generates such a road-network graph using OpenStreetMaps map data. Generated road-network of Czech Republic had around 60MB. Nonetheless, problem of this approach is that complete graph has to be loaded to RAM. This has shown unfeasible as it took minutes to load complete graph to memory. Moreover, object-oriented data structures that were holding the graph have shown to be much larger than the file itself[1].

Need to use temporal cache with only parts of the graph became apparent. This would require the file to hold vertices list sorted and indexed by both latitude and longitude which, in consequence, would prevent to use simple list of indices to describe edges of the graph. For this reason we changed approach and used SQL database. It supports indexing while it can manage cross-links between vertices and indices lists.

So, second attempt was to store the road-network graph using SQLite3 embedded database. We used tables `vertices` and `indices`. Table `vertices` contain fields ID, latitude and longitude. Table `indices` contain IDs of starting and ending vertices. SQL's LEFT JOIN relation can be used to traverse the graph. For example, one could use SQL command in Listing 4.1 to find vertices adjacent to vertex with ID=10.

```
SELECT *
   FROM vertices v
   LEFT JOIN indices i
   ON v.id = i.end_v
   WHERE i.start_v = 10;
```

Listing 4.1: retrieving adjacent vertices

We wrote utility program named "osmer-sql" that implements this. Problems with start-up time due to graph loading were solved, but query from Listing 4.1 have shown to be slow on full-sized road-network graphs.

We have attacked this problem by pre-computing adjacent vertices for each vertex and encoding them into the `vertices` table. Table `indices` was dropped and `vertices` was extended with field that holds string with encoded IDs of adjacent vertices.

Base64 encoding was used to encode binary 24-bit vertex IDs into 4 textual characters. Hence, each four characters in the string encode single ID of adjacent vertex.

In this way we are not bounded to use relations anymore to retrieve adjacent nodes. It is also beneficial as we can read list of adjacencies of a vertex without needing to load them immediately. This allows to create cache where adjacencies of all cached vertices are known by their ID, but can be loaded later as needed.

The cache is implemented in class `RoadNetwork` in package rvn-app. It caches the road-network graph using `TreeMap` that maps vertex IDs to instances of `Node` class (that holds information about the vertex). We chose `TreeMap` because it allows to search for `Node` instance according to given ID in O(log(n)) time.

[1]Instance of `Object` class has 16B - this is theoretical minimum per any class instance as all classes are inherited from `Object` in Java.

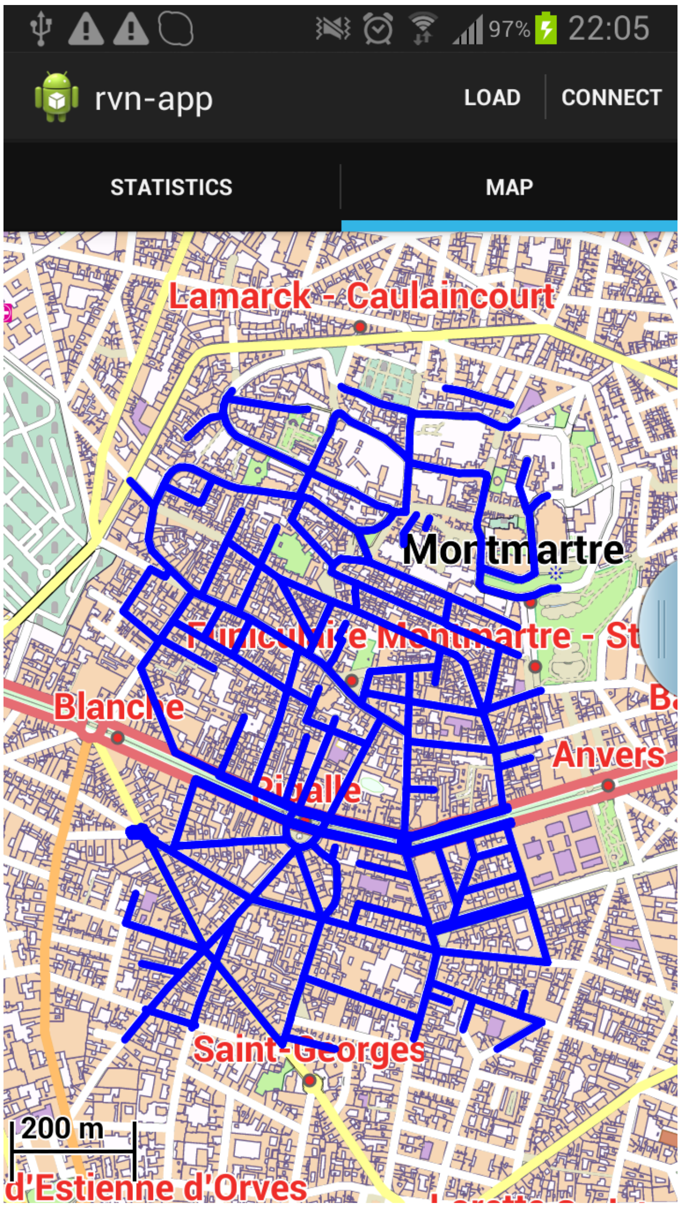


Figure 4.1: Blue lines represent edges of road-network graph around Montmartre, Paris returned by method `cacheArea()`.

Class `Node` contain geographical position of the vertex (latitude, longitude) and references to adjacent vertices (instances of `Node`). Also, it features boolean field `loaded` that indicates whether the vertex was loaded from database. If not, then the instance of `Node` is empty placeholder serving as "connector" between cached and not cached parts of the graph[2].

`RoadNetwork` class features a number of methods that the program can use for caching. Method `cacheArea` is used to cache road-network over rectangular area given as variation of latitude and longitude around some central point, see Figure 4.1. Other method, `cacheNode`, allows to cache single vertex identified by ID, or by instance of `Node` class that has not been loaded yet. Finally, method `cacheAdjacent` allows to cache adjacent vertices to given vertex.

[2]Set of not loaded `Node` instances constitutes a graph cut known from graph theory.

4.2 Map-matching algorithm

Our algorithm was designed with simplicity in mind. It can be categorized as mutation of geometric, topological and probabilistic approaches, taking the best features of each. Due to its simplicity and generality the algorithm can be easily adopted by other systems.

It requires the navigation system to provide position and velocity plus related uncertainties. Note that the velocity is used to compute traveled distance and can be easily replaced with difference between two consecutively reported positions, see Section 4.2.5.

In accordance with our analysis, we haven't designed the algorithm to refine reported position. Instead, we leave outlier removal and filtering to the underlying navigation system. The reasons are discussed in detail in Section 2.5.2. This means that both position and velocity is used to identify correct road segments but not to filter position of the vehicle along them.

Main principles are as follows: there is point-to-curve geometric algorithm (see Section 2.5.2) that is active during track initialization and track loss (see figure 2.19). When possible tracks are identified, then algorithms are switched and main algorithm that uses multiple hypotheses technique develops them as vehicle proceed along road-network graph. The algorithm is completely recursive. Likelihood function that maintain discrepancy between measurements and each hypothesis is used. Note that algorithms are automatically switched back on track loss so geometric algorithm can immediately identify new set of possible hypotheses. This allows to recover from failure as soon as its effect is gone.

4.2.1 Notion of closeness

"Closeness" of road segment to measured position p_g is observed as distance between p_g and its perpendicular projection on the line segment (see Figure 4.2a). In case projected point lies outside the segment we take distance to the end of the segment that is closer (see Figures 4.2b, 4.2c).

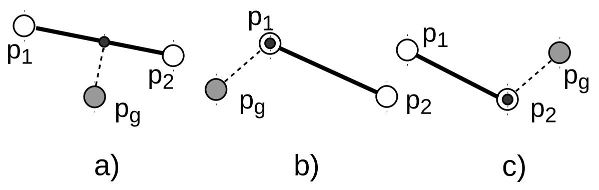


Figure 4.2: Cases that might occur when computing closest point on line segment.

Computing the position of the closest point on line segment is matter of simple geometry. If we want to find projected point p_l on the segment such that it is the closest point to the measurement we can take $n = p_2 - p_1$ and construct a line with parameter t. Closest point from the line to the measured position p_g can be found by means of obtaining parameter t from equation (4.1).

$$t = \frac{(n \cdot p_g) - (n \cdot p_1)}{n^T n} \tag{4.1}$$

if $t \in \langle 0, 1 \rangle$ then we have case on Figure 4.2a and point $p_l = nt + p_1$. If $t < 0$ then $p_l = p_1$ (case on Figure 4.2b) and, finally, if $t > 1$ then $p_l = p_2$ (Figure 4.2c).

With ability to find closest point to measurement on a line segment we can easily find closest point to measurement on hypothesis. As hypothesis is essentially a sequence of linear segments we just need to identify line segment that is closest to the point.

4.2.2 Notion of distance along hypothesis

For purposes of our algorithm we also need means to measure distance between two points along hypothesis. Again, this task is easy as hypotheses are piece-wise linear. We just need to identify road segments on which the two points lay and then we can compute distance by summing Euclidian distances between the points and nodes in between (see Figure 4.3).

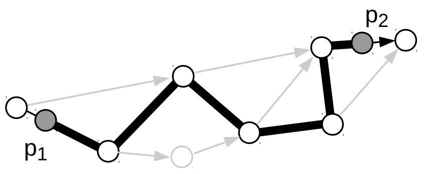


Figure 4.3: An example with two points on some hypothesis for which we want to compute distance along the hypothesis. Length of the thick line is what we seek to compute.

4.2.3 Notion of probabilistic gating

We borrow term "gate" from [8] where it was used to account for uncertainty along measured position. For our algorithm these gates are spheres defined by their origin and radius. Origin is the reported position and radius is based on uncertainty. We use radius $r = \max(20, 3\sigma)$ with σ being standard deviation. Assuming normally distributed measurements the 3σ term gives 99.7% probability that measurement error is bounded by the region. By taking at least 20-meter radius we account for displacement between used driving lane and road centreline.

