COM SYS Communication & Distributed Systems

RWTH AACHEN UNIVERSITY

Opportunistic Deployment Support for Wireless Sensor Networks

Master Thesis

Paul Anthony Smith

RWTH Aachen University, Germany

Chair for Communication and Distributed Systems

Advisors:

Dipl. Inf.	Jó Ágila Bitsch Link
Dr. rer. nat.	Tómas M. Fernandez-Steeger
Prof. Dr.	Klaus Wehrle
Prof. Dr.	Rafig Azzam

Registration date: 2013-07-02
Submission Date: 2013-09-27

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Aachen, den 27. September 2013

Abstract

Wireless Sensor Networks are versatile tools to monitor the environment. However, deployments of such networks are challenging even for experts. With the emergence of ubiquitous general purpose mobile devices, we have a platform available which has the potential to greatly reduce these challenges.

In this work, we present a smartphone based system that supports WSN deployments by keeping track of sensor locations and providing debugging tools, such as real-time display of the network topology and sensor values. Support for opportunistic communication allows for a distributed deployment methodology, where data is immediately synchronized, even in the absence of supporting infrastructure. This allows the automatic boot-strapping of domain expert evaluation and assessment tools already during initial deployment. We validated our design successfully during a deployment in Thórsmörk, Iceland.

Abstract

Kabellose Sensornetze sind ein vielseitiges Werkzeug zur Umweltüberwachung. Selbst für Experten kann das Ausbringen dieser Netze jedoch eine Herausforderung sein. Die Verbreitung universell einsetzbarer Mobilgeräte stellt uns hier eine Plattform zur Verfügung, mit dem Potential diese Probleme deutlich zu reduzieren.

In dieser Arbeit wird ein System vorgestellt, welches mit Hilfe von Smartphones die Ausbringung von kabellosen Sensornetzen vereinfacht, zum einen durch die Aufzeichnung der Gerätestandorte sowie durch die Echtzeitvisualisierung der Netzwerktopologie und der Messdaten. Die Verwendung opportunistischer Kommunikationsparadigmen erlaubt dabei eine verteilte Ausbringungsstrategie, in welcher Daten sofort zwischen den Helfern synchronisiert werden, auch ohne den Zugriff auf unterstüzende Kommunikationsinfrastruktur. Weiterhin wird hierdurch die Initialisierung von spezialisierten Evaluations- und Bewertungswerkzeugen schon in der ersten Phase ermöglicht. Das System wurde im Rahmen eines Aufbaus in Thórsmörk, Island erfolgreich getestet.

Acknowledgments

I would like to thank Jó Ágila Bitsch Link for supervising this thesis. It was a great experience, and he provided me with inspirations and support throughout this work. I thank Prof. Dr. Klaus Wehrle and Prof. Dr. Rafig Azzam for giving me the chance to write this thesis. I also thank them for providing the hardware I needed during testing and evaluation. Last but not least, I thank my family, and girlfriend for their invaluable support.

Contents

1 Introduction **1**

1.1 Structure of this Thesis 2

2 Background **3**

2.1 Wireless Sensor Networks 4

2.1.1 Sensor Nodes 4

2.1.2 Routing 6

2.2 WSN Use Cases 8

2.2.1 SLEWS 8

2.2.2 PermaSense 9

2.3 Wireless Networking Technologies 9

2.3.1 Bluetooth (IEEE 802.15.1) 10

2.3.2 Wi-Fi (IEEE 802.11) 10

2.3.3 Cellular Networks 11

2.3.4 IEEE 802.15.4 12

3 Related Work **13**

3.1 Surveying 14

3.1.1 Geopaparazzi 14

3.1.2 OSMTracker 15

3.1.3 iButton Assist 16

3.1.4 idNotebook 16

3.1.5 SensorTune 16

3.1.6 GSN Project 17

3.2 Map-Making 17

3.2.1 Balloon and Kite Mapping 18

3.2.2 MapMill 19

3.2.3 MapKnitter 20

3.2.4 MapCruncher 20

4 Design **23**

4.1 General Requirements . . . 23

4.1.1 Opportunistic Deployment Support . . . 23

4.1.1.1 Mobile Devices . . . 25

4.1.2 Use Cases . . . 25

4.1.2.1 Surveying . . . 25

4.1.2.2 Deployment and Maintenance . . . 26

4.2 File System Layout . . . 27

4.3 Communication . . . 28

4.3.1 Discovery . . . 28

4.3.2 Ranges . . . 28

4.3.3 Bootstrapping . . . 30

4.3.4 Synchronization . . . 30

4.4 Surveying . . . 31

4.4.1 Online/Offline Maps . . . 31

4.4.2 Annotations . . . 31

4.4.3 Mapping . . . 32

4.4.3.1 Ballon/Kite Aerial Image Retrieval . . . 32

4.4.3.2 Stitching . . . 33

4.4.3.3 Distribution of Map Packs . . . 33

4.5 Deployment and Maintenance . . . 34

4.5.1 Gathering of Node Data . . . 34

4.5.2 Visualization of Network Status . . . 34

4.5.3 Finding Nodes . . . 35

5 Implementation **37**

5.1 System Overview . . . 37

5.2 Open Source Libraries . . . 39

5.2.1 Protocol Buffers . . . 39

5.2.2 OSMdroid and OSMdroid bonuspack . . . 39

5.2.3 ZXing (“Zebra Crossing”) . . . 40

5.2.4 Otto Event Bus . . . 40

5.3 NodeMap . . . 41

5.4 The DAISY Library . . . 43
5.4.1 Osmdroid Layers . . . 43
5.4.2 Daisy Protocol . . . 43
5.5 The UVWXY Library . . . 44
5.5.1 Abstract Connection Layer . . . 44
5.6 XBee Driver . . . 44
5.6.1 Parsing WSN Messages . . . 47
5.7 Summary . . . 48

6 Evaluation **49**
6.1 Pre-Deployment: Range Measurements . . . 49
6.1.1 Initial Survey . . . 49
6.1.2 Local conditions . . . 50
6.1.3 Communication . . . 51
6.1.4 Data Gathering . . . 51
6.2 Survey on Requirements . . . 51
6.3 Demo Deployment at ExtremeCom 2013 . . . 52
6.3.1 Data Gathering/Monitoring . . . 54
6.3.2 Aerial Maps . . . 55
6.4 Survey Results of our System . . . 55
6.5 Discussion . . . 62
6.6 Summary . . . 67

7 Conclusion **69**
7.1 Future Work . . . 70

Bibliography **73**

A Appendix **77**

1

Introduction

Deploying a WSN is a cumbersome task even as an expert in the area. Outside of testbeds [21], the question if the network was deployed as planned remains open until the end of the deployment, when we compare the obtained measurements with expected theoretical data. In this work, we present a system that assists us during surveying, deployment and maintenance of a WSN. We provide a smartphone application with which we keep track of sensor nodes, their identifiers, location and their view of the network during the deployment. With our system, we can collaboratively create and explore the network topology, and exchange gathered information about the WSN across the deployment site opportunistically among participants.

Long-term environmental monitoring is an important part of ongoing research. The collection of sensor data over longer periods of time enables us to study and understand our environment. A Wireless Sensor Network (WSN) is an improvement of existing data collection infrastructure. It facilitates the application of a scalable and autonomous sensing network that requires low maintenance. The collected data is communicated to a single station that relays it to a remote server, eliminating the need to manually recover memory cards or hard disks from every sensor. The result is a distributed sensing network that measures the environment continuously during its deployment. The application areas for senor networks already include deployments on, for example mountains [5], glaciers [29], in volcanoes [49], and in oceans [48].

The problems that arise during sensor network deployment are hidden in the details of the process. Experiences, such as a missing ice-ax to anchor sensor nodes, or a missing Phillips screwdriver [4], are examples of how easy the installation of a WSN can be delayed. Also, when developing a WSN, bugs are more likely to arise from hardware failures than in traditional software [45]. The early detection of error sources, and a rapid approach to eliminate them, is key to a successful deployment [18]. Despite, the advantages of WSNs we have to concede the difficulty of setting up such a network. A WSN is developed in a laboratory where we have the necessary tools to debug and analyze errors. The transfer of the development lab to a remote

environment is not feasible. Thus, we have to accept a limited set of tools and devices that are at our disposal during the deployment of a WSN. Additionally, if the final system is tested under the wrong assumptions [28], it is likely to fail in the field.

Our goal during WSN deployment is to be able to respond to unpredicted wireless behavior and other obstacles, to reduce the time needed to set up the system. While passing through the deployment area, we monitor the WSN traffic of active nodes, and maintain an overview on where nodes are located and how they participate in the network. This way we can keep an overview of the network, identify nodes that are not responding and act accordingly.

We developed tools that simplify the task of recording a node's location and its integration into the WSN. In a simple step we obtain the id, location and orientation of a deployed node. Additionally, we can append this information with notes, images, and an estimation of the node's height from the ground. From the sniffed WSN traffic we can establish if the node is sending sensor data to the base station, and what the sensor are reporting. Thus, we can see if the node is working correctly. The collected data is automatically distributed using available radio technologies, i.e., Bluetooth, Wi-Fi or a GSM network. This way, the information is available to all participants, enabling a collaborative methodology for deploying WSNs.

1.1 Structure of this Thesis

This thesis is structured as follows. In Chapter 2, we introduce WSNs, and explain the design of a sensor node using an X-SLEWS sensor node. This is followed by two sensor network use cases. In Chapter 3, we describe existing systems that are related to our work. Then, we establish the requirements of WSN deployments, that motivate our work, and thus how we designed our system, in Chapter 4. In Chapter 5, we present details on how we implemented our application, protocols and how we used and modified existing open source libraries. Then, in Chapter 6, we present the evaluation of our system. For this, we tested our work at the ExtremeCom 2013, where we collected data during a demo deployment and surveyed the participants regarding the underlying requirements and the quality of our work.

2

Background

With the miniaturization of computer components it is possible to build tiny devices to measure and sense properties of the environment. A collection of these devices can be distributed across the environment and connected to gather information and draw conclusions, to either monitor given factors, e.g. air flow, temperature, wildlife, or also to improve an existing system. Intelligent buildings make use of this information to better control the humidity or temperature of rooms to reduce energy waste or improve the well-being of residents. To enable communication between the participating components of such a network, we need not only wired communication but also wireless communication systems to bridge inaccessible terrain, or to avoid invading an environment with infrastructure such as cables between devices. Depending on the environment and what we want to observe we require different architectures of devices and sensors to sustain in the area for a longest possible time span and provide the required quality of measured data. The location and thus distances between nodes also influence the choice of which wireless technology is used to communicate data between devices.

Despite the advances in technology that enable us to build wireless sensor networks, the process of setting up such a network is a tedious task. Many components, especially radio communication, behave differently in the field making it hard to debug errors in the system. To the end-user a wireless sensor node is a black box that has to deliver the desired data to computer located at a different location. Errors are concealed along the path through the sensor network to the Internet, and thus to the final destination, where end-users have limited options to fix them.

In the following sections we will introduce the background of our work, i.e. components and examples of sensor networks, as well as the radio technologies used in this subject area.

2.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) is a network embedded into an environment. It consists of small devices that autonomously establish a communication network between them to exchange data, without significant possibilities of human interaction. The roles of the participating devices in the network can be classified into three types: *Sources* gather data and communicate it to others, *Sinks* receive data from the network and may function as a gateway to other networks or store an entire data set, and *Actuators* that operate a controlling mechanism depending on measured or received data from the network. The devices for these roles commonly share a common architecture, differing in the connected sensors, radio modules or controlling mechanisms. Each role is then realized by the installed software configuration on each device, creating a network of nodes.

The task of a sensor network is to autonomously create a network to communicate data between arbitrary end points in the network, or to forward data over a network bridge to a remote site. It needs to provide self organization, i.e. no manual setup of communication links, and self healing capabilities against node failures, to provide the functionality of the network for the longest possible timespan.

2.1.1 Sensor Nodes

A sensor node is a computing device comparable to a router. The components it shares is a main controller, memory, a power supply, and at least one communication device. Additionally a node contains sensors and actuators to measure and control the environment. A variety of sensors exist that can be added to a sensor node, e.g. accelerometer, barometer, hygrometer, inclinometer, magnetic field sensor or a temperature sensor. Depending on the node's architecture these can be added to empty or stackable sockets to the main board. As a sensor node is designed for interfacing with other nodes and not human interaction, we do not require the presence of a display, a keyboard, or other input/output interfaces designed for human interaction. Typically, the only externally available connections are sockets to attach a radio antenna, the power supply, depending on the application a serial port for debugging and LEDs to indicate the state of the device, see Figure 2.2. The nodes have to be sealed against the environment, e.g. enclosed in a water proof box, see Figure 2.1 and 2.3.