It is important to know what road segments are intersecting with the gate in order to identify candidate roads. It can be done by testing for line-sphere intersection. Combining line and sphere equations leads to quadratic equation which is straightforward to solve. First, consider equation of a line constructed from positions p_1, p_2 of two nodes along an edge on the road network graph and a sphere with center at measured position p_g and radius r.

$$(p_2 - p_1)t + p_1 = p_l \tag{4.2}$$

$$||x - p_g||^2 = r \tag{4.3}$$

where p_l is a point on the line and x is simply a point on the surface of the sphere. Then, we can plug equation for a line to the equation of a sphere in order to find intersections.

$$||(p_2 - p_1)t + p_1 - p_g||^2 = r \tag{4.4}$$

By solving for t we obtain quadratic equation:

$$||p_2 - p_1||^2 t^2 + 2(p_2 - p_1)^T (p_1 - p_g) + ||p_1 - p_g||^2 - r^2 = 0 \tag{4.5}$$

We can test for intersection existence by testing for non-negativity of discriminant. Note that it is not enough to know whether the intersection exist, it is also important to check whether the intersection happens to be on the segment (part of the line) that constitutes the road. Hence, there is intersection of road segment with uncertainty sphere if discriminant is non-negative and at least one solution of t is in range $\langle 0, 1\rangle$.

4.2.4 Maintaining hypotheses

The algorithm have to continuously update its set of candidate hypotheses as measurements come. Convenient way to do this is to use probabilistic gating.

New hypotheses can be generated simply by branching older ones along the edges of the road network graph. Gate region can be used as criterion for branching. For algorithm that branches single hypothesis see Algorithm 1.

Algorithm 1 Branching hypotheses

```
procedure BRANCH(hypot)
    $n \leftarrow$ last node in $hypot$
    if $n$ inside the gate then
        for each adjacent edge $adj$ of $n$ do
            $newhypot \leftarrow$ CLONE_HYPOTHESIS(hypot)
            EXTEND_HYPOTHESIS($newhypot$, $adj$)
            BRANCH($newhypot$)
```

Function CLONE_HYPOTHESIS($\cdot$) creates copy of given hypothesis and function EXTEND_HYPOTHESIS($hypot$, adj) extends hypotheses by adding edge adj to the end of the sequence. Note the recursive call for BRANCH(). It works as safeguard mechanism against short road segments. It is possible that a single hypothesis needs to be branched more times in case the road segments are comparatively smaller than gate radius.

Admittedly, it is not preferable to have recursive functions running on small embedded systems with limited stack sizes. It is possible to remove the recursive call from the algorithm but chances are that it will loose track from time to time. Nonetheless, if the fast failure recovery is implemented correctly, then the algorithm should recover within single iteration (see Section 2.5.2).

Pruning hypotheses can be done by observing distance between hypothesis and latest reported position (see Section 4.2.1). If the distance is larger than gate radius then the hypothesis can be safely removed.

4.2.5 Likelihood function

Likelihood function is designed to compute "likeliness" of single hypothesis with respect to measurements. Geometric displacements are accounted for with positioning measurements. Heading discrepancies are accounted for with distance measurements.

Positioning error $\delta_n(h_i)$ is computed as magnitude of displacement between reported position p_n and its closest point on hypothesis h_i (see section 4.2.1).

Velocity measurements are used to compare distance observed on hypothesis with distance traveled by the vehicle. We convert velocity to distance by multiplying it with updating period τ. Note that small number approximation is applied here, so velocity update rate has to be sufficiently high.

For comparison we take current and previous reported positions and find their closest counterparts on hypothesis. Having the two points projected on hypothesis h_i we can compute distance along the hypothesis $\vartheta_{n,n-1}(h_i)$ (see Section 4.2.2).

Difference between $\vartheta_{n,n-1}(h_i)$ and observed distance acts as both velocity and heading error indicator, see Figure 4.4. Its advantage over classically used heading based on GNSS [12] is that this version is more stable. Indeed, GNSS heading is not reliable unless the vehicle is moving relatively fast (more than 3m/s according to [3]). Our solution doesn't rely on direction of positioning data but on their projection on different hypotheses.

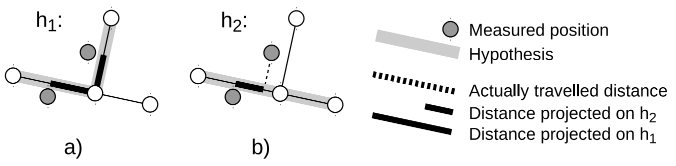


Figure 4.4: Principle of traveled distances observed on different hypotheses. Hypothesis h_1 is clearly the most likely one if we take the length of dotted line as actually traveled distance and compare it to the length of distances projected on both hypotheses.

In order to combine observations from both sources optimally we propose two factors a_n and b_n such that $a + b = 1$. They are "trustworthiness" ratios based on uncertainties reported by navigation system. Taking σ_p as uncertainty in position (in meters) and σ_v as uncertainty in velocity (in meters per second) then the formulas for a_n and b_n are as follows:

$$a_n = \frac{\sigma_v \tau}{\sigma_v \tau + \sigma_p} \qquad b_n = \frac{\sigma_p}{\sigma_v \tau + \sigma_p} \tag{4.6}$$

With a_n representing our trust in position and b_n our trust in velocity. Note that these coefficients are unitless. With them at hand, we are ready to introduce inverse of likelihood function:

$$s(h_i)^{-1} = \frac{1}{N} \sum_{n=2}^{N} a_n \delta_n(h_i) + b_n(v_n \tau - \vartheta_{n,n-1}(h_i)) \tag{4.7}$$

with N being total number of measurements included and v_n reported velocity for measurement n. Note that this function is well-formed since all terms are in meters or unitless. Resulting value can be understood as average error of all measurements with respect to hypothesis h_i.

Note that likelihood in (4.7) is inversed. It is so because maximum likelihood hypothesis is the one with minimum error (error is represented by right hand side). In practice we can simply skip the inversion and search for hypothesis that minimizes the right hand side.

Moreover, in practice the normalization to N is not necessary and can be omitted. The function will loose its real-world meaning but the most likely hypothesis is still the one with maximal likelihood as N is always the same for all hypotheses.

4.2.6 Algorithm workflow

Algorithm runs each time when new measurement arrives, see Figure 4.5 for workflow diagram. It's asymptotic complexity is linear with number of active hypotheses (experiments show that most of the time there is actually only one hypothesis). Supplementary geometric algorithm that implements fast failure recovery (see Section 2.5.2) runs also in $O(n)$ time with n being number of edges in vicinity.

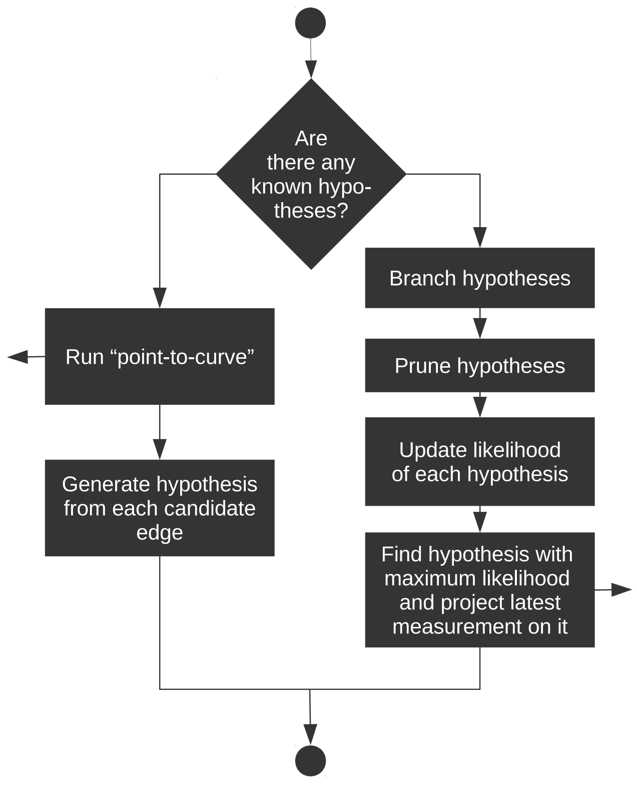

Are there any known hypo-theses?
Branch hypotheses
Run “point-to-curve”
Prune hypotheses
Update likelihood of each hypothesis
Generate hypothesis from each candidate edge
Find hypothesis with maximum likelihood and project latest measurement on it

Figure 4.5: Algorithm workflow. Each step is described in detail in Section 4.2 with exception of point-to-curve algorithm that is described in Section 2.5.2.

4.3 Map-matching algorithm simulations

We have run simulations based on real measurements and real maps before implementing it in Java. We have used data provided by our embedded system (see chapter 3). Specifically, we have used position from GNSS receiver and vehicle speed read via OBD-II interface. Uncertainty in position is obtained from GNSS receiver as well. Uncertainty in vehicle speed is modeled as 5% from reported velocity[3]. It accounts for scaling errors due to pressure in the tires, tire wear, etc..

We have recorded measurements in rural and urban areas. The simulation is written for GNU Octave[4] (version 3.8.1).

[3]We have estimated the actual scaling error in the data to be about 1% (with Kalman filter that used information from GNSS to observe it, see Section 2.3.1).

[4]Language for numerical computations mostly compatible with MATLAB.

4.3.1 Tests in rural areas

The route we tested was recorded in rural areas of Czech Republic on regional road that connects number of villages in vicinity. The land is mostly flat and about 20% of the route is covered with tree canopies of various thickness.