The base station usually contains more radio equipment to transfer data to an off-site location, see Figure 2.4. This can be done with a physical connection, a GSM/3G module [16], Wi-Fi or via a satellite link [3]. During periods without outside connectivity it must provide sufficient data storage for gathered measurements to avoid data loss. A power supply for the base station requires to have a greater capacity or energy harvesting capabilities, as this node has greater power consumption due to multiple radio antennas and greater work load. This is done with solar panels, wind energy, or fuel cells.

Figure 2.1 An X-SLEWS Sensor Node on the left connected to a power supply on the right. The QR code on the node is scanned during the deployment with our system.

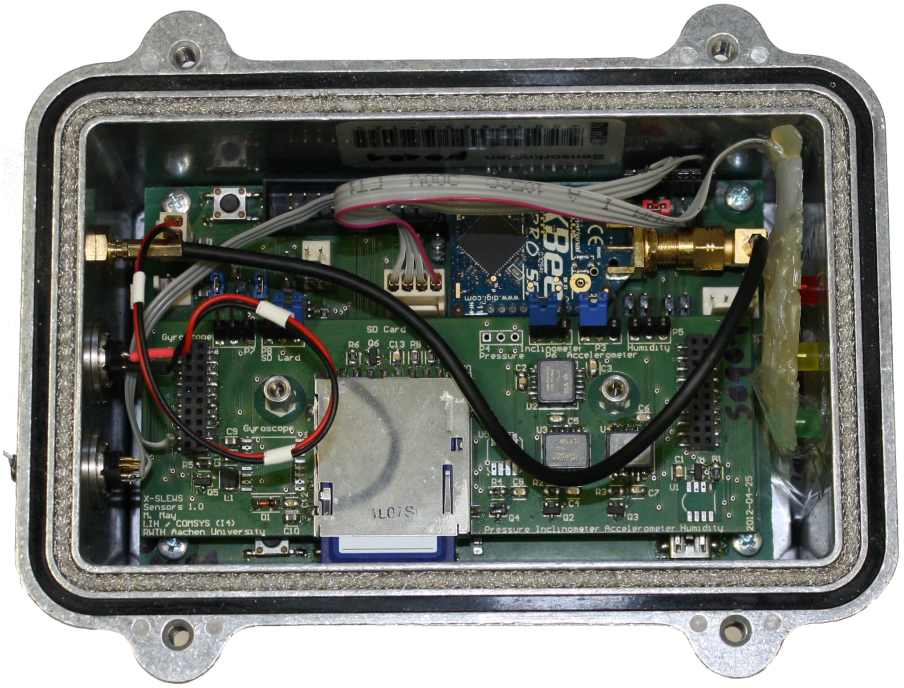

Figure 2.2 The inside of an X-SLEWS sensor node and its power supply. The connections leaving the encasing of the node from top to bottom are used to attach an antenna, the power supply and a serial port for debugging. The memory card is located int the lower middle of the board, with two sensors solder next to it on the right hand side: an inclinometer and an accelerometer. The radio module is located in the top right corner in blue.

Figure 2.3 The power supply of a sensor node. A rechargeable battery is wrapped in bubble wrap to keep it in place and isolated against the metal casing to keep it warm. Robustness is an important requirement in harsh environments. Also, the performance of batteries degrade in cold environments.

2.1.2 Routing

Routing in traditional wired networks is simpler than in wireless sensor networks. The topology of wired networks is static and requires fewer routing updates to be negotiated by connected nodes. By design, protocols do not keep track of mobile nodes, and thus require less network overhead. In wireless sensor networks nodes can be mobile and, depending on the routing protocol, require frequent updates of routing tables, causing traffic overhead on the network. Other approaches only request routing tables when they need to transmit data to a node, which leads to higher latencies when connecting to a node for the first time after the route has changed. Also, temperature has significant effects on the performance of radio transceivers [9]. Harsh environmental conditions changing over days or hours can cause delays, or lead to partitioning of the network, requiring changes in routing decisions.

The communication subsystem often represents the main power drain of a sensor node [3] Radio transceivers require the same energy for receiving as for sending data, making it feasible to deactivate the radio module whenever possible with the goal to increase the lifetime of each node and thus the lifetime of the sensor network. Depending on this, routing protocols have to cope with intermittent connectivity of nodes along end-to-end paths, require time synchronization, as well as frequent delays in the delivery of messages [33]

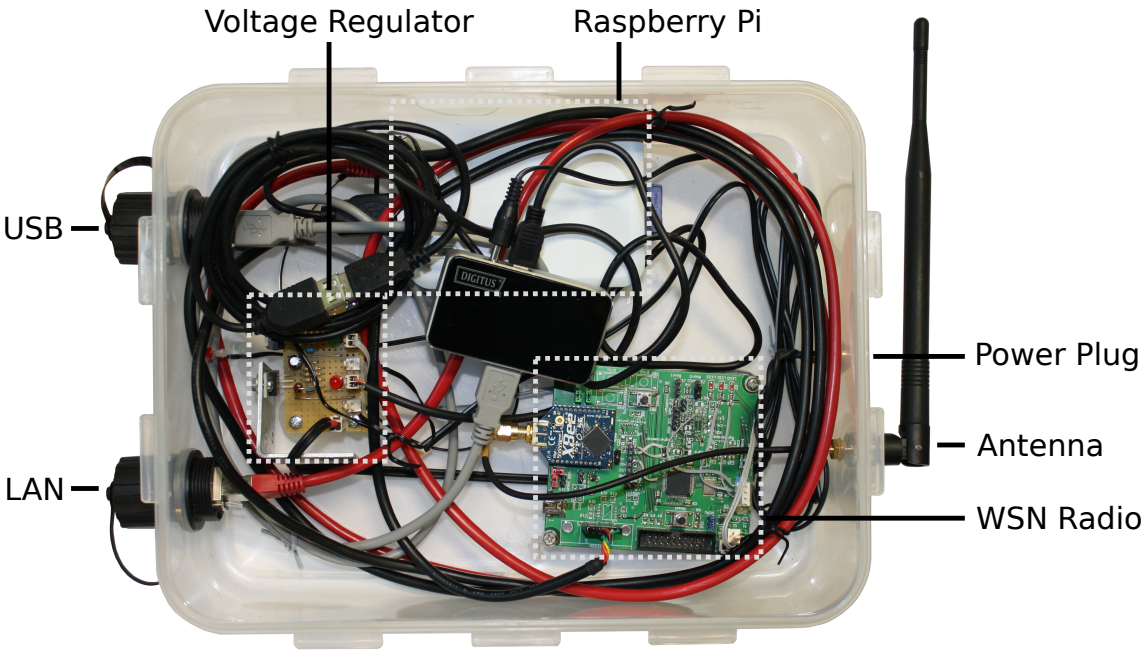

Figure 2.4 The base station of the Wireless Sensor Network. The node for communicating with the WSN is located in the bottom right corner, with the WSN antenna exiting on the right hand side of the container. The power connector is located above the antenna socket in the middle of the right hand side of the container. Physical connections to a Raspberry Pi (white box, top middle) are provided through a weather sealed network socket on the bottom left side of the case with weather sealed USB connection located above. The Raspberry Pi provides debugging and GSM communication functionality to upload collected sensor data to a remote server. The power socket is connected to the voltage regulator located on the left between the USB and LAN cables connected to the external connectors. The Raspberry Pi as well as the WSN node obtain their required power from this regulator.

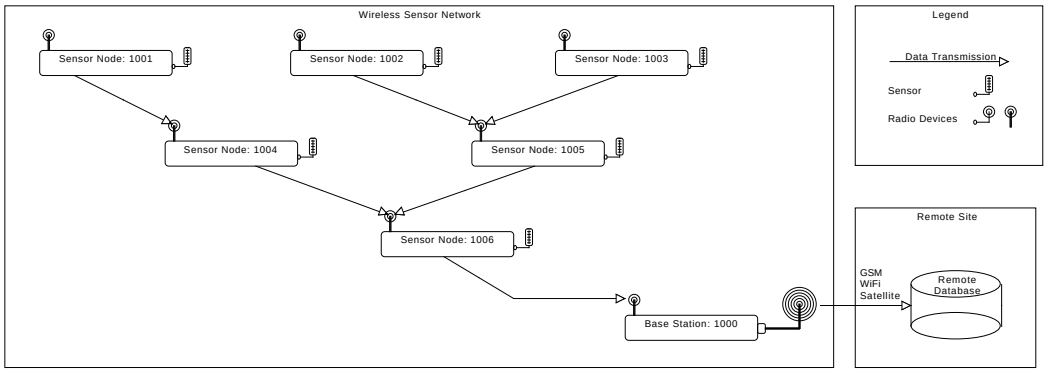

Figure 2.5 Routing of packets in a Wireless Sensor Network. Nodes forward and store packets of their children from top to bottom to the base station. From there the collected data is transfered to an off-site server for analysis and visualization. Depending on the depth of the tree, data has to be routed along multiple hops to reach the root of the tree.

One of the distributed routing techniques are collection trees, see Figure 2.5. The nodes establish a routing tree to transmit messages along its branches to a base station, which has additional radio equipment to send the collected data to a remote server. During the lifetime of the network nodes have to find direct, or multi-hop paths to the root of the tree. The topology of the tree can change over time as intermediary nodes can be temporarily unreachable or failed. Thus, the location of a node has an effect on the topology of the network. This has to be considered during the deployment of a sensor network to avoid partitioning, and makes the deployment of these networks a cumbersome task. Usually, in the setup phase you have to check with the base station or, depending on the scenario, a remote server which node's sensor data is received. If this is not the case the location of the node has to be changed, or further intermediary nodes have to be placed to create further routs for the unconnected nodes in the network.

2.2 WSN Use Cases

There are a range of existing monitoring system utilizing Wireless Sensor Networks. In the following we will present two projects. The first, namely "A Sensor-based Landslide Early Warning System" (SLEWS) aims at a systematical development of a prototyping alarm- and early warning system for different types of landslides utilizing ad hoc wireless sensor networks, developed by Fernandez-Steeger et al. [16]. The second, PermaSense, focuses on wireless sensing systems customized for long-term autonomous operation in high mountain environments.

2.2.1 SLEWS

The Sensor-based Landslide Early Warning System (SLEWS) aims at providing a flexible and extensible system to perform data gathering, evaluation, interpretation and data visualization of geological changes in landslide areas [16]. This enables

the study of landslides, i.e. tracking of surface deformations and movement of rocks in the affected area. Landslides present a danger as they can destroy and harm infrastructure and humans in the vicinity. Due to deforestation and urban development, an increasing number of buildings and inhabitants advance into affected regions. To predict or even avoid incidents it is a prerequisite to study changes in the environment, and thus to be able to develop mitigation strategies.

In SLEWS, sensor nodes are equipped with tilt sensors and accelerometers to observe the orientation change of each sensor node over time. Additionally, an extensometer can be attached to a node to measure the opening and closing of cracks and fissures in rocks [16]. A multitude of nodes is then distributed across the area of observation, spanning up a wireless sensor network for autonomous data gathering, using a gateway node. The wireless design of the system makes it feasible to be used compared to existing monitoring systems, that only provide one or two measuring devices. Using multiple nodes across the observation area leverages the possibility to isolate sensor errors, or confirm changes in the monitored values.

2.2.2 PermaSense

The PermaSense project provides long-term high-quality sensing in harsh environments [5]. They measure the influence of climate change on permafrost as well as the stability of rock walls in alpine regions. They provide a wireless sensor network where nodes can live off of a single battery for 3-5 years and survive the harsh alpine conditions. Monitoring the stability of slopes in the mountain's cryosphere is an important geological field of study in order to be able to develop theoretical models for natural hazard assessment.

Due to the inaccessibility of their field sites, e.g. on the Jungfraujoch or on the Hoernli ridge of the Matterhorn, both at 3500 m a.s.l. in the Swiss Alps [38], data delivery using radio communication is essential, as well as real-time data delivery to external sites and the possibility to store data over longer periods at each node are important properties, to obtain sensor information as fast and reliable as possible.

The PermaSense nodes are equipped with a custom sensor rod to measure temperature and resistivity at different depths up to a meter into the rock. Similar to the extensometer used in the SLEWS project, a crack meter is used to measure crack movements. The WSN is connected to the Internet using a combination of Wi-Fi and GSM/GPRS at the base station.

Wireless Sensor Networks provide the foundation for the study of environmental parameters particularly in extreme environments. The goal of a self organizing, data gathering network, provides a valuable tool for geo-engineers, enabling the observation of an area in nearly real time.

2.3 Wireless Networking Technologies

Networking technologies play an important part of our day to day lives. Looking at smartphones we see that several technologies are present in our smartphones:

GSM/3G, Wi-Fi and Bluetooth as the most established standards. These technologies are one of the basic principles on which our system relies on, to be able to establish data connections between the participants of a sensor network deployment. They cover different aspects of wireless transmission requirements and transmission ranges.

2.3.1 Bluetooth (IEEE 802.15.1)

Bluetooth is a short-range communications technology. It was originally intended to be a wireless replacement for cables on phones, headsets, keyboards and mice [8], but finding more widespread use in different areas. Recent years have shown applications in healthcare to connect heart rate monitors to smartwatches or the connection of consumer EEG headsets to tablets and smartphones to visualize brainwaves or attention levels such as the NeuroSky MindWave Mobile [34]. In-vehicle systems make use of Bluetooth to connect the user's smartphone to information sources such as vehicle diagnostics, or provide phone calls via the car's stereo system.

The application specific ranges of Bluetooth devices vary from 1 m (class 3 radio), to 10 m (class 2 radio) and up to 100 m (class 1 radio). By design Bluetooth has a very low power consumption profile (2.5 mW for class 2 radios). This can be seen in the subsequent improvements of the Bluetooth standard, where the focus has been on reducing the consumed energy instead of improving data transfer rates. The recent inclusion of Bluetooth Low Energy (BLE) in to the Bluetooth Core Specification Version 4.0 consumes between 1/2 and 1/100 the power of classic Bluetooth technology [7].

As Bluetooth is available in smartphones, it is used especially when locally sharing images between two smartphones. To establish a connection devices need to be discoverable, and then scan for visible devices. After the user controlled pairing process, devices can communicate without further authentication in a peer to peer fashion using the unlicensed industrial, scientific and medial (ISM) band at 2.4 to 2.485 GHz.

2.3.2 Wi-Fi (IEEE 802.11)

In comparison to Bluetooth, Wireless-Fidelity (Wi-Fi) provides faster connectivity between mobile phones, computers, media players and other devices [50]. Also, with the traditional goal of eliminating cables, it is now the most wide spread technology for offering internet access for example at home, bars, hotels or airports. It provides bandwidths of up to 1.3 Gb/s (802.11ac) to cope with data intense tasks such as video streaming or the transfer of large files. As it provides a higher bandwidth than Bluetooth, the energy consumption of Wi-Fi radios is 2 to 3 times higher compared to classic Bluetooth [19].

In general, Wi-Fi requires an access point (AP), connected to an infrastructure network, to which clients can connect, see Figure 2.6. Through this the clients can connect to the intra- and Internet. Connections are usually secured using the Wi-Fi Protected Access 2 (WPA2), to protect against eavesdropping of network traffic between the clients and the access point.

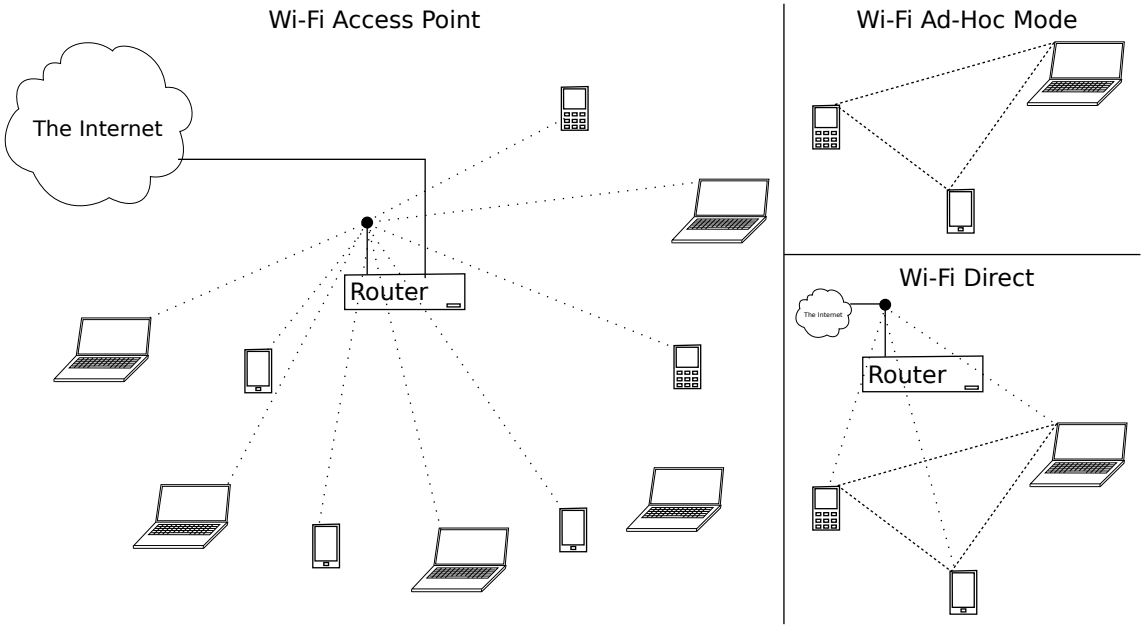

Figure 2.6 Overview of the available Wi-Fi modes. The connection to the Internet via an Access Point is the most common variant. Wi-Fi Ad-Hoc is the predecessor to Wi-Fi Direct. Both support peer-to-peer connectivity, but in case of Wi-Fi Direct without disconnecting from an existing Access Point.

Additional to the infrastructure mode of Wi-Fi, we can make us of Wi-Fi Direct [51]. Wi-Fi Direct is a certification mark that stands for devices capable of establishing direct peer to peer connections without the need for a third device, i.e. the access point. A similar legacy protocol is the Wi-Fi Ad-Hoc mode. This enables direct connections between devices without the need for an AP. The advantage of Wi-Fi Direct is that it can directly communicate with other devices while simultaneously keeping a connection to an infrastructure network [52], see Figure 2.6. Similar to Bluetooth, devices must be discoverable to create Wi-Fi Direct groups for communication directly between devices. The application area of this is to share data between devices, connect to printers, cameras or remote displays.

2.3.3 Cellular Networks

In competition with fixed telecommunication lines, wireless systems were developed. Originially, the Global System for Mobile communication (GSM) was designed to purely provide voice telephony services to wireless devices [14]. Here, mobile stations connect to a base transceiver station that is connected to the land telecommunications network to establish calls between mobile stations as well as to and from landlines. The Short Messaging Service (SMS) was one of the first popular services on the GSM system, that has been extended to include the exchange of images and other media. Due to the high demand of dial-up modem use on fixed lines, first data capabilities of up to 9600 b/s were added to the mobile network.

The main evolution cellular networks underwent, was the change from circuit switching to packet switched communication, to provide always-on data connections with-

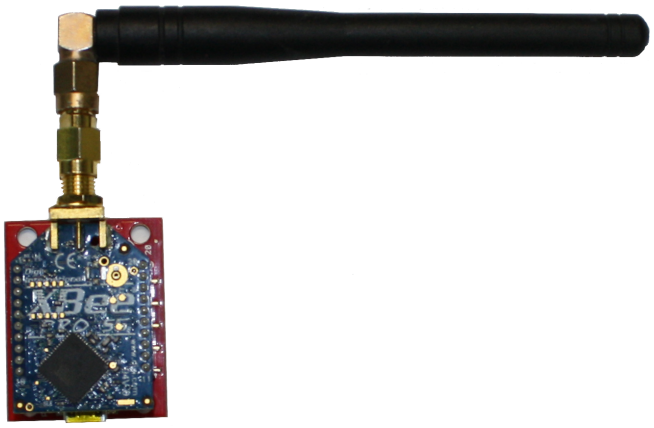

Figure 2.7 An XBee module mounted on an adapter to connect it to a USB port. The USB port is located at the bottom edge of the board. This module is then connected to an Android powered device to sniff the network traffic.

out permanently requiring a channel on the base station. From here on different generations of functionality were added, mostly to improve data rates, and catch up with bandwidths available for fixed lines. At the time of writing the third generation (3G) of mobile networks provided by the Universal Mobile Telecommunications System (UMTS), provides gross channel transmission rates of up to 14 Mbit/s [15], with fourth generation (4G) networks slowly being introduced in selected urban areas, supporting gross channel transmission rate of up to 300Mbit/s downlink and 170Mbit/s uplink [1]. In recent years, data plans for mobile handsets have become cheaper, making it affordable to the general public, finding widespread adoption especially in the combination with a smartphone.

2.3.4 IEEE 802.15.4

The standard IEEE 802.15.4 describes a low-rate wireless personal area network. It aims at providing a simple and flexible protocol, that is easy to install, provides reliable data transfer at extremely low cost with a reasonable battery life [43]. The application targets are radio modules for sensor nodes and custom devices that require a wireless connection but with little bandwidth. Figure 2.7 shows an XBee Pro module operating at 868 MHz. It is mounted to a serial to USB adapter, and can be used with any USB host device that has driver support for the given module. This way it is possible to create networking topologies using this standard that are either peer to peer or star communication patterns. The available frequency bands are 780 MHz, 868 MHz, 915 MHz, 950 MHz and 2.45 GHz, with bit rates between 20 kbit/s and 1000 kbit/s, depending on the applied frequency band and modulation [43, p. 163]. Although communication ranges of up to 1.6 km are given in hardware specifications [12], communication ranges depend on location and orientation of the radio module. Due to the dynamics of wireless signal propagation it is hard to predict the communication ranges in the field.

Due to the low cost, small size and feasible transmission ranges of up to 1.6 km make IEEE 802.15.4 an ideal radio network for WSNs.

3

Related Work

We differentiate the existing related work into two categories, namely *Surveying* for tools that gather tracks the user has covered as well as the creation of map annotations, and *Map-Making* techniques, that either cover the process of map generation and/or methods of obtaining maps.

Geopaparazzi and *OSMTracker* are both geological surveying tools with the focus on GPS tracking and map annotation techniques. Captured images are saved with GPS coordinates and orientation vectors.

iButton Assist and *idNotebook*, both developed in the context of the PermaSense project, aide in acquiring location information on deployed sensor nodes. Where *idNotebook* is the simpler of both approaches, *iButton Assist* allows to synchronize collected information with a server if there is an available data connection. The *GSN Project* is used by the sensor networks deployed with the help of the two previously mentioned approaches to display and evaluate sensor data. *SensorTune* allows for better placement of sensor nodes according to the interpretation of audio signals representing the reception quality of previously deployed sensor nodes.

As the previously mentioned approaches assume that we already have maps available to be used, we will take a look at how we can create our own maps, or offline data sets that we can pre-install onto our devices. The *Mobile Atlas Creator* is such a tool to create map tile bundles to be installed on mobile devices for map display. It incorporates several map sources and tools to select areas of arbitrary shape to be included in a map archive or database. *MapMill* and *MapKnitter* are two closely related web tools. The former is a crowd sourcing approach to selecting appropriate images from sets of aerial images to be used with the latter. This tool then enables the manual transformation of images, i.e. dragging, rotating and stretching onto a reference map, to stitch a series of aerial images into a custom map. *MapCruncher* is a similar approach by Microsoft. It's a Windows tool that allows for georeferencing of images with the use of manually added control points on each image with according points on a reference map source.