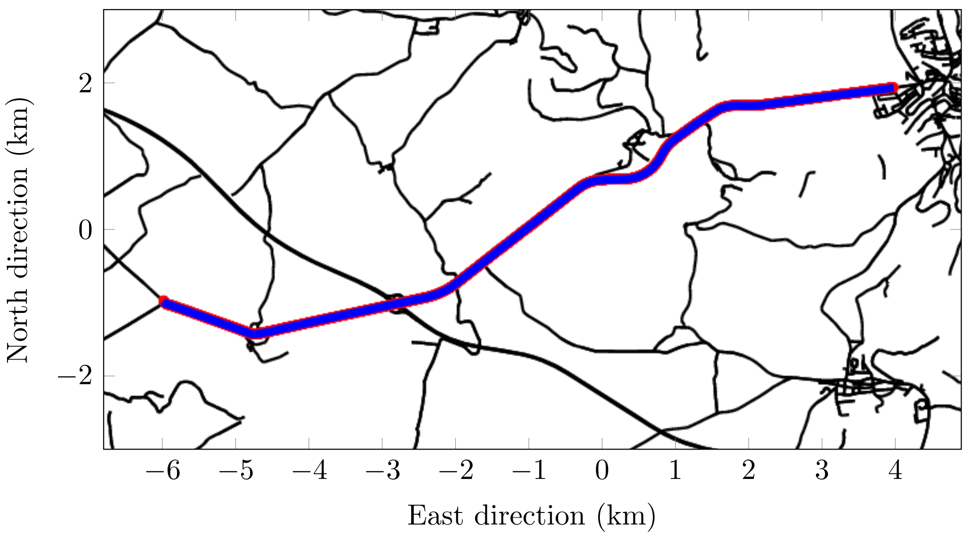


Figure 4.6: Test in rural area, blue line is map-matched route.

Simulation involved 3309 measurements from which none were falsely matched. When on junction the algorithm always chose the right path on first try. If situation is unclear the algorithm tends to stay on the junction and wait for the situation to clarify with forthcoming measurements.

4.3.2 Tests in urban areas

Tests in urban areas were designed to span small streets with tall buildings around, streets running closely in parallel, various squares as well as open spaces with good visibility.

Urban areas are difficult because satellite navigation provides lower-quality signal and increased number of junctions makes the task more involved for the algorithm. See Figure 4.7 for tested route.

Simulation of the route on Figure 4.7 involved 2138 measurements. By manually reviewing the results we have found that there were two occasions where the algorithm chose wrong turn on junction. The wrong match lasted 1.4 seconds in a first case and 800 milliseconds in second case. Scenario that caused the error was in both cases similar - increased noise due to disabled filtering of GNSS position together with measurements biased to the right side in the direction of driving (due to driving in right lane) caused the algorithm to wrongly match turn when there was junction with straight and right turn choices.

What we observed with rural areas has shown up in this test again - the algorithm tends to make sober decisions by waiting on the junctions until it is clear which direction vehicle actually took.

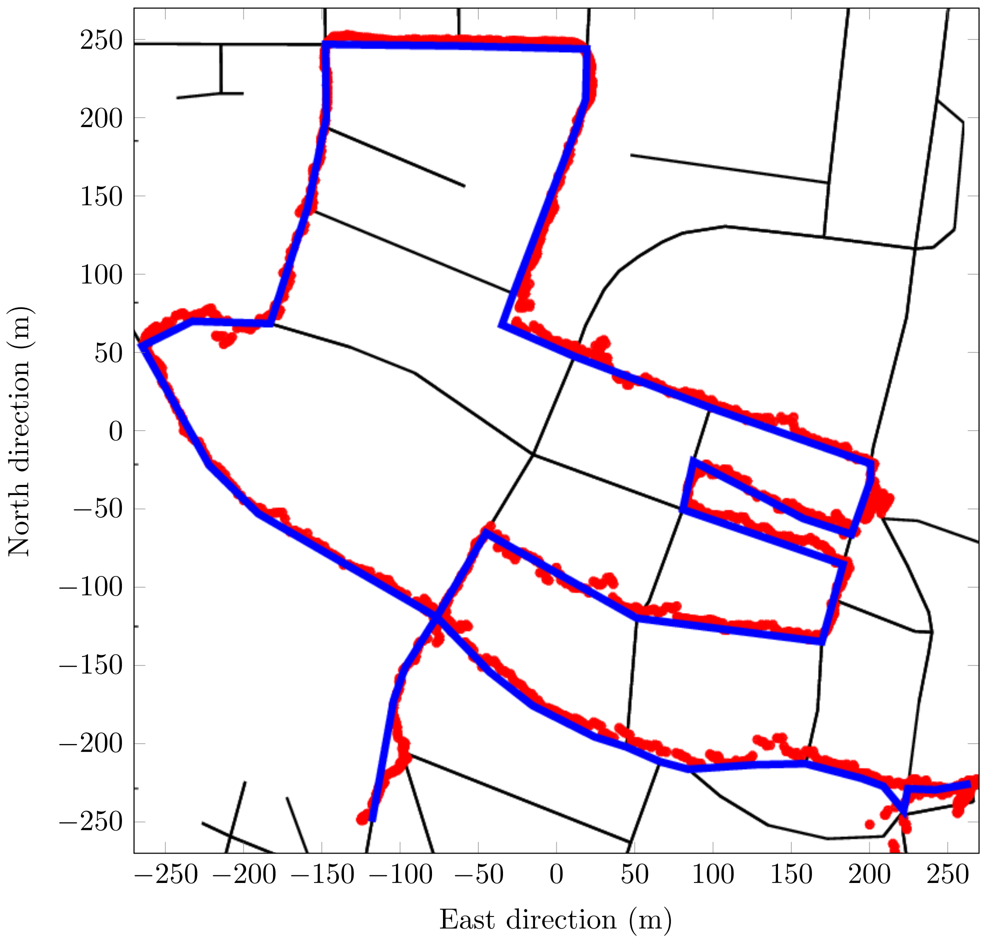


Figure 4.7: Test in urban area. Blue line is map-matched route for test in urban area, red dots are successive reported positions.

4.4 Road-vehicle navigation application (rvn-app)

Android application rvn-app is a prototype of host system. It implements road-network cache (Section 4.1), map-matching algorithm (Section 4.2), communication with embedded system (Section 3.4.1) and presentation layer that shows road-referenced position on classical map.

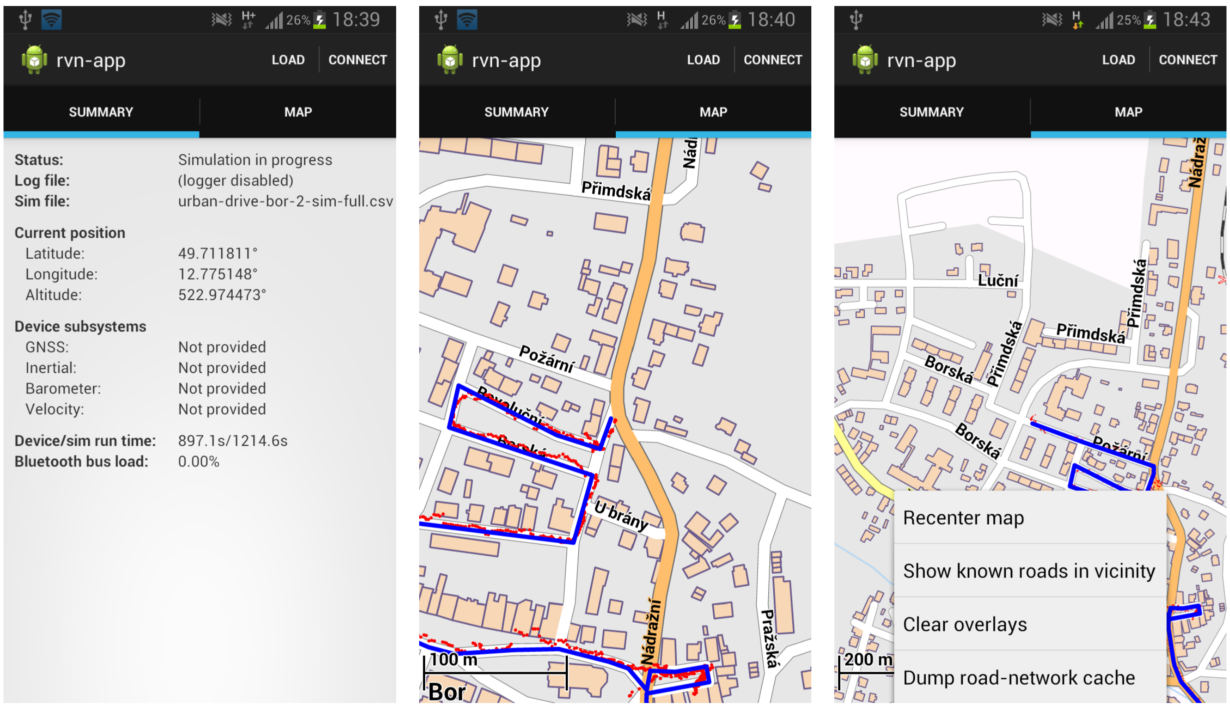
Moreover, rvn-app automatically creates communication logs, allows to replay them and allows to inspect structure of road-network graph. Following paragraphs describe features of rvn-app in detail.

For application layout see Figure 4.8. Buttons "load" and "connect" lay in top-right corner. Former is used to run simulation and latter to connect to embedded system.

When button "connect" is pushed the application will try to search for embedded system. If found, it will establish connection and start transmitting SDO1. Note that embedded system needs to be paired with Android device and Bluetooth has to be enabled.

When button "load" is pushed user is presented with dialog to choose simulation file. This file is simple CSV table where each line convey same information as PDO13. It can be generated from communication logs with log2csv utility (see Appendix A.1). Simulations are useful to test the map-matching algorithm.