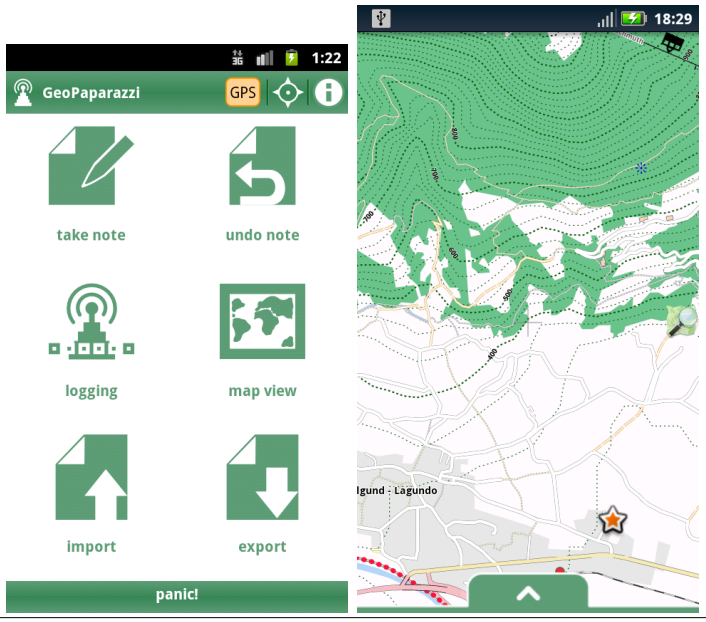

Figure 3.1 Screenshots of *Geopaparazzi*. On the left we see the main menu, the starting point for map annotations. On the right we see the map view displaying vector maps.

3.1 Surveying

Surveying or geological surveying is the process of systematically investigating an area, whilst taking notes on geological features, landforms or the composition of rocks beneath the ground. It includes the creation of maps, gathering of georeferenced images, GPS tracks and further annotations describing the area. Later, this information is then used to create geological maps of the visited site. In the following, we discuss mobile phone applications that aid in this process.

3.1.1 Geopaparazzi

Geopaparazzi is an application that runs on Android powered devices. Its main features are the creation of GPS tracks, capturing of georeferenced images including the orientation of the device whilst taking them [2]. Also you have the possibility of adding georeferenced notes in the form of voice recordings, sketches and handwritten notes. It is developed under the GNU General Public License v3 and the code is accessible in a public repository[1]. Furthermore, its map display supports online as well as offline maps. The latter can be added prior to surveying.

Collected data can be exported to the *gpx* and *kmz* file formats, with the main purpose being used with Geographic Information System (GIS) tools. The GIS tools *uDig* [42] and *BeeGIS* [39] are able to import data directly from the local database created on the phone by Geopaparazzi. Then, with the help of the imported data it is possible to create or extend geological maps.

[1] https://code.google.com/p/geopaparazzi/

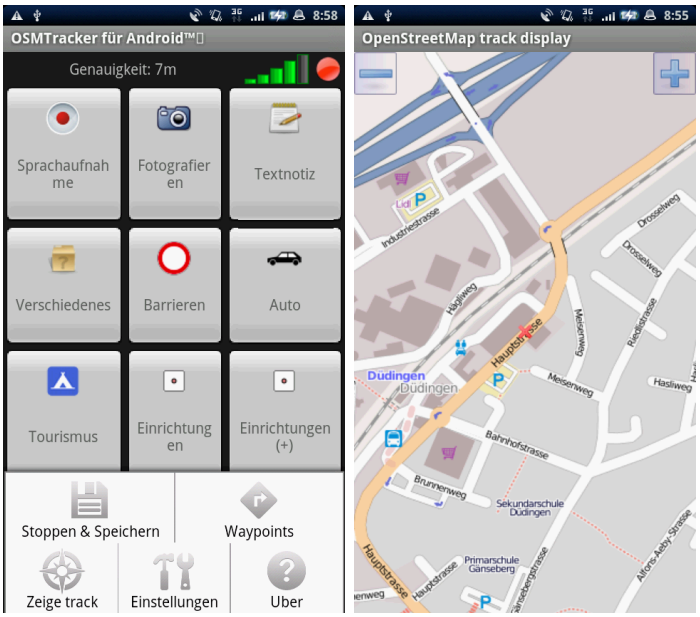

Figure 3.2 Screenshots of *OSMTracker*. On the left we see the main menu, from which we can start the recording of GPS tracks and place annotations at our current location. On the right we see the map view displaying our current location on top of OpenStreetMap map tiles.

3.1.2 OSMTracker

OSMTracker is a similar tool to Geopaparazzi, which is developed under the GNU General Public License v3 [2].

It is a tool to support the collection of data to create maps for OpenStreetMap (OSM), hence the name OSMTracker [35]. Note, OpenStreetMap is a collaborative, open source approach to create free maps of the world. The tool tracks your location during your tour, that can be exported to the *gpx* file format. During your trip you can add way points, to specify for example road types, signs or amenities. Also, it allows the creation of georeferenced voice notes, text notes and pictures. The gathered information can then be used with the Java OpenStreetMap Editor (JOSM), to create or add to existing OSM maps.

Although both *Geopaparazzi* and *OSMTracker* support the collection of data in form of annotations, or tracks that the user has covered, neither of them provide any means of doing so collaboratively. Both tools are designed to be used independently, and then join the collected information on a computer after leaving the field. We have to remark that the former tool provides the possibility of using offline maps, the latter only through displaying cached tiles that have been downloaded on viewing the map area for the first time. Thus, examining an area of the map that has not been shown will result in an empty map with the latter.

[2] https://code.google.com/p/osmtracker-android/

3.1.3 iButton Assist

iButton Assist is an Android application to help in keeping track of deployed *iButton* devices [46]. These are small embedded devices that can measure temperature or humidity, or in varying configurations for authentication purposes. Despite their size of 1.6 cm diameter, they can store sensor data in an 8 kb flash memory [30]. See Figure 3.3, on the left for an example. Data logs can be read using a 1-Wire transmitter/receiver pin on the bottom of the device, as iButton devices are not capable of wireless transmissions. With this application it is possible to record the location where a device has been deployed, combined with images taken of the devices at their respective location. The latter is an important option as the deployed devices can be hard to find due to their small size. The goal of the application is to enable the mass deployment of, for example, miniature temperature loggers, with the possibility to synchronize already recorded data with a remote server. The adapter used with their software to read data from the sensors can be attached to a USB-host capable android smartphone.

3.1.4 idNotebook

idNotebook is an Android tool to track an arbitrary deployed device [47], see Figure 3.3. The location is annotated with pictures and an id of the deployed device given by the user. The data is saved in comma separated text files on the SD-card of the smartphone, to be exported by the user for further processing. Both idNotebook and iButton Assist are developed by the PermaSense Consortium.

Both applications are promising tools for keeping track of deployed devices, but as they are not open source[3], we can not extend them or include them for our purposes. iButton Assist only supports one device type, whereas idNotebook is very basic without any visualizations of deployed nodes or the network.

3.1.5 SensorTune

SensorTune addresses the challenge of setting up a Wireless Sensor Network for non-expert users, by deploying the nodes in iterative steps using sonification of received signal strengths of previously deployed nodes [10]. Analogous to tuning a radio users can hear the increase or degradation of received signal strength and quality. As the receive signal strength lessens the perceived signal, a classical piece of music, becomes quieter and is disturbed by noise. When the node that is currently being deployed loses connection to the network the music turns silent and only noise can be heard. The authors of SensorTune could show that using this technique helps non-expert users to setup a WSN significantly faster than using a graphical UI representing received signals of the deployed sensor node.

[3]Only a desktop version of iButton Assist is available at: https://code.google.com/p/iassist/

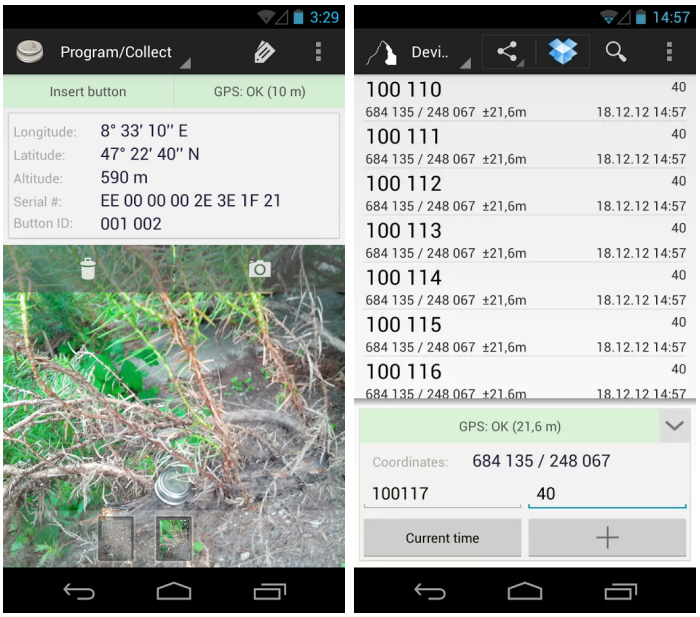

Figure 3.3 Screenshots of *iButton Assist* [46] on the left, and *idNotebook* [47] on the right. Both screens show the current logging status of the last deployed device(s). On the left you can see an *iButton* sensor between the roots of a plant for temperature measurements, annotated with its current position and the sensors unique ID. The right screen shows a list of IDs of deployed devices and their distances to the user.

3.1.6 GSN Project

The Global Sensor Network (GSN) Project is a data processing engine to simplify the integration of multiple sensor networks [24]. Sensor data can be streamed in to GSN via multiple computers acting as an acquisition network, see Figure 3.4. These streams can then be bundled and processed, e.g. to calculate the average of a value and then divide it into two data streams, one for storing in a database and one that is displayed in real time on a web site. The configuration of the streams and processing steps is specified via XML files.

An example project using GSN is PermaSense to process and visualize its sensor network data from its deployment sites[4].

3.2 Map-Making

Also, related to our work is the process of map making. We do not primarily focus on the generation of digital maps, but rather on the live acquisition of aerial images via an android powered device lifted by a balloon or kite, and the possibility of selecting images based on position and sensor data.

[4]http://data.permasense.ch/

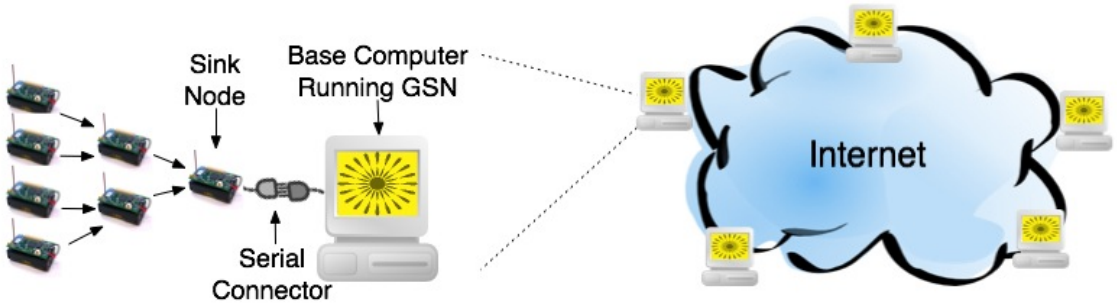

Figure 3.4 An overview of the GSN project [24]. Multiple computers act as a backbone of the acquisition network. This way, data from multiple WSNs can be aggregated, stored, processed and visualized.

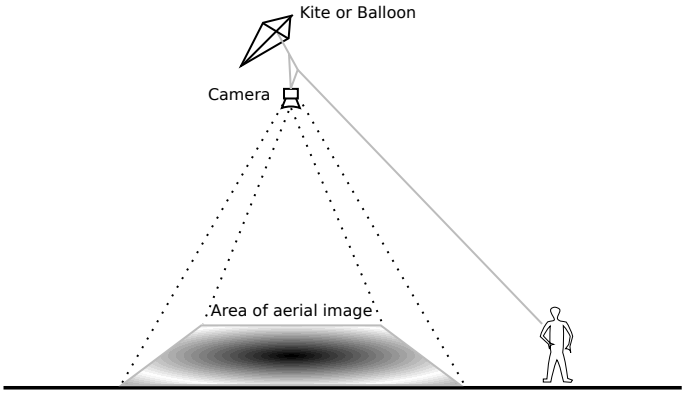

Figure 3.5 An overview to kite/balloon mapping. A camera is lifted into the air with a kite or a balloon whilst continuously capturing images of the ground below.

3.2.1 Balloon and Kite Mapping

Balloon and kite mapping is an inexpensive method of obtaining aerial images. It involves a camera or smartphone with a camera tied to a balloon or kite, whilst being set to a mode where the images are taken repeatedly in an infinite loop. As the balloon/kite is flown above the area that is to be surveyed, the camera captures the ground below. The captured area of the images can be controlled by either varying the height at which the camera is flown or by controlling the zoom. We can obtain remote access to a camera to a certain degree, if the device runs Android, with our system installed. But with standard digital cameras this is not as easy, as we can not run our own software on them, and would require mechanical controlling of the device. There are Android powered devices with optical zoom, e.g. the Galaxy Camera and the Galaxy S4 Zoom.