See Figure 4.8a for layout of "summary" tab. This tab shows various information about rvn-app's operation. Status field specifies current operation - either that it is connected



Status: Simulation in progress
Log file: (logger disabled)
Sim file: urban-drive-bor-2-sim-full.csv

Current position
Latitude: 49.711811°
Longitude: 12.775148°
Altitude: 522.974473°

Device subsystems
GNSS: Not provided
Inertial: Not provided
Barometer: Not provided
Velocity: Not provided

Device/sim run time: 897.1s/1214.6s
Bluetooth bus load: 0.00%

(a) "summary" tab **(b)** "map" tab **(c)** contextual menu

Figure 4.8: rvn-app layout

to embedded device, runs simulation, or that the program is idle. Fields "log file" and "sim file" specify files in external storage where incoming Bluetooth stream is logged and where the simulation data are taken from. Fields under "device subsystems" header indicate healthiness and stability of embedded system subsystems when connected. Field "device/sim run time" shows current time on the embedded system or simulation time when simulation is running. Finally, field "Bluetooth bus load" indicates current load of Bluetooth channel.

Figure 4.8b shows "map" tab. It features rendered map based on OpenStreetMaps (OSM) data. We use mapsforge library [25] to render both map and overlays. The overlays are used to show reported positions (red dots) and maximum likelihood hypothesis (blue line) in context of the map. Map data are stored in a file on device's external storage. The library uses own maps based on regional OSM extracts. Note that enclosed DVD contain maps of Île de France and of Czech Republic.

Figure 4.8c shows options available through contextual menu. Option "Recenter map" is used to recenter map around latest reported position. Option "Show known roads in vicinity" shows known roads in current view (such as on Figure 4.1) when ticked on. It can be used to visually inspect structure of road-network graph. Option "Clear overlays" does exactly what the title says - clears all overlays. Last option "Dump road-network cache" is used to dump current content of road-network cache into files "road-network-cache-dump.nodes" and "road-network-cache-dump.edges" in device's external storage. These files contain cached part of road-network that can be loaded by Octave or MATLAB and used for analysis and simulations.

This page is intentionally left blank.

Chapter 5

Conclusion

This work started with thorough analysis of the road-vehicle navigation problem. Outcome of this analysis was a proposal of optimal road-vehicle navigation system in terms of performance. Special embedded system with sensors, navigation systems and communication interfaces of interest was designed and manufactured to run tests on the devices. Test results are shown and discussed in the analysis.

Part of the proposal was recommendation to design integrated navigation system running on our own embedded system instead of using sensors embedded in Android smart devices as my supervisors were planning originally. The embedded system used for analysis has proven sufficient for this job as all sensors were on board, it was fully functional and able to work for days without failure. This has saved us plenty of time as I didn't need to design new embedded system based on knowledge gained from the analysis.

The optimal road-vehicle navigation system should integrate satellite-based and inertial-based navigation systems. The analysis details how this should be done, but the implementation in following chapters does not do the integration. This is so because there was not sufficient time to implement it. Nonetheless, it does not mean that the project would not be finished. The road-vehicle navigation system works, only sub-optimally. This was expected as it is not feasible to develop road-vehicle navigation system according to the proposal in given time.

Important part of the road-vehicle navigation system is map-matching algorithm. I conducted review of current map-matching algorithms as part of the problem analysis. Nonetheless, I didn't find sufficiently robust algorithm. For this reason I developed under lead of my supervisors novel map-matching algorithm that is described in Chapter 4. The algorithm is proposed in our conference paper "On designing robust real-time map-matching algorithms" that was submitted to IEEE ITSC2014 conference. At the time of writing this thesis the paper was accepted.

Final chapter in this thesis describes application "rvn-app" for Android smart devices. It implements prototypical road-vehicle navigation system by integrating navigation system based on our embedded system with map-matcher running on the device.

This page is intentionally left blank.

Appendix A

Software utilities

We have developed number of utilities while working on this project. This addendum describes how to use them. Source codes and compiled binaries (for `x86_64-linux-gnu`) are enclosed on the DVD.

A.1 Dump measurements (log2csv)

Tool log2csv is used to dump measurements from communication logs. It takes binary log file and generates human-readable CSV table with measurements it contains. The tables can be used for offline analysis. A number of Octave scripts that plot the measurements is available on enclosed DVD. Usage instructions are printed when invoked with no arguments, see Listing A.1.

```
matej@lenovo:~$ log2csv
Usage: log2csv {gps|raw|ins|alt|obd2|eph|iono|sim} <log-file> <csv-file>
```

Listing A.1: log2csv usage instructions

First argument is output type specifier. There are eight options - `gps`, `raw`, `ins`, `alt`, `obd2`, `eph`, `iono` and `sim`, see Table A.1 for their description. Second argument is path to log file and third is output file.

`gps`	PDO11	position, velocity, time, course and related uncertainties as well as detailed information about all visible satellites
`raw`	PDO12	GNSS raw measurements (pseudo-ranges, etc..)
`ins`	PDO8	inertial measurements (accelerometer, gyro)
`alt`	PDO9	altitude measurements
`obd2`	PDO10	vehicle speed
`eph`		satellite ephemerides
`iono`		coefficients of Klobuchar's ionosphere model
`sim`	PDO13	generates data for map-matching simulation in rvn-app

Table A.1: output type specifiers

The log files are automatically generated by rvn-app and stored to device's external storage memory. Their names follow pattern "`rvn-log DD-Month YYYY HH:MM:SS.bin`".

For exemplary output see Table A.2. The program prints summary of log file contents.

```
matej@lenovo:~$ log2csv sim "rvn-log 16-June 2014 15:13:11.bin" out.csv
Processed: 2458875 bytes
Ignored: 976 bytes (0.039693%)
Checksum errors: 0 (0.000000%)
Received alive stats SDOs: 747
Received imu PDOs: 62174
Received alt PDOs: 386
Received obd2 PDOs: 3683
Received gps PDOs: 11580
Received gps raw PDOs: 0
```

Listing A.2: log2csv output example

A.2 Combine measurements (csvcomb)

This tool combines two tables generated by log2csv into one. Resulting data are combined in such a way that measurements from both tables relate to same time instants. This is useful for development and simulations of integrated filters. Usage instructions are listed in Listing A.3.

```
matej@lenovo:~$ csvcomb
Usage: csvcomb <csv-file> <csv-file> <output-file>
This program expects first column in both csv tables to be timestamp of
the measurement that follows. It combines the records from both files
```

Listing A.3: csvcomb usage instructions

A.3 Compute road-network graph (osmer-sql)

This program takes OSM map and extracts road-network graph from it. The graph is stored into Sqlite3 database file. For details on its operation see section 4.1. It takes two arguments. First is OpenStreetMap XML file and second is the output file.

The map can be obtained freely from `planet.osm.org` website. Note that full-planet map has about 500GB so using regional extracts is advisable. Note that even regional extracts can have tens of gigabytes.

```
matej@lenovo:~$ osmer-sql
Usage: osmer-sql <osm-map.xml> <output.sqlite3>
```

Listing A.4: osmer-sql usage instructions

Enclosed DVD contains OSM extracts of Czech Republic and of region Île de France in France (dated April 2014). Note that both files are compressed with gzipped tarballs. See Listing A.5 for program output when processing the map of Czech Republic. This map has 9.2GB, its processing took 1.8GB of RAM and 5 minutes 11 seconds on regular PC. Resulting SQLite3 file has about 350 megabytes (most of which are indexing tables).

```
matej@lenovo:~$ osmer-sql czech-republic-latest.osm czech.sql
Collecting geo-points: 40485674
Sorting...
Analyzing map polylines/routes: 4931584/546261
Looking up junctions: 615636
Saving 4234630 nodes into czech.sql using sqlite3
```

Listing A.5: osmer-sql output when run on a map of Czech Republic

This page is intentionally left blank.

Appendix B

Communication with host - protocol

Communication between the embedded system and the host system is based on data objects. Format of a data object is shown on Figure B.1 below. It has three parts - header, data and footer. Header has 5 bytes in total. First three bytes are fixed `0x13`, `0x27` and `0xC8`. This sequence marks beginning of a new data object[1]. Last two bytes of the header encode data object identifier (field DO; upper 4 bits) and message size (filed SIZE; lower 12 bits). Note that 4 bits in DO field allow for 16 different data objects and 12 bits of SIZE field limit message size to 4kB.

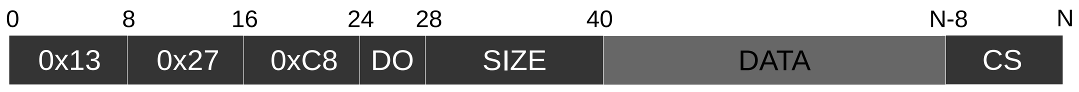


Figure B.1: process/service data object frame

Message data follow after the header. Receiver counts number of received data bytes and when complete message is received it expects 1-byte checksum. The checksum is computed as sum of all data bytes in the message modulo 256 (lowest 8 bits of the sum).

This type of checksum is particularly ineffective, but it is fast to compute while sufficient for our purposes. It is so because Bluetooth uses its own data validation mechanism (CRC checksum) and transmission line between the microcontroller and Bluetooth module is only few millimeters long (see Figure C.1).

B.1 Data objects definition

Following subsections specify data objects currently in use. Data types designated as `uintXX` are unsigned integers of XX bits. Data types `float` and `double` are standard C types for floating point numbers encoded according to IEEE 754. All variables are sent in little endian fashion except for some variables in PDOs 8 and 9. These PDOs contain raw measurements that are register bank snapshots of device they originate from.

Several PDOs contain variable `timestamp` typed as `uint48`. In order to convert it to seconds multiply `timestamp` with $\frac{32}{MCLK}$. MCLK is system clock (48092000 Hz).

[1]its suitability was verified by statistical analysis of 1-megabyte communication sample

B.1.1 SDO1 - Master alive

This message is periodically sent by master to declare its presence (master is the host system, slave is the embedded system, see Section 3.4.1). It contains two bytes with list of requested optional services. Second byte is bit inverse of first (for increased security).

#	field	type	comment
0	`services`	uint8	
1	`inv_services`	uint8	bit-wise inverted `services`

Table B.1: fields in SDO1

Relevant bit positions for `services` field are defined in `stats.h`. Here is excerpt:

```
#define SERVICE_IMU 7 /* enables PDO8 */
#define SERVICE_ALT 6 /* enables PDO9 */
#define SERVICE_OBD 5 /* enables PDO10 */
#define SERVICE_GPS 4 /* enables PDO11 */
#define SERVICE_GPS_RAW 3 /* enables PDO12 */
```

Listing B.1: bit positions in `services` field

B.1.2 SDO2 - Slave alive, statistics

This message is periodically sent by slave whenever master is connected. It has period of 500 milliseconds and contain diagnostic information. Its purpose is to acknowledge SDO1 sent by master. For format see Table B.2, meaning of the fields is explained in Section 3.4.2.

#	field	type	comment
0	`checksum_errors`	uint16	
2	`missed_irqs`	uint16	
4	`ignored_bytes`	uint16	
6	`can_unknown_message`	uint16	
8	`rbuf_level`	uint16	
10	`tbuf_level`	uint16	
12	`gps0_rbuf_level`	uint16	
14	`gps0_tbuf_level`	uint16	
16	`gps1_rbuf_level`	uint16	
18	`gps1_tbuf_level`	uint16	
20	`looptime`	uint32	multiply with $\frac{32}{MCLK}$ for seconds
24	`services`	uint8	
25	`stability`	uint8	
26	`health`	uint8	
28	`system_time`	uint32	in milliseconds
32	`nmea_frame_errors`	uint16	
34	`nmea_checksum_errors`	uint16	
36	`nmea_ignored_bytes`	uint16	
38	`nmea_unknown_messages`	uint16	

Table B.2: fields in SDO2

Size of SDO2 is 40 bytes. Note that there is 1-byte gap between fields `health` and `system_time`. It pads `system_time` to 4-byte boundary (as its type is uint32). Bit fields `services`, `stability` and `health` are encoded according to Listing B.1.

B.1.3 SDO3 - Weather correction

This SDO is used by master to push differential corrections for barometric altitude meter. These corrections can be used to correct for its bias errors. Message has one field - pressure at mean sea level in pascals (encoded as uint32). Size of SDO3 is 4 bytes.

In general case one cannot measure pressure at mean sea level directly. Nonetheless, if the reference pressure sensor is at known altitude it is possible to use atmosphere model described in Section 2.4 to estimate pressure at mean sea level.

B.1.4 PDO8 - Inertial measurement unit raw data

This PDO carries latest data from MPU-9150 inertial measurement unit. It is sent by slave whenever new data arrive. Content of the message is MPU-9150's register bank snapshot (registers from `0x3B` to `0x48`). See MPU-9150 register map [23] for encoding of the values. This is raw data, slave does not alter them in any way. Size of PDO8 is 20 bytes.

#	field	type	comment
0	`acceleration_x`	uint16	big endian; abnormal coding
2	`acceleration_y`	uint16	big endian; abnormal coding
4	`acceleration_z`	uint16	big endian; abnormal coding
6	`chip_temp`	uint16	big endian; abnormal coding
8	`gyroscope_x`	uint16	big endian; abnormal coding
10	`gyroscope_y`	uint16	big endian; abnormal coding
12	`gyroscope_z`	uint16	big endian; abnormal coding
14	`timestamp`	uint48	

Table B.3: fields in PDO8

B.1.5 PDO9 - Barometric altitude meter raw data

This PDO carries latest data from MPL3115A2 barometric altitude meter. It is sent by slave whenever new data arrive. Content of the message is in Table B.4, the data are readouts from MPL3115A2's register bank. They are not altered in any way. See MPL3115A2 datasheet [24] for instructions to decode these values. Size of PDO9 is 13B.

#	field	type	comment
0	`altitude`	uint24	registers `0x01`-`0x03`; big endian; abnormal coding
3	`temperature`	uint16	registers `0x04`-`0x05`; big endian; abnormal coding
5	`correction`	uint16	registers `0x14`-`0x15`; big endian; abnormal coding
7	`timestamp`	uint48	

Table B.4: fields in PDO9

B.1.6 PDO10 - OBD-II raw data

This PDO carries latest data read from vehicle's ECU via OBD-II interface. It is sent by slave every 100ms. See Table B.5 for its format.

#	field	type	comment
0	`vehicle speed`	uint8	1LSB = 1km/h
1	`timestamp`	uint48	

Table B.5: fields in PDO10

B.1.7 PDO11, PDO12 - GNSS messages

These two PDOs contain messages received from NV08C-CSM receiver, they are sent by slave. Content is forwarded exactly the way as it was received, adding only 1-byte identifier to the beginning. This identifier allows to distinguish between different messages.

PDO11 forwards messages listed in Table B.6. These are related to position, velocity and time solution that the GNSS receiver provides. PDO12 forwards messages listed in Table B.7, these are outputs from ranging processor that allow to re-implement receiver's navigation processor in microcontroller and to tightly integrate it with inertial navigation system (see section 2.6). Note that PDO12 can, in some cases, overload the Bluetooth channel.

Format of messages listed in Tables B.6 and B.7 is described in BINR protocol specification [17].

41h	Course angle and current speed
52h	Visible satellites
88h	Position, velocity and time vector
64h	DOP and calculated uncertainties

Table B.6: messages forwarded by PDO11 (for their specification see [17])

4Ah	Ionosphere parameters
F5h	Raw data (pseudo-ranges, pseudo-range rates, doppler)
F7h	Extended ephemeris of satellites

Table B.7: messages forwarded in PDO12 (for their specification see [17])

B.1.8 PDO13 - Integrated data

This PDO carries navigation solution of the embedded system. The message contains position, velocity and related uncertainties. Position is expressed in ECI coordinates (Cartesian system), standard deviations are in meters, velocity in meters per second. For its format see Table B.8, the message has 50 bytes.

#	field	type	comment
0	position_x	double	
8	position_y	double	
16	position_z	double	
24	stddev_x	float	
28	stddev_y	float	
32	stddev_z	float	
36	velocity	float	
40	stddev_vel	float	
44	timestamp	uint48	

Table B.8: fields in PDO13

Note that the data are based on vehicle speed and GNSS positioning solution as the integration filter does not exist yet. The map-matching algorithm uses information provided by this PDO.

This page is intentionally left blank.

Appendix C

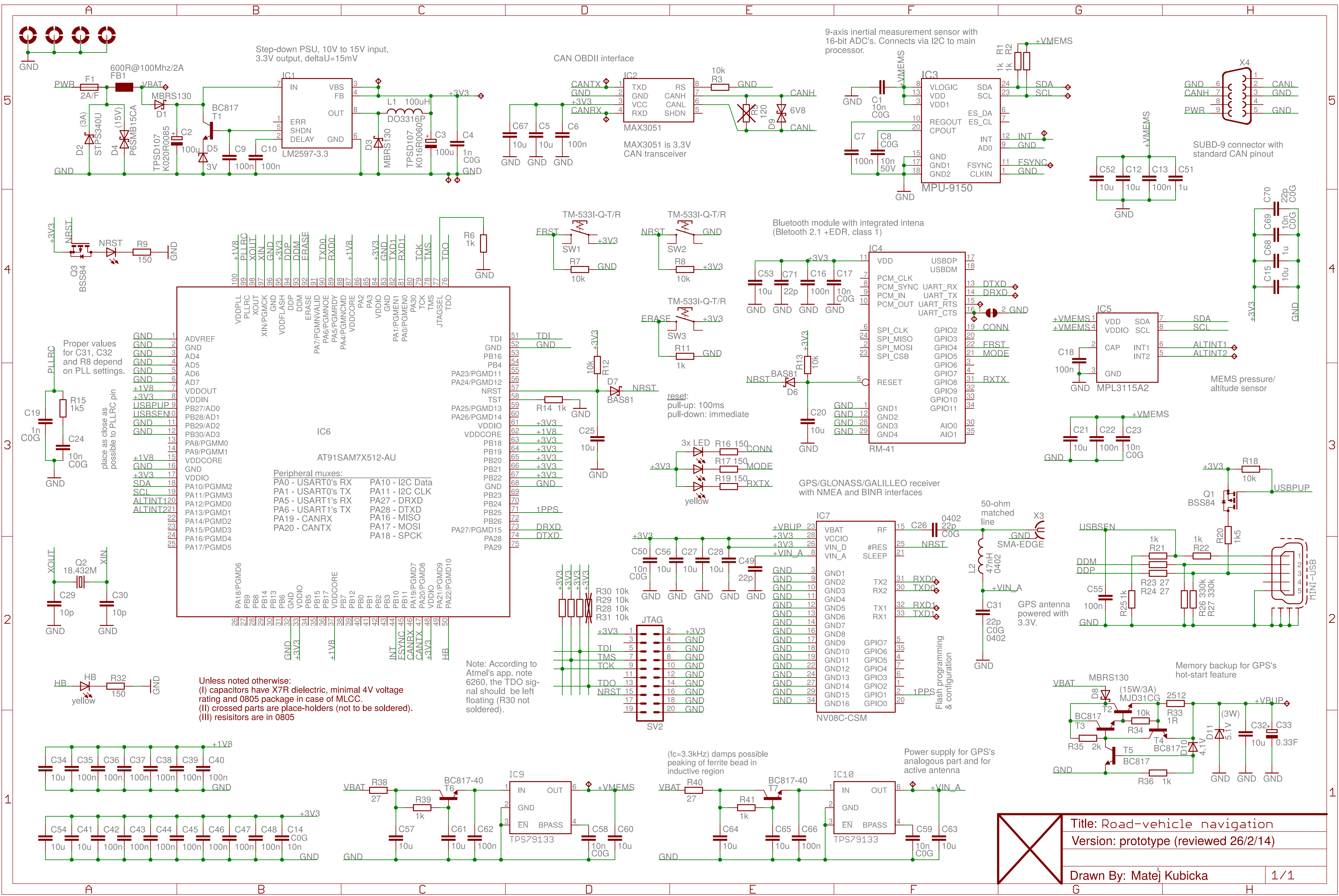
Printed circuit board specifications

Resources related to printed circuit board design are listed here. See following pages for the schematic, detailed view of the PCB copper traces and PCB parts placement. For the bill of materials see Table C.1 below. Note that bill of materials with more details and ordering codes is available on enclosed DVD.