After the flight the captured images need to be filtered for duplicates and blurry images. Depending on the way the camera is attached to the balloon or kite, blurry images occur due to shaking or dangling of the camera. The selected images can then be georeferenced and further processed in to a map.

Figure 3.6 Screenshots of the browser based image selection tool *MapMill*. Using three categories users can rate each displayed image's quality. The goal is to filter blurry and non-vertical images, and then use the remaining for aerial map making.

3.2.2 MapMill

MapMill facilitates the selection process from series of aerial images [26]. It is a crowd sourcing approach meaning that the workload is distributed across all users. This is necessary as there are thousands of images that are created by map making communities, from which the best need to be selected. Each user can browse the images one after the other and rate it according to sharpness and the angle of the shot. The rating is done in three categories: *really good*, *just ok* and *not useful*, see Figure 3.6.

The efficiency of this method depends on the number of people participating in the project. Also, if you want to use this in an area without Internet connectivity you have to install and run your own instance to be provided to other deployment participants through a local network. In this case the size of the group of people aiding in the selection process depends on the people involved in the deployment. As smartphones provide us with sensors to measure the movements of the device, such as the accelerometer or compass, we could record these with every captured image and use them to help us in automating the selection process to find images with the right properties. From the accelerometer measurements we can decide if a device is pointing down or to the sides. A further application area is to help government agencies asses damages from natural disasters, by rating the amount of damage to infrastructure or if they are still flooded [25].

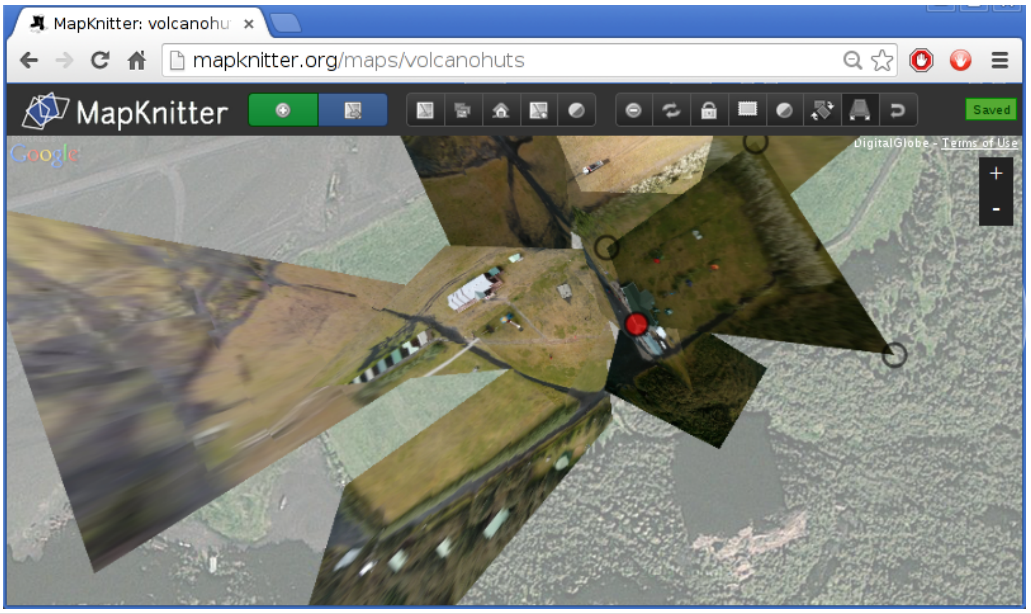

Figure 3.7 Screenshots of *MapKnitter* a browser based collaborative tool to align aerial images to create maps.

3.2.3 MapKnitter

MapKnitter is a web based, open source tool to manually align images onto a reference map [27]. Using this tool you can upload your own aerial imagery images to a website running the software and collaboratively combine them into a GeoTiff and TMS/OpenLayers map [40]. Once an images is uploaded it appears at the location specified for each created map. Then, each image can be rotated and resized to match scale of the underlying reference. With a second mode an image can be transformed to match the reference map, to compensate for the original aerial image not pointing directly to the ground. For reference maps the user can select from a series of online tile sources, such as Google Maps, Bing Maps and OpenStreetMap.

One of the goals of MapKnitter is speed and simplicity with which you can create maps and align images without any initial knowledge about image projections or image stitching processes. After manually aligning the selected aerial images we can either save the result in a variety of offline and online map files or we can further refine the results with other geographic information system tools, such as ArcGIS or QGIS [13, 41]. Most important for us is the possibility to save the result in a TMS/OpenLayers compatible archive, that we can deploy with our system to provide offline maps created from our aerial images. One of the drawbacks that come with from the trade of between ease and use and mapping of aerial images is that the transformation of each image is only possible by adjusting the corners and stretching the image into the corresponding direction. This limits the possibility to stretch areas of images independently due to the angle at which the image was captured.

3.2.4 MapCruncher

An alternative to browser based manual aligning of images to a reference map is MapCruncher. It is a tool for Microsoft Windows that aligns images with the help

of control points. The process consists of the user adding control points to the aerial images, i.e. matching locations on each images to locations on the reference map. MapCruncher then transforms the source image according to the control points. The more points are given for each image the better the final projected images fit to the reference map. Similar to MapKnitter the reference map is fetched from an online source, but providing only Bing Map tiles

Compared to *MapKnitter* this tool provides better support in mapping aerial images taken at steep angles, as we can add multiple control points to a single image. This way we are not limited to stretching an image into four directions, but we can also adjust arbitrary parts of the image. Unfortunately it is not possible to view multiple aerial images at the same time. We can only adjust each image to the reference map and not to previously aligned images.

4

Design

In the this chapter, we will look at the design choices we made to develop a system to improve the deployment of wireless sensor networks. We chose the smartphone as the platform for our system as they are small, powerful and widely-used. A smartphone comes with multiple sensors, GPS and radio modules that we can employ to gather information about the location and events during WSN deployment.

In the following, we will look at general requirements for opportunistic sensor network deployment support. We divided these into surveying, deployment and maintenance tasks, using the arising requirements from these areas to establish our design choices. An overview of our design is shown in Figure 4.1.

4.1 General Requirements

Experience shows, that sensor networks are a lot of work. The setup and maintenance of nodes requires expert knowledge, and even with this knowledge it is hard to avoid mistakes and unforeseen errors. It is not uncommon that deployment sites are located in remote areas. Thus, a WSN is deployed with a team of participants, that deploy the nodes together. Often these areas are only accessible on foot or climbing through rough terrain.

4.1.1 Opportunistic Deployment Support

At the end of the day and during the deployment, we need to know when and where each node has been placed. Throughout the deployment, we want to record information regarding the deployment. This includes the notation of further sites of interest, landmarks that provide orientation, or images of the terrain. Also, we need to exchange this information with other participants to share the overview that is obtained of the site. Furthermore, we need to communicate whether all nodes in

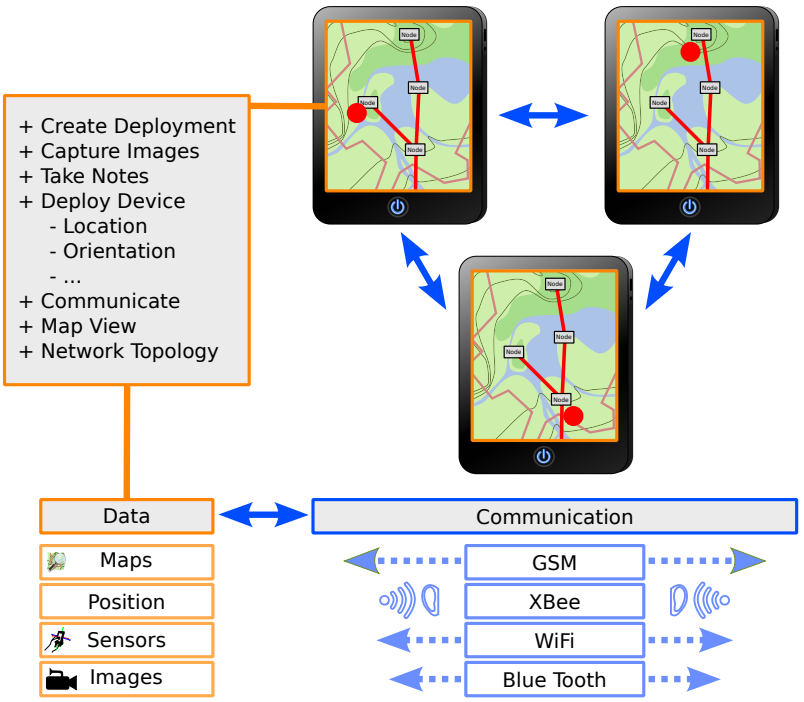

Figure 4.1 Overview of system components and activities. The user creates and adds information to our system during the deployment. This data is exchanged via GSM, Wi-Fi or Bluetooth, between participants running our software. With an additional XBee device attached to the smartphone we can sniff the traffic of the WSN. The obtained data is logged and displayed on the map. This way we can display the network topology, as well as sensor data on the user's device.

the network are able to reach the base station. If this is not the case we need to reconsider the nodes' location or the position of the nodes' antenna.

4.1.1.1 Mobile Devices

During the deployment we need to keep a log of our actions, e.g. when and where a node was placed. Currently we take handwritten notes and take pictures with a digital camera, to document our work. Unfortunately, this brings the limitation of the notes and images being located at the person who created them until they are distributed by other means of content replication. Our experience has shown, that people prefer using a smartphone to take pictures in contrast to using a point and shoot digital camera. Due to the available options to connect a smartphone to the Internet, they have become popular for sharing digital media with family and friends. The quality of components, e.g. the integrated camera, is improved yearly. Thus, smartphones have started to replace the compact digital camera. In addition to a camera, smartphones include a range of sensors: most commonly an accelerometer and a digital compass, but also a gyroscope and a barometer in midrange to high end devices. Furthermore, we can rely on the availability of Bluetooth, Wi-Fi as well as GPS receivers. These sensors and radio modules provide us with additional functionality that we can use to leverage support of sensor network deployment. Also, there are devices that support USB-host functionality. We can employ this to attach additional radio modules to the smartphone via USB. We developed our system for devices running Android Ice Cream Sandwich and up (Android API v14). This provides us the easiest way to develop custom drivers for devices connected via USB.

4.1.2 Use Cases

In the following we look at problems and inconveniences that occur during sensor network deployments. For this, we have divided the tasks into three categories: surveying, deploying and maintenance. We discuss the latter in a combined section as they share some of their requirements.

4.1.2.1 Surveying

Annotations: Before we can start with the deployment we need to consider the targeted environment. We need to know how we can access the site, and ultimately where it is possible or required to place sensor nodes. This can be done in advance,for example by the end-user, by adding annotations to a map. She has to provide images, scans, and maps of the deployment site, thus provide as much information as possible and necessary for the WSN deployers to obtain an overview on what the situation is on-site. With the knowledge about the environment it is possible estimate how the characteristics of the WSN will have to be in order to be fully functional. From this we can decide how we place the sensor nodes. For example: loosely on the ground, mounted on rocks, or placed on existing infrastructure. Also, it is important to know whether the planned locations of nodes are in line of sight of each other. Many of

the information need for these decisions can be extracted from a map, but is greatly improved with annotations generated on site.

Maps: During the survey of the site we need to know where we are. As our system is built on a smartphone we have two possibilities of obtaining and displaying a map. The first is via online access. The region we are viewing is downloaded and cached on demand, provided we have a data connection to the Internet. Nowadays, the easiest and cheapest way to obtain such a map is through online services such as Google, Bing, or OpenStreetMaps. They provide online, aerial images for free, with an option to cache map data in form of tiles or vector data. Also, there are commercial aerial/satellite imagery service providers. Whether a commercial offer is feasible depends on the budget of the WSN deployment. Another aspect is the resolution of the provided images and whether they are up-to-date. Thus, we need a possibility to obtain recent maps or aerial images of the deployment site. Ideally, the date of the aerial maps is the on day of the deployment. Finally, with the right map we can obtain an overview of the deployment site which helps us in keeping track of the sensor nodes, and our own location.

4.1.2.2 Deployment and Maintenance

Gathering of node data: As we mentioned before, a map helps us in obtaining a sense of orientation. With this, we can lookup where to deploy nodes or what the distances between sensor nodes and the base station are. To keep track of where and when the nodes have been placed and switched on we need to provide the exact location and time the nodes where placed. This helps to know from which point in time we might expect sensor data to be delivered to a remote server. After a certain time we can then conclude that the node is not integrated into the network and needs checking if the hardware is working correctly. Another possible solution is to change the placement of the antenna to better reach other nodes. Additionally, to the time and location of the node, it is advisable to record as much information about the node as possible. For geo-engineers it is interesting to know what the orientation of the device is to interpret sensor data. Also, we can help others in retrieving a node by capturing images of the node's placement, or at which height it has been installed.

WSN status: A further problem is that we have no direct way of obtaining status information about a specific node in the field. Currently, this is mostly done through a connection to a remote server to which the base station has been uploading the received sensor data. This way, have no intuitive view on what is happening in the network. As described in Chapter 2, sensor nodes usually do not provide us with any interfaces that tell us if the node is connected to the network. Also, we do not know what the sensors are reporting and whether this information is correct. Explaining to end-users what a debug LED is supposed to indicate is cumbersome, and only possible if it is mounted on the surface of the node. As Fernandez-Steeger et al. [17] pointed out, this is not always the case and greatly hinders the process of debugging problems in the network and to provide fast fixes for them. Thus, to improve the process of finding faulty parts as soon as possible and reduce the time we need to deploy a WSN, we need a way of monitoring the network's topology, and each single node's state. The process of walking back to the base station and asking someone

if the recently deployed node is received can be very time consuming. Monitoring the textual log messages on the server for messages from a particular node is only possible if we have a mobile data connection. Additionally, if there are problems at intermediary nodes or at the base station we have no indications if the respective node is communicating with the network. The easiest way to establish the connection of each node is to be able to listen into the network messages between the nodes and display them on a map. This way we can easily see what the current network topology is, and also where it might be feasible to further support the network with additional nodes.

Finding nodes: Sensor network maintenance requires knowledge about the location of previously deployed sensor nodes. Regarding the size of the deployed sensor nodes this is a difficult task. As shown in Chapter 2, a device of the size of a button, is easily overlooked, even at close distances. A map indicating the placement of the nodes, with annotations in the form of images, environmental landmarks or audio notes can greatly improve the retrieval speed of nodes. Furthermore it is feasible to support the user with hands free directions to a sensor node. This is is necessary if the user is carrying equipment or climbing through rough terrain.

4.2 File System Layout

We organize the data collected on the devices running our system. In the phone's storage we create folders for different data types and organize them in sub folders for each deployment. This way, each deployment is identified by its deployment id. The id is the `$timestamp` of the instance when the deployment is initially added to the device. The following folders contain the data created during surveying, deploying and maintenance:

- `daisy_deployments/`: Main storage for deployment data. Everything is saved within a `$timestamp.dpl` file. This is the binary ProtoBuf data of the main data structure. Furthermore we track the largest sequence number generated by the local instance in the file `$timestamp.dpl.seq`.
- `daisy_maps/`: Contains all maps used on the device. These are not deployment specific, but are synchronized to all devices.
- `daisy_images/$timestamp`: Contains all images from annotations and nodes.
- `daisy_recordings/$timestamp/`: Recordings of voice notes.
- `daisy_balloonImages/$timestamp/`: Original images captured by the balloon device. Also storage folder of the retrieved originals. Sensor data recorded during image capture is saved in `$image.jpg.pb` files as binary ProtoBuf data.
- `daisy_balloonPreviews/$timestmap/`: Storage folder of retrieved previews.

The file system layout aims to facilitate content retrieval by the user for later evaluation.

4.3 Communication

Combining the annotations and gathered information of all the persons involved in the deployment of the sensor network is a major step towards improving the methodology of sensor network deployment. This has to happen automatically, synchronizing the available data between devices, finally copying the available data to all participants.

As shown in Chapter 2, there are various communication devices already built into smartphones that can be used in distributing information across multiple devices. With the help of these radio technologies, we can detect if there are participants close by and synchronize the devices.

4.3.1 Discovery

There are several radio modules and libraries that provide discovery of devices and services in networks. We keep a list of all discovered devices, to give the user a choice to communicate with them later on. One way of obtaining knowledge which devices are nearby is to scan for Bluetooth devices. This scan only returns devices that are set to being visible to others, and this only for a limited time span. The returned scan results are Bluetooth devices, e.g. smartphones, that are close enough for communication. These can then be queried if they they are participating in the deployment, i.e., they answer to Bluetooth service connection requests, specific to our solution. Another possibility is to connect to Wi-Fi access points initially specified for the deployment. As not all devices connected to the same access point run our software, we have to find out which do. Thus, we have to connect to either a predefined IP and port or obtain a knowledge about participants via network service discovery. Also, we can discover devices nearby through Wi-Fi direct. In this case the discovery is analogous to the Bluetooth discovery process, with the network service discovery on top. See Figure 4.3 for an example of the participant discovery screen of our system.

4.3.2 Ranges

As our approach is based on Android phones we have the choice of using Bluetooth, Wi-Fi, and a mobile data connection. Our system works with all mentioned technologies to provide cross-platform connectivity. This is motivated from the fact that not all phones support Wi-Fi direct, as well as not all users subscribe to mobile data plans. Bluetooth can be considered the common denominator here, as it is a requirement since Android version 2.1, i.e. for a device to be Android compatible it must include Bluetooth transceivers [22].

The following list gives a short overview on communication ranges of the respective radio technologies:

- **Bluetooth**: Provides short range communication.
- **WiFi**: Provides mid-range communication at higher data rates.

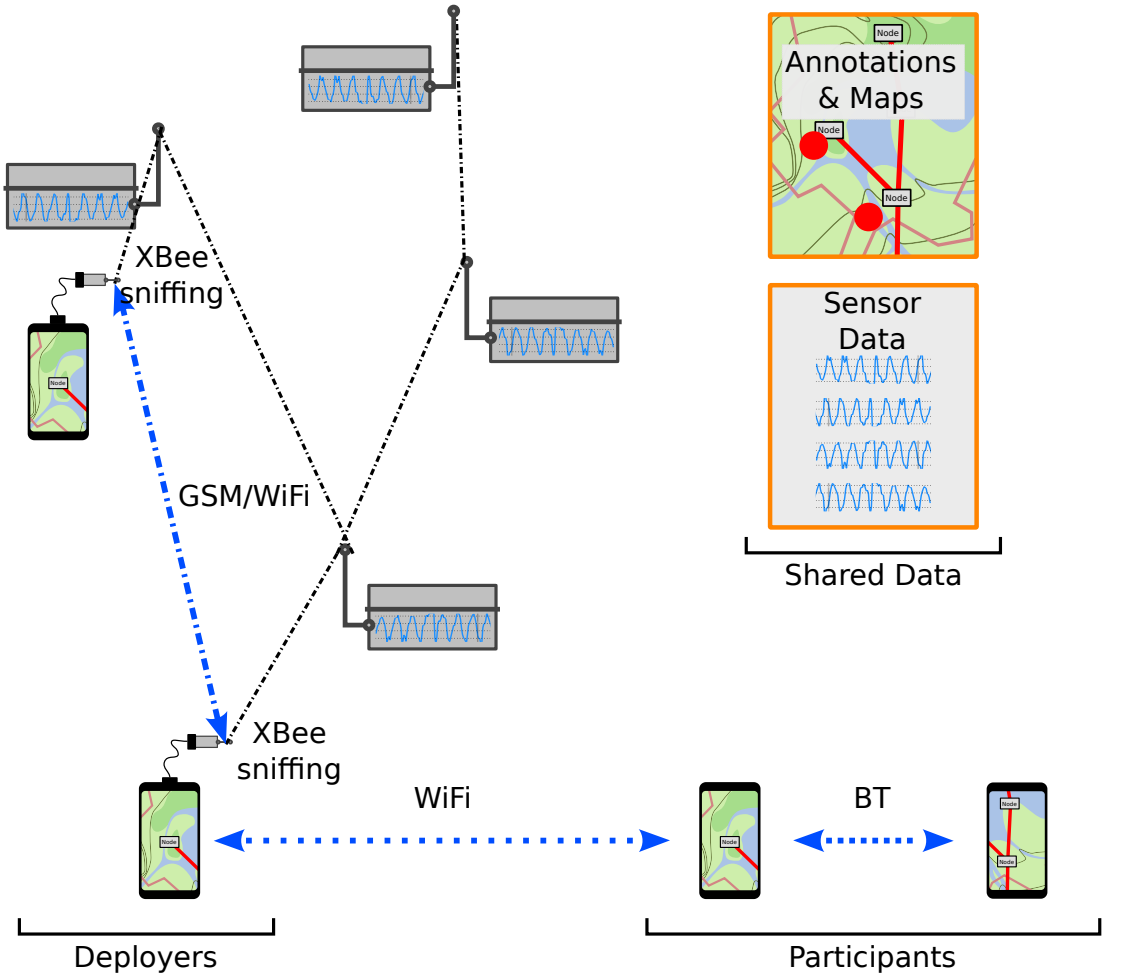

Figure 4.2 Overview on collaborative deployment support. Users can either track the deployment of a wireless sensor network on their mobile device, or actively deploy nodes. During node placement they scan the node's location, orientation, and additional images and annotations. Data is shared amongst the participants using GSM, Wi-Fi, or Bluetooth. If the user's device supports multiple communication channels, the first radio channel to connect is used for data synchronization. With an XBee radio module attached to a smartphone we can sniff the WSN traffic. The network packets contain sensor data as well as routing information about the network. Using this, we display each node's sensor data, as well the network's topology on the map.

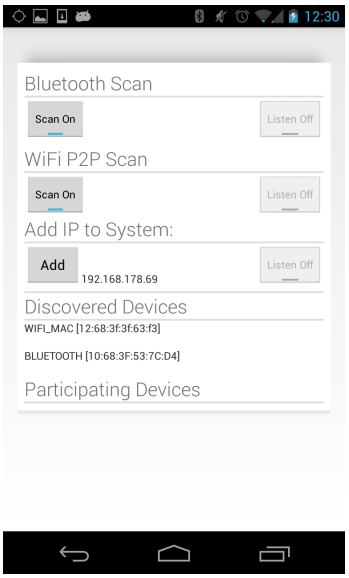

Figure 4.3 Device discovery view of our system. The user can select from the found devices which to query for initial deployment data. Once two clients connect to each other the associated connection information is saved and distributed to other participants for use as well. This way our system collects and distributes information on how to connect to each device.

- **GSM/3G**: Optionally allows global connectivity, if available in deployment area.

As shown in Figure 4.2, the choice of a communication channel depends on the distances between the participating devices. One of the design goals of our system is that it does not matter which radio channels are used throughout its use. It is possible to mix them between participants, i.e., to allow Bluetooth only devices, e.g. a smartwatch, to join the network whilst other participants communicate via Wi-Fi or GSM.

4.3.3 Bootstrapping

To keep track of all data related to a deployment we need an initial creator of the deployment data set, i.e. an identifier with a data structure that keeps track of notes, images, maps that correspond to a specific WSN deployment. The idea is that we either start a deployment on our device or we join a deployment. We call this process *bootstrapping* as we require additional information about the initial data set, as well as already participating devices. To keep this a simple as possible we provide an interface, see Figure 4.3, to connect to devices returned from discovery scans, to request all necessary data to participate in the network.

4.3.4 Synchronization

After bootstrapping every connection between two devices running our system is to synchronize the data sets available on both sides. The goal is to guarantee that

both users end up with the same data on both devices after the synchronization. The most efficient way here is that each device only transmits what the synchronization partner has not received so far, or continues to send parts of files that are not complete yet. We achieve this by keeping track of all the objects each user' device has created, through sequence numbers. This way each user has a unique identifier and a sequence number for each object. During synchronization both sides exchange the highest sequence number for each user id they have. Then they can look up which sequence number per user id they have and request the missing. This way both sides know what is missing and transmit the requested objects accordingly.

As images and maps can take between several seconds and minutes to transmit a notification informs the user about the progress of the current file being in transmission. This behavior is is similar to sharing files via Android Beam, and is thus familiar to the user.

4.4 Surveying

Our system provides a map view to display multiple layers of maps with the user's annotations on top. We provide a selection of online map sources that come with the *osmdroid* library we utilize to display map tiles on the device.