Qty	**Part name**	**Description**
5	HB, LED1, LED2, LED3, NRST	LED diodes, 0805, yellow
1	C33	Backup capacitor, BUC-0.33
24	C6, C7, C9, C10, C13, C16, C18, C22, C35, C36, C37, C38, C39, C40, C42, C43, C44, C45, C46, C47, C48, C55, C62, C66	Capacitor 100n X7R, 0805, 50V
1	C2	Capacitor, 100uF, 85mΩ
1	C3	Capacitor, 100uF, 60mΩ
11	C1, C4, C8, C14, C17, C23, C24, C50, C58, C59, C69	Capacitor 10n, C0G, 0805, 50V
2	C29, C30	Capacitor 10p, C0G, 0805, 50V
23	C5, C11, C12, C15, C20, C21, C25, C27, C28, C32, C34, C41, C52, C53, C54, C56, C57, C60, C61, C63, C64, C65, C67	Capacitor 10u, X7R, 0805, 6.3V
1	C19	Capacitor 1n, C0G, 0805, 50V
2	C51, C68	Capacitor 1u, X7R, 0805, 6.3V
5	C26, C31, C49, C70, C71	Capacitor 22p, C0G, 0402
1	F1	Fuse 2A, fast, 1206
1	R5	Resistor, 120Ω, 0805
11	R3, R7, R8, R12, R13, R18, R28, R29, R30, R31, R34	Resistor, 10k, 1%, 0805
6	R4, R9, R16, R17, R19, R32	Resistor, 150, 1%, 0805
1	R33	Resistor, 1R, 1%, 2512
11	R1, R2, R6, R11, R14, R21, R22, R25, R36, R39, R41	Resistor, 1k, 1%, 0805
2	R15, R20	Resistor, 1k5, 1%, 0805

4	R23, R24, R38, R40	Resistor, 27, 1%, 0805
1	R35	Resistor, 2k, 1%, 0805
2	R26, R27	Resistor, 330k, 1%, 0805
1	D2	Power schottky, 3A
3	D1, D3, D8	Power schottky, 1A
1	D4	Transil, 600W, 15V
1	D9	Transil, 600W, 6.8V
2	D6, D7	Schottky diode
1	D5	Zener diode, 4.1V, 0.6W
1	D10	Zener diode, 3V, 0.6W
1	D11	Zener diode, 5.1V, 3W
1	T2	NPN Transistor, 3W
6	T1, T3, T4, T5, T6, T7	NPN signal transistor
2	Q1,Q3	P-channel MOSFET
1	L1	Power inductor, 100uH, 1.3A
1	L2	Inductor, 0402, 47nH
1	FB1	Ferrite Bead, 600R@100MHz, 2A
1	Q2	Crystal, 18.432Mhz
3	SW1, SW2, SW3	Microswitch, SMD, 50mA
1	IC1	PSU switcher LM2597, 3.3V
1	IC2	CAN transceiver MAX3051
2	IC9, IC10	TP79133, LDO, 3.3V,
1	IC6	Microcontroller AT91SAM7X512
1	IC5	Pressure sensor MPL3115A2
1	IC3	9-axis IMU MPU-9150
1	IC7	GNSS receiver NV08C-CSM
1	IC4	Bluetooth module RN41
1	JTAG	HARTING, 20-pin, male
1	X2	MINI USB-B Conector
1	X1	SUBD9, edge mount
1	X3	SMA Antenna Connector
15	TP3, TP4, TP5, TP6, TP7, TP8, TP9, TP10, TP11, TP12, TP14, TP15, TP16, TP21, TP1	Spring test probes

Table C.1: bill of materials


Step-down PSU, 10V to 15V input, 3.3V output, deltaU=15mV
600R@100Mhz/2A
LM2597-3.3
CAN OBDII interface
MAX3051
MAX3051 is 3.3V CAN transceiver
9-axis inertial measurement sensor with 16-bit ADC's. Connects via I2C to main processor.
MPU-9150
SUBD-9 connector with standard CAN pinout
Bluetooth module with integrated intena (Bletooth 2.1 +EDR, class 1)
RM-41
MEMS pressure/altitude sensor
MPL3115A2
Proper values for C31, C32 and R8 depend on PLL settings.
place as close as possible to PLLRC pin
IC6
AT91SAM7X512-AU
Peripheral muxes:
PA0 - USART0's RX
PA1 - USART0's TX
PA5 - USART1's RX
PA6 - USART1's TX
PA19 - CANRX
PA20 - CANTX
PA10 - I2C Data
PA11 - I2C CLK
PA27 - DRXD
PA28 - DTXD
PA16 - MISO
PA17 - MOSI
PA18 - SPCK
reset:
pull-up: 100ms
pull-down: immediate
GPS/GLONASS/GALILLEO receiver with NMEA and BINR interfaces
NV08C-CSM
50-ohm matched line
SMA-EDGE
GPS antenna powered with 3.3V.
Flash programming & configuration
MINI-USB
Note: According to Atmel's app. note 6260, the TDO signal should be left floating (R30 not soldered).
JTAG
Unless noted otherwise:
(I) capacitors have X7R dielectric, minimal 4V voltage rating and 0805 package in case of MLCC.
(II) crossed parts are place-holders (not to be soldered).
(III) resisitors are in 0805
(fc=3.3kHz) damps possible peaking of ferrite bead in inductive region
Power supply for GPS's analogous part and for active antenna
TPS79133
Memory backup for GPS's hot-start feature
Title: Road-vehicle navigation
Version: prototype (reviewed 26/2/14)
Drawn By: Matej Kubicka
1/1