4.4.1 Online/Offline Maps

As shown on Figure 4.6, we provide the user with a menu to select from a list of predefined online map sources. Once a tile has been downloaded from the map server using a Wi-Fi network or a mobile data connection, it is cached on the device to avoid multiple downloading of the same tiles. These cached tiles are available even if we have no connection to the Internet. One way of obtaining a map of an area would be to previously view the entire area at each available zoom level. As this is not a very convenient way for larger areas, to view the whole site at all zoom levels, we can also add archives with tiles to predefined folder on the device. To obtain maps, we can either download tile sets with the tool *Mobile Atlas Creator* [32], or use the resulting TMS archive from aerial images stitched with *MapKnitter*.

4.4.2 Annotations

During the initial survey of the deployment site, the user can view her location on the provided maps. From the map menu she can create text and voice notes as well as capture images of the area around her location. They then appear on the map, each with an icon at the respective location. Selecting an annotation will show a tool tip window in the map, to view the preview of the image, the text of the note, or a button to open the audio annotation at the corresponding location. All annotations are saved on the device, and are exchanged between all participating devices. This way we can track where someone has been, and what has been annotated. Additionally, the user can select an area to be guided to with our audio guide using head phones, by long clicking at any point on the map, see Section 4.5.3.

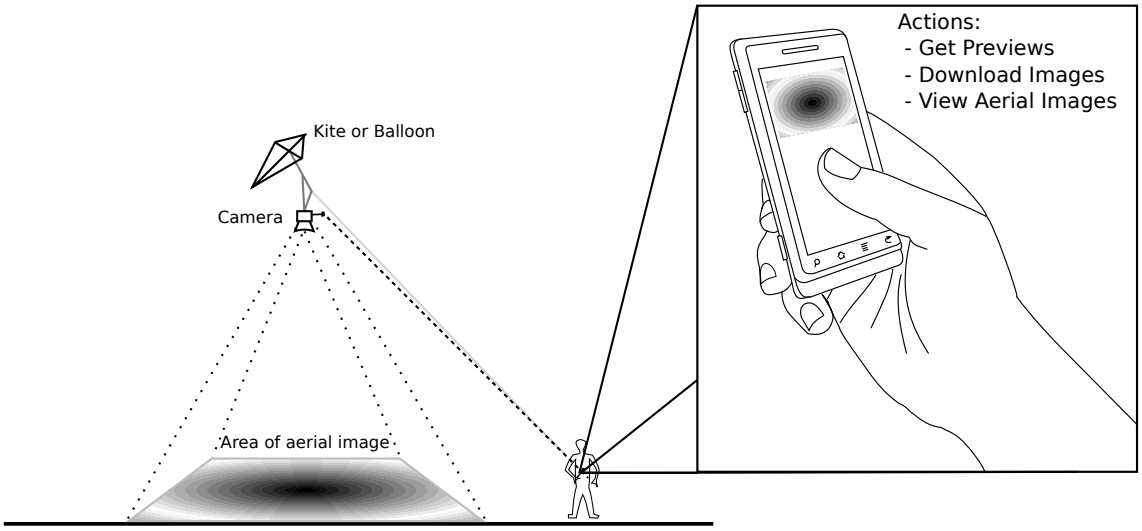

Figure 4.4 An overview of our kite/balloon mapping approach. A smartphone is launched with a kite or a balloon into heights of up to 300 meters. As long as the device is in communication range of either Wi-Fi Direct or Bluetooth we can connect to the smartphone and download the captured images. First we can stream the previews of all captures images so far to the device on the ground. Then we can select the images we want to download and view in full resolution.

4.4.3 Mapping

Google Maps and Bing Maps are good starting points to consult for aerial imagery. Still, we decided to improve the method of creating our own aerial images. We can create our own maps that are up to date, i.e., from the day of the deployment, with the help of an Android camera or smartphone attached to a kite or a balloon.

4.4.3.1 Ballon/Kite Aerial Image Retrieval

To improve the process of map making and the retrieval of map aerial maps while the capturing is in progress, we decided to create a protocol to establish a connection to the device in the air and retrieve the images from it, see Figure 4.4. After the connection between the two Android devices has been established and verified that the device in the air is currently providing images from the camera, we can select at which resolution and speed the images should be captured. Also we can select at which resolution the preview images should be transmitted. With each received preview we also get status information about at which altitude a.s.l. the device is currently flying at, how much space the memory has left for images and what the charge of the device's battery is. The preview images are cached on the connecting device, and do not need to be retransmitted. The user can either manually request the latest images available on the device, or set the request the missing previews to be transmitted to his device. The received previews appear in a scrollable list and can be selected to be either viewed in the resolution set for previews, or request the original image to be downloaded to her device. This way it is possible to review what images have been taken so far and manually download those images that appear to

```
battery_level: 90
free_bytes: 1982160896
barometric_pressure: 985.688232422
compass_angle_x: 15.7571763992
compass_angle_y: -13.5315732956
compass_angle_z: 5.59644889832
location {
  latitude: 63.69090888
  longitude: -19.54062887
  altitude: 264.899993896
  bearing: 81.9000015259
  accuracy: 3.0
  provider: "gps"
  speed: 0.5
  time: 1377620164000
}
acceleration_x: -1.10285949707
acceleration_y: 2.17993164062
acceleration_z: 6.69160461426
```

Figure 4.5 An example of the data saved with each image.

satisfy the properties to be stitched into an aerial map. This way, we can obtain aerial images without the need to retrieve the kite/balloon and can continue with the capturing process.

With each image we also obtain a file that contains the data from the sensors that were recorded close to the instance the image has been captured. An exemplary output of the data saved with each image is shown in the Figure 4.5. The sensor data can then later be used to filter images according to their location, height, or depending on the sensor data on the angle of the device, or how by how much it was shaking.

4.4.3.2 Stitching

To create a deployable map from aerial images we either use *MapKnitter*, or *MapCruncher*. The former allows us to align the images simultaneously with multiple users. Both tools allow us to manually align the images by adding reference points, or by manually dragging the images into place. At the end of the stitching process we choose to export the created map as a zipped Tiled Map Service (TMS) archive.

4.4.3.3 Distribution of Map Packs

We chose the map view library *Osmdroid*, to display online, cached, and map archives. The archives are read from a predefined folder for each deployment. For each archive or tile data base, we require a text file with the name of the map inside the archive or data base. The text file has the same name as the archive with the

addition of a postfix `.mapName.` This allows the splitting of an archive into smaller files, as well as multiple maps in a single archive.

In the case of TMS tile archives, we need to provide an empty file with the same name as the archive, but with the added postfix `.tnms` to notify our system that the archive uses the TMS tile naming, instead of the OSM tile naming. Both tile naming conventions are identical, with a difference in where the origin of the numbering of the tiles is. In case of TMS the origin is in the bottom left corner, where OSM tile numbering has the origin in the top left corner of the map. If an additional `.tms` file is found we switch the y axis of the tile numbering to correctly display TMS tiles from offline archives. If an archive contains OSM tiles, we do not need to provide the additional `.tms` file.

Once we have manually copied the archive with the additional configuration file(s) into the `daisy_maps/$deployment_id` folder of a device it will be distributed to the other participants as soon as they reconnect to each other for synchronization.

4.5 Deployment and Maintenance

During the deployment and for maintenance tasks we display the locations of the nodes on top of the selected map layers. With the help of an XBee module attached to the USB host of our deployment device, we can sniff the WSN network packets that are being sent by nodes within reception range of us. We can analyze these packets and thus display the nodes battery status and the sensor data from the packet's payload.

4.5.1 Gathering of Node Data

To identify wireless sensor nodes in the field we mark them with pre-coded QR codes or NFC tags. They can store configuration data, such as channel and group ID of the sensor network, required for WSN traffic sniffing. The QR code scanning process itself allows us to automatically record the orientation of a sensor node. This is calculated from the angle of the QR code in the image and the orientation of the smartphone at the time when the QR code is detected by the camera. Included in the node identification step, we record the location and height above sea level using GPS, which can be repeated to improve the location accuracy by keeping the location with the smallest location error. Furthermore, we can set the id and the orientation of the node manually by measuring the orientation of the deployment device while it is aligned to the sensor node. The id is entered manually into a text input field. We can estimate the node's relative height above ground from the barometric pressure difference between ground and sensor level. Additionally, we can record text and images associated with the node that is currently deployed, see Figure 4.7, on the right.

4.5.2 Visualization of Network Status

The possibility of non-expert users successfully deploying a sensor network requires a representation of information understandable and intuitive way without needing

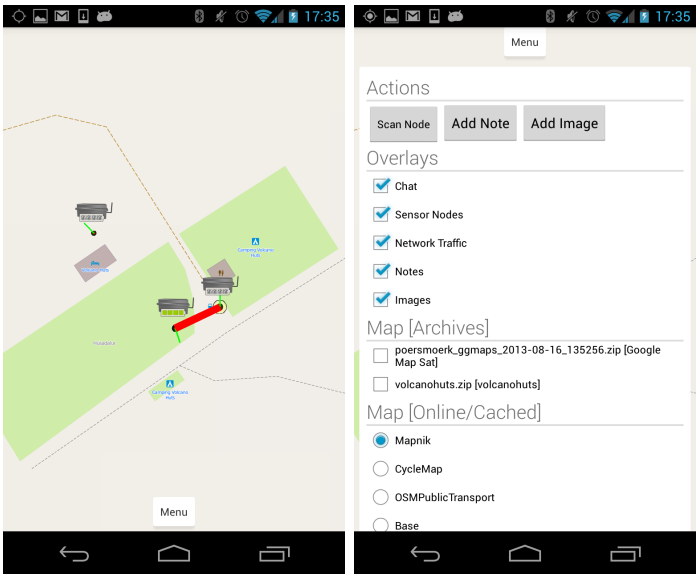

Figure 4.6 On the left we see the map view with two sensor nodes, of which one is connected to the base station. On the right we see the map options menu. Here, we can select which overlays to display, which map archives should be displayed on top of the online/cached map source.

a background in computer science. Manually filtering for messages via a Telnet connection to a remote server does not help in understanding what the network topology looks like. In Figure 4.6, left, we can see scanned nodes and the network topology on the map. If we have not received any messages from a sensor node yet, we can not tell what the battery state is. The state of one node is known, as this node is already connected to the base station, as indicated by the red line. By selecting the nodes on the map we can see the assigned images and ids, thus can identify the base station. One node is already connected to the base station, the top left node needs adjusting to be able to communicate with the other two nodes.

The information about the status of nodes is received by sniffing the WSN's traffic with an XBee device attached to our smartphone. All received messages are logged, and those that contain payload data, i.e. sensor data, are assigned to the respective node on the map and can be viewed in real time, see Figure 4.7, on the left. This data is also synchronized along participants and thus is viewable by all after synchronization.

4.5.3 Finding Nodes

When the team later tries to find the nodes again to carry out maintenance tasks, such as replacing depleted batteries, we can accurately guide them to the respective locations. Considering the network was deployed with our system and the location of each node recorded with additional information about orientation landmarks, or pictures where the node is located, a user can easily find the device. We designed the interface in such a way, that hands-free and eyes-free operation, e.g. via audio feedback, is possible, enabling the maintenance crew to focus on the task at hand in possibly hazardous environments. The user then follows instructions supplied via headphones that indicate the direction and distance where a selected node is

Figure 4.7 On the left we can see the sensor data reported by a sensor node. On the right we can see the sensor an image associated with a deployed node.

located. The distance and direction is perceived in the speed and pitch of repeated signal. This mode of operation supports scenarios where the user has no free hands to operate the smartphone.

Furthermore, we provide the user with an Augmented Reality (AR) view of the locations of the nodes. For this the user hold the smartphone screen and views the camera preview as if recording a video. On the live preview camera stream the sensor nodes are draw at the respective locations on the image. This is done by reading the orientation of the smartphone via sensors and the location via GPS. The closer the nodes are, the bigger they appear, becoming smaller with greater distance to the user.

5

Implementation

We implemented our tools for opportunistic deployment support as multiple packages that are used in a single application. Thus, we deploy the final application with a single installation step. We facilitate code re-usability, also for other purposes, through our modular design. For example, our USB-driver to communicate with an XBee module connected via USB to the smartphone is a standalone library. We provide the core functionality of our system through a single library, namely *Daisy*.

DAISY is the development name of our project, which stands for *Distributed And Intelligent Sensornetwork deploYment*. Accordingly, all the code specific to our system resides in the packages with the prefix `de.uvwxy.daisy`. Everything we required, but is not application specific resides in packages with the prefix `de.uvwxy.*`. An overview of the packages is shown in Figure 5.1.