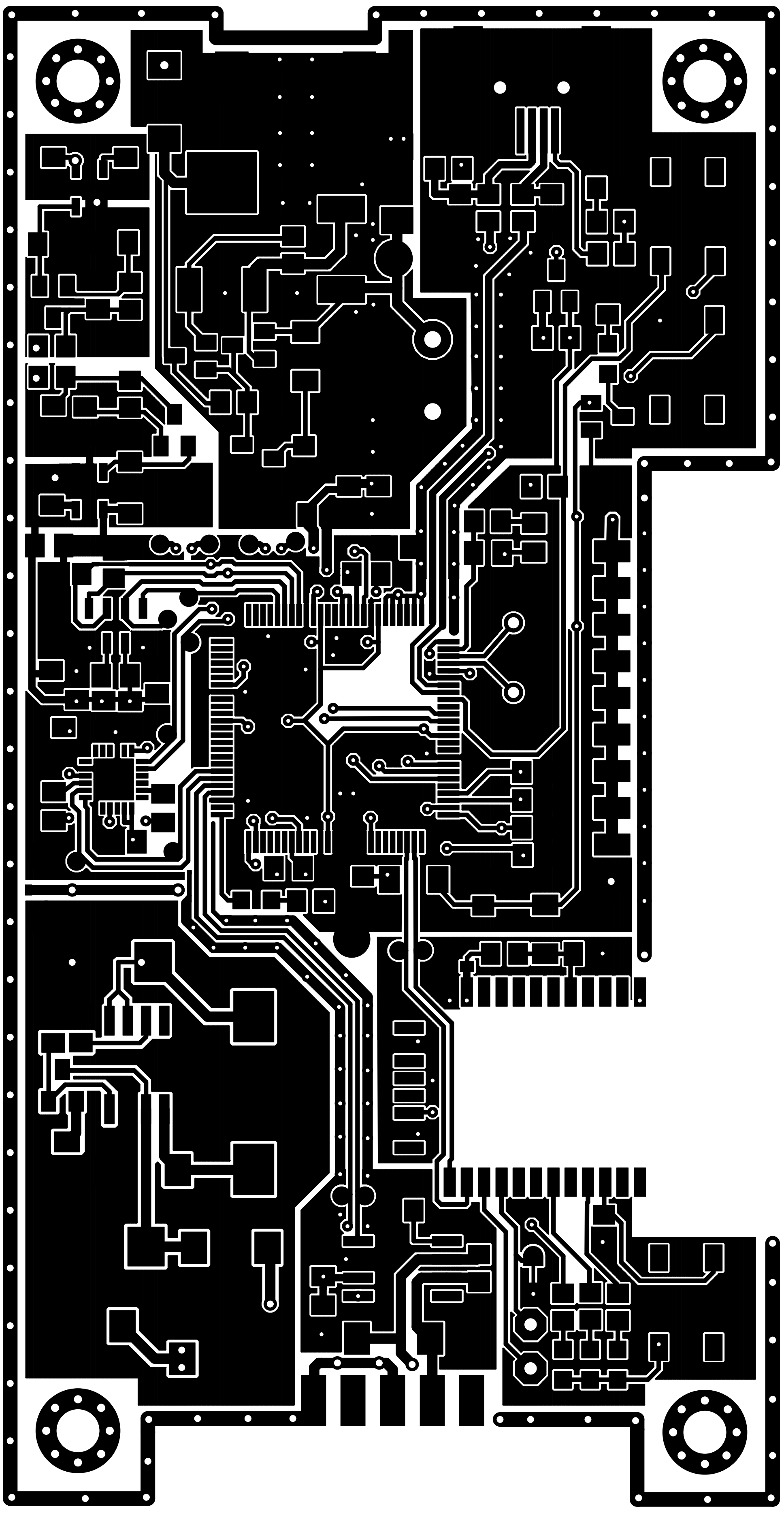


Figure C.1: PCB copper trace, top side

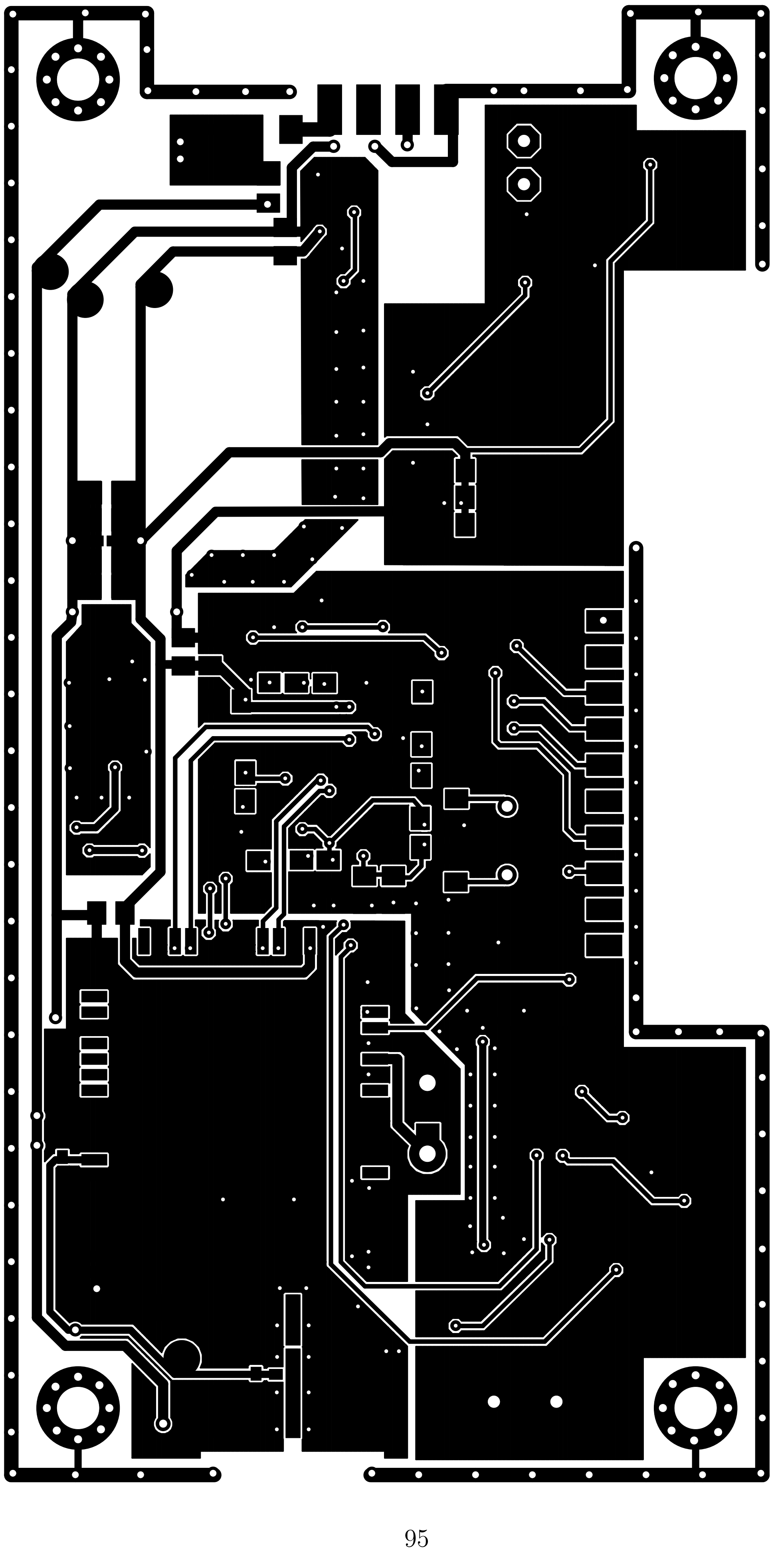


Figure C.2: PCB copper trace, bottom side

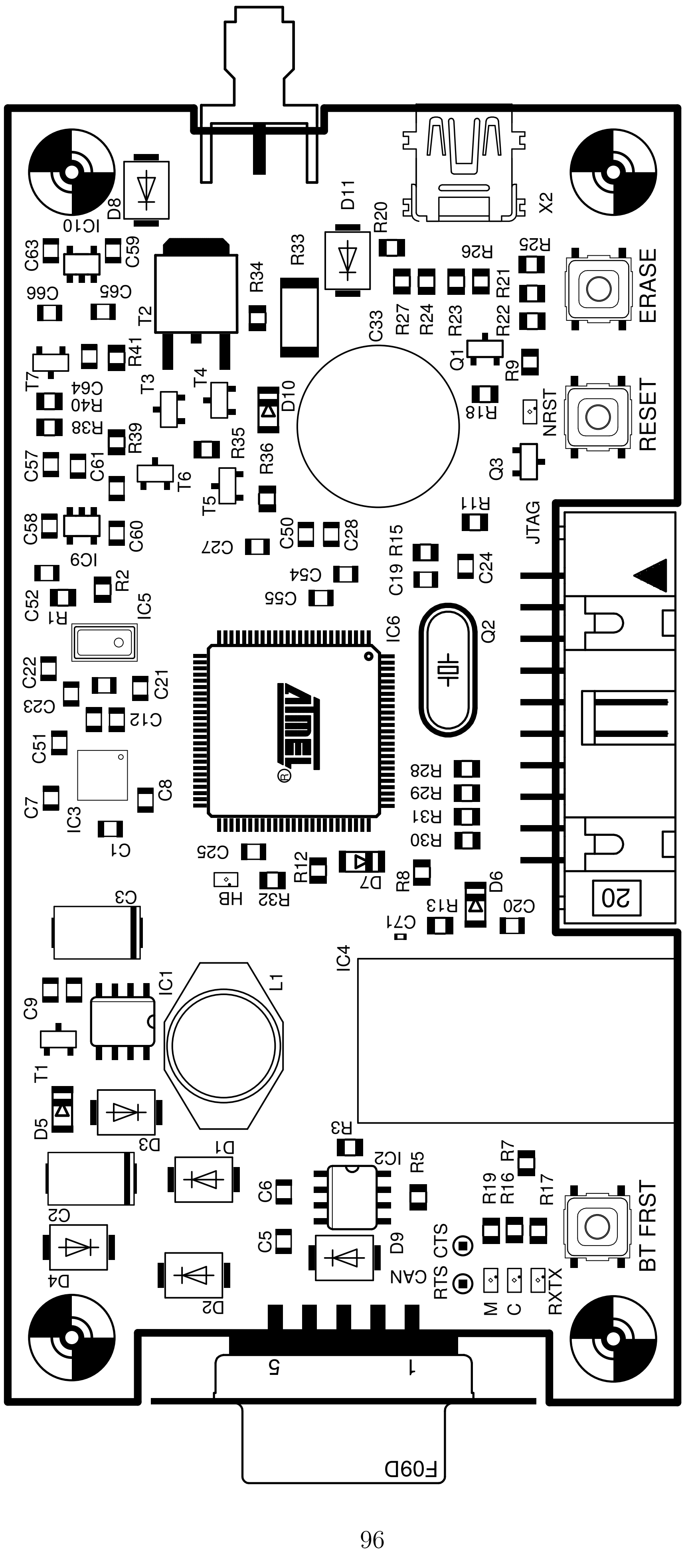


Figure C.3: PCB parts placement, top side

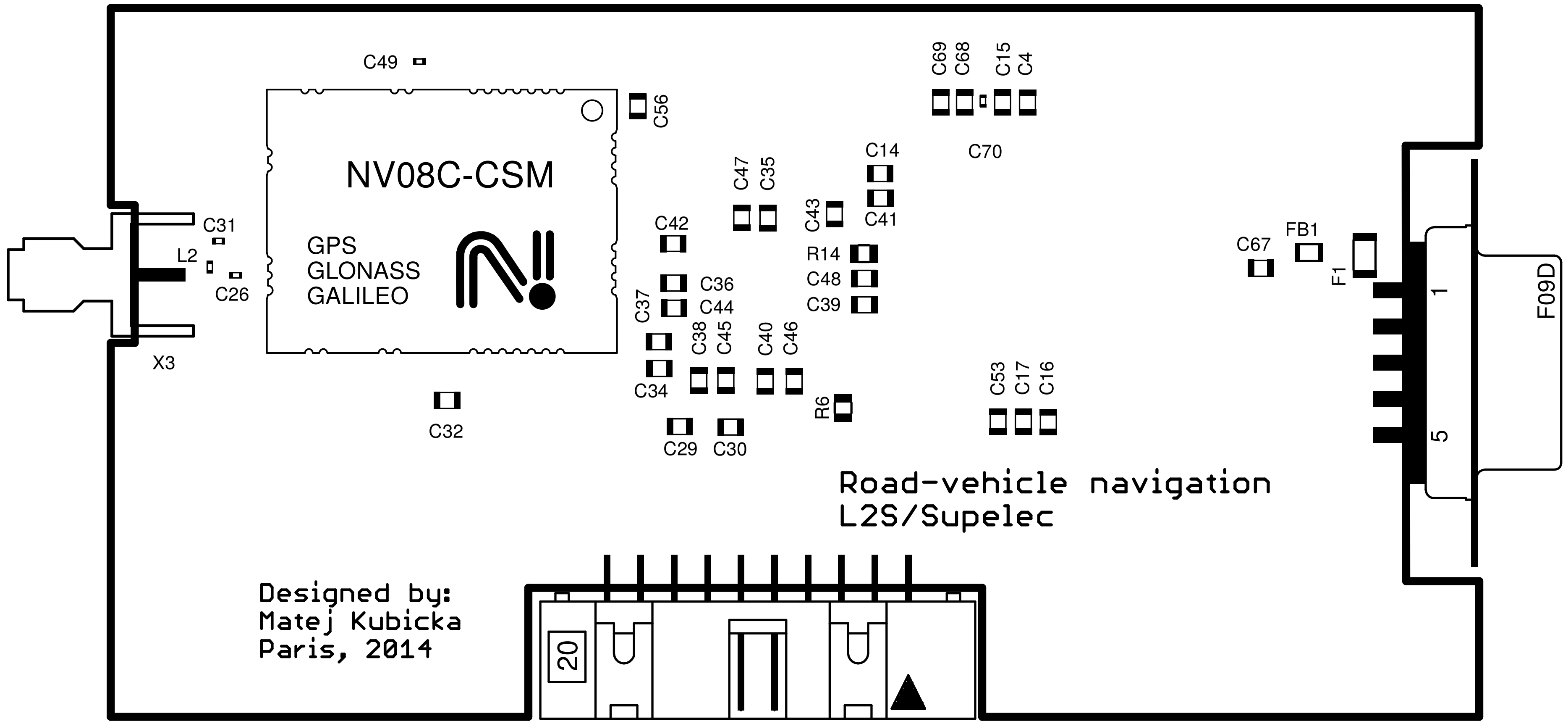


Figure C.4: PCB parts placement, bottom side

This page is intentionally left blank.

Appendix D

List of acronyms and abbreviations

1PPS	One pulse per second
ADC	Analog to digital converter
API	Application interface
ARM	Advanced RISC Machines
CAN	Controller area network
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CDMA	Code-division multiple-access
CNRS	Centre National de la Recherche Scientifique
CPU	Central processing unit
CRC	Cyclic redundancy check
CSV	Comma separated values
DBGU	Debug unit
DGPS	Differential GPS
DMA	Direct memory access
DOP	Dillution of precision
DSP	Digital signal processor
ECEF	Earth-centered Earth-fixed
ECI	Earth-centered inertial
ECU	Engine control unit
EDF	Earliest deadline first
EEPROM	Electrically erasable programmable read-only memory
EGM96	Earth gravitational model 1996
EGNOS	European geostationary navigation overlay service
EMI	Electromagnetic interference
FDMA	Frequency-division multiple-access
FPU	Floating point unit
GAGAN	GPS Aided GEO Augmented Navigation
GDOP	Geometric dilution of precision
GLONASS	Globalnaya Navigatsionnaya Sputnikovaya Sistema
GNSS	Global navigation satellite system
GNU	GNU Not Unix
GPR	General purpose registers

GPS	Global Positioning System
IC	Integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial measurement unit
INS	Inertial navigation system
IRNSS	Indian Regional Navigation Satellite System
ISO	International Standardization Organization
JTAG	Joint Test Action Group
LDO	Low drop-out
LED	Light emitting diode
LGA	Land grid array
LORAN	Long range navigation
LSB	Lowest significant bit
MCLK	Master clock
MCM	Multiple chip module
MEMS	Micro electro-mechanical systems
MHT	Multiple hypotheses technique
MLCC	Multiple layer ceramic capacitor
MOSFET	Metal-oxide semiconductor field-effect transistor
NMEA	National Marine Electronics Association
OBD-II	On-board diagnostics II
OSM	Open street maps
PCB	Printed circuit board
PDC	Peripheral data channel
PDO	Process data object
PID	Parameter ID
PIO	Parallel input/output
PIT	Periodic interval timer
PLL	Phase locked loop
PPS	Precise positioning system
PRN	Pseudo-random number
PSU	Power supply unit
RAIM	Receiver autonomous integrity monitoring
RAM	Random access memory
RMS	Root mean square
RTCM	Radio Technical Commission for Maritime Services
RTOS	Real-time operating system
SBAS	Satellite-based augmentation system
SCL	Serial clock
SDA	Serial data
SDO	Service data object
SMD	Surface mount devices
SMT	Surface mount technology
SiSNET	Signal-In-Space available over the Internet
SPP	Serial port profile

SQL	Select query language
SRAM	Static random access memory
TCP/IP	Transmission Control Protocol/Internet Protocol
TDMA	Time-division multiple access
TWI	Two-wire Interface
UART	Universal asynchronous receiver/transceiver
USART	Universal synchronous/asynchronous receiver/transceiver
USB	Universal serial bus
UTC	Universal time coordinated
VLSI	Very large scale integration
WAAS	Wide Area Augmentation System
XML	Extensible markup language
ZVU	Zero-velocity updates

Bibliography

[1] Kaplan, E. D., Hegarty C. J. (2005). “Understanding GPS: Principles and Applications (second edition)”. Artech House. ISBN 1-58053-894-0.

[2] Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., Huang, Y. (2009). “Map-Matching for Low-Sampling-Rate GPS Trajectories”, ACM GIS ’09, November 4-6, 2009. Seattle, WA, USA.

[3] Quddus, M.A. (2006). “Current map-matching algorithms for transport applications: State-of-the art and future research directions”, Transportation Research Part C 15, p. 312-328.

[4] Artese, G., Trecroci, A. (2008). “Calibration of a low cost MEMS INS sensor for an integrated navigation system. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences”. Vol. XXXVII. Part B5.

[5] Groves P.D. (2008), “Principles of GNSS, inertial, and multi-sensor integrated navigation systems”. Artech House, ISBN-13: 978-1-58053-255-6.

[6] Quddus, M.A., Noland, R.B., Ochieng, W.Y. (2006). “A high accuracy fuzzy logic-based map-matching algorithm for road transport”. Journal of Intelligent Transportation Systems: Technology, Planning, and Operations 10 (3), 103-115.

[7] Pyo, J., Shin, D., Sung, T. (2001). “Development of a map-matching method using the multiple hypothesis technique”. IEEE Proceedings on Intelligent Transportation Systems, p. 23-27.

[8] D. B. Reid (1979) “An algorithm for tracking multiple targets”, IEEE Trans. Automatic Control, vol. AC-24, pp843-854.

[9] Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., Nordlund, P. (2002). “Particle filters for positioning, navigation, and tracking”. IEEE Transactions on Signal Processing 50, 425-435.

[10] Schuessler, N., Axhausen K. W. (2009). “Map-matching of GPS traces on high-resolution navigation networks using the Multiple Hypothesis Technique (MHT)”, Working paper 568, Transport and Spatial Planning.

[11] Marchal, F., Hackney, J., Axhausen, K.W. (2005). “Efficient map matching of large global positioning system data sets: Tests on speed monitoring experiment in Zurich”. Transportation Research Record 1935, 93-100.

[12] Greenfeld, J.S. (2002). “Matching GPS observations to locations on a digital map”. In proceedings of the 81st Annual Meeting of the Transportation Research Board, January, Washington D.C.

[13] White, C.E., Bernstein, D., Kornhauser, A.L. (2000). “Some map-matching algorithms for personal navigation assistants”. Transportation Research Part C 8, 91-108.

[14] Ochieng, W.Y., Quddus, M.A., Noland, R.B. (2004). Map-matching in complex urban road networks. Brazilian Journal of Cartography (Revista Brasileira de Cartografia) 55 (2), 1-18.

[15] NVS Technologies (2013), NV08C-CSM datasheet [Online]. Available (17 July 2014): http://www.nvs-gnss.com/products/receivers/item/download/58.html

[16] ISO standard 15031-5:2006, “Road vehicles - Communication between vehicle and external equipment for emissions-related diagnostics, Part 5: Emissions-related diagnostic services”.

[17] NVS Technologies (2013), “BINR Protocol Specification” [Online]. Available (17 July 2014): http://www.nvs-gnss.com/support/documentation/item/download/39.html

[18] Atmel Corporation (2007). “AT91SAM7X and AT91SAM7XC Microcontroller Series Schematic Check List”. Application Note [Online]. Available (17 July 2014): http://www.atmel.com/Images/doc6260.pdf

[19] Texas Instruments (2013). “USB 2.0 Board Design and Layout Guidelines”. Application report [Online]. Available (17 July 2014): http://www.ti.com/lit/an/spraar7b/spraar7b.pdf

[20] Atmel Corporation (2011). “AT91SAM7X512/526/128 product datasheet” [Online]. Available (17 July 2014): www.atmel.com/Images/Atmel_32-bit-ARM7TDMI-Flash-Microcontroller_SAM7X512-256-128_Datasheet.pdf

[21] ISO standard 11898-1:2003, “Road vehicles - Controller area network (CAN), Part 1: Data link layer and physical signalling”.

[22] National Maritime Electronics Association (2002). “NMEA 0183 - Standard For Interfacing Marine Electronic Devices”, Version 3.01.

[23] InvenSense, Inc. (2013). “MPU-9150 Product Specification” [Online]. Available (17 July 2014): www.invensense.com/mems/gyro/documents/PS-MPU-9150A-00v4_3.pdf

[24] Freescale Semiconductor (2013). “Xtrinsic MPL3115A2 I^{2}C Precision Altimeter”. Data Sheet: Technical Data [Online]. Available (17 July 2014): http://www.freescale.com/ webapp/sps/site/prod_summary.jsp?code=MPL3115A2

[25] Mapsforge - free mapping and navigation tools. Project home [Online]. Available (17 July 2014): http://code.google.com/p/mapsforge/

[26] Kalman, R. E. (1960). “A New Approach to Linear Filtering and Prediction Problems”x. Journal of Basic Engineering 82 (1): 3545. doi:10.1115/1.3662552