The source code of this thesis will be released under the GNU Affero General Public License on GitHub using the working title *Daisy*:

https://github.com/uvwxy?tab=repositories

5.1 System Overview

Our implementation consists of our tools and libraries, but also uses third party Open Source libraries. The main GUI of the application package is implemented in the package `de.uvwxy.daisy.nodemap`. We consider this as the starting point of our system. Each package that is built on top of, or using an Open Source library is referenced in the packages shown in Figure 5.1.

In the following we explain how we use the most important Open Source components within our work. Then, we explain the structure of our libraries and tools our system is built on.

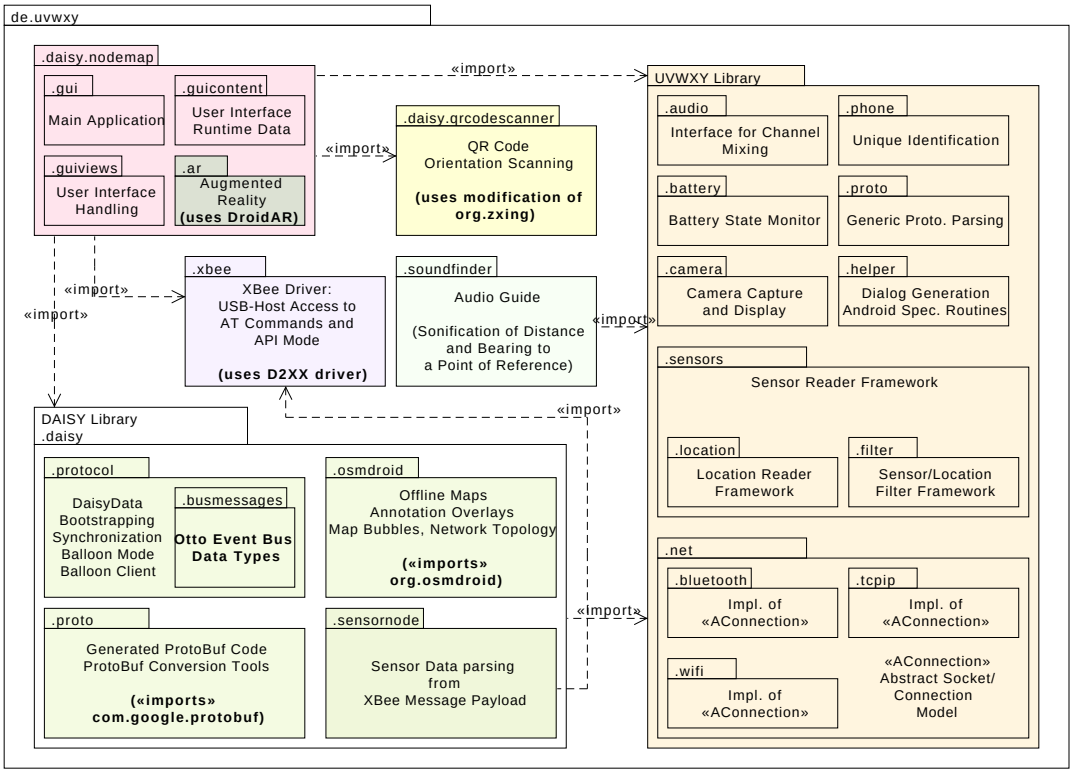

Figure 5.1 System overview of our work. Shown are the Java packages (boxes), and projects. Packages with the same color belong to the same project. The main application is the package `de.uvwxy.daisy.nodemap`. All our functionality is implemented by importing everything into the main application. This way we can independently work on parts of our work, or re-use within other system, e.g. the XBee driver and sensor/location readers.

5.2 Open Source Libraries

In this section we present the most important Open Source libraries that we use in our system. We explain where they are are used and which modifications we had to do to achieve our goals. Other Open Source libraries we used have are omitted at this point as their functionality is not critical for our system. These can be found in the `libs/` folder of the respective projects.

5.2.1 Protocol Buffers

Protocol Buffers (ProtoBuf) is data definition language that helps us in defining the data structures our system uses to store data into a binary file. We exchange the same binary format in our protocols. The code that ProtoBuf generates is forward compatible, i.e., if we extend our data structures and thus protocols. The generated code remains compatible with previous versions of the data definition [23]. Also, older generations of our protocols will work with newer types of objects. Unknown fields and values will be ignored. This way we can extend the system without breaking the implemented protocol on devices running older instances of our system. There are numerous implementations of the ProtoBuf compiler. Thus, we can read and write our binary objects, using other programming languages. This also simplifies porting our code to other platforms. As an example, our system is written in the Java programming language and the data is stored in a binary ProtoBuf file. Now, we can generate the code to read our data for the Python language and evaluate our data on a desktop computer. Any changes we make to the data structures are applied by generating the code for our platforms again. Also, we can use our defined objects in the respective language without the need to repeatedly write boiler plate code. The objects we use are defined in a single `Messages.proto` file. A usage example is shown in Figure 5.2.

5.2.2 OSMdroid and OSMdroid bonuspack

OSMdroid is an Open Source replacement for Android's MapView (v1 API) [37]. It supports a collection of online map tile sources, as well as the possibility to load tiles from offline archives and data bases. We use this to provide online, and thus cached map tiles as our background map layer. On top of this, we add a layer for each offline map found in the `daisy_maps` folder. The user can choose from the map options menu which of these maps to display. On top of these maps we can select which data layer the user wants to display. Initially, all layers are set to be visible: Chat, Sensor Nodes, Network Traffic, Notes and Images, as shown in Figure 4.6, on the right. The viewing area is saved locally in the main deployment file for each user and is copied to the next user on bootstrapping. The preview bubbles that appear when selecting map icons from the overlays is realized with the help of the *OSMdroid Bonuspack*. This library provides general overlay bubbles, routes, as well as reverse geocoding, i.e. an on click listener that returns GPS coordinates at the selected position on the map [36].

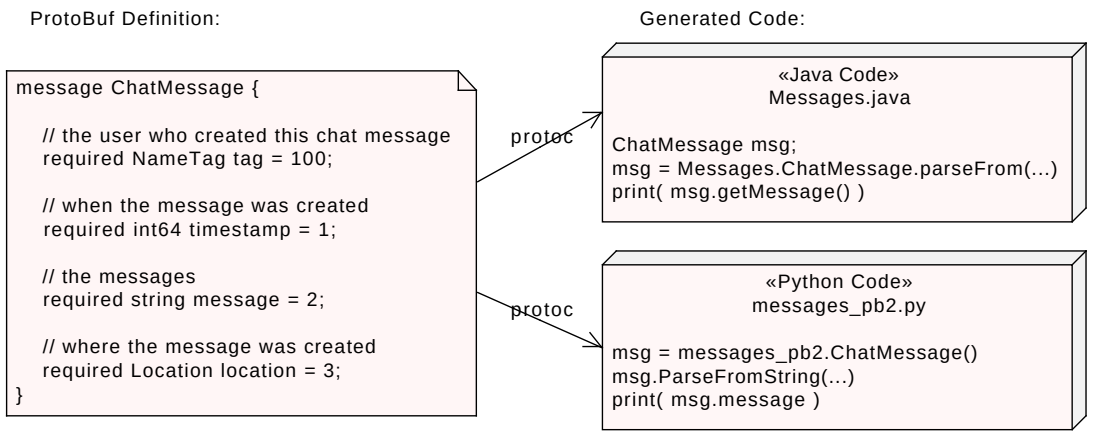

Figure 5.2 A usage example of ProtoBuf. The *protoc* compiler generates code for Java as well as Python and C. Other compilation targets are supported through third party implementations. Protocols built on the exchange of binary ProtoBuf objects remain functional with future versions of the original data definitions [23].

5.2.3 ZXing ("Zebra Crossing")

ZXing is an open-source, multi-format 1D/2D bar-code image processing library implemented in Java with Android support [53]. The ZXing application is available for free in the Android Play Store. Instead of using the installed application, we build the functionality of ZXing directly into our system, as we need to measure the smartphone's orientation during the QR code scanning process. This is the most convenient way, as we are guaranteed that the orientation is measured during the detection and saved as soon as a QR code is detected. Then, we manually calculate the angle of the QR code in the image and return it in correspondence to the angle of the device. The resulting variables are returned with the scan data from the ZXing library. Note, there already is a defined return value for the angle of the scanned QR code, but the code was not completed yet during the development of our system.

5.2.4 Otto Event Bus

Throughout the runtime of our application, we have to deal with data being created at some point either by the user through the graphical user interface, received via synchronizing with other devices, or automatically collected by background threads, such as the logging of the user's location. For a better quality of experience, we want to refresh the newly created/received data on the display, without requiring the user to hit a refresh button. For this, we utilize the *Otto* event bus [44]. Components can register for specific objects, and as soon as such an object is written onto the bus, all registered components receive this object, see Figure 5.4 for our usage overview of the bus. There are multiple components that are interested in the created data. One component is the main data structure that collects and saves all data in a deployment file, thus it registers for all object types that need to be saved. This structure is queried during synchronization, or when displaying data within the application. As

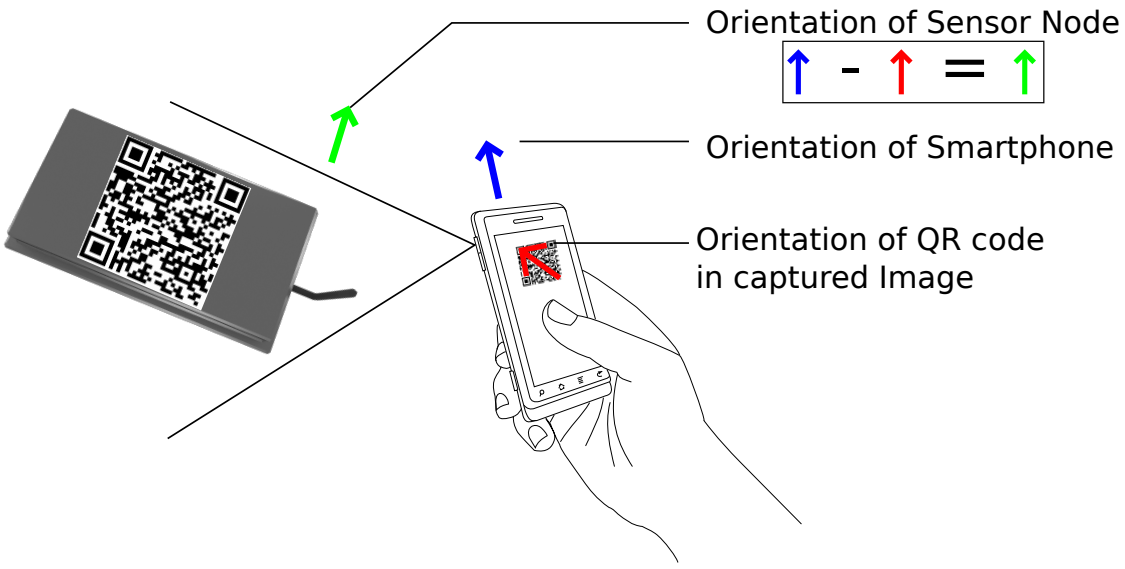

Figure 5.3 Calculation of node orientation during QR code scanning. The scanning process provides us with the deployed node's ID and an orientation of the node. With the orientation of the phone and the orientation of the QR code we can calculate the bearing of the deployed node.

soon as the map shows the current data set we need to update the screen as soon as we receive new objects that can be displayed. For example annotations, or newly deployed sensor nodes. Instead of polling the main data structure we notify the map layers via the *Otto* event bus that we have received a new item. This way the single views of the user interface that require updating of information simply register for the type of data objects they display and automatically receive them to refresh the screen as soon as they appear in the system.

5.3 NodeMap

The starting point of our code is the GUI of the Android application. It is located in the package `de.uvwxy.daisy.nodemap`, c.f. Figure 5.1. Once the user runs the application, she is asked to enter her user name. This name is displayed in the notifications when synchronizing with other participants. Also, the user name is displayed in the chat view or the chat map overlay.

From the main menu the user can select which of the tools to be run during the deployment. The tools are the following:

- **Node Scanner**: The Node Scanner is an Android Activity with which the user collects the information about a sensor node during deployment. There are two mandatory steps the user has to do to add a node to the deployment data structure. The first is to obtain a GPS fix to position the node. The second is to either manually enter an ID of the deployed node, or to acquire this ID from a QR code. Scanning the QR code provides us with the orientation of the sensor node. This orientation can be manually overridden by recording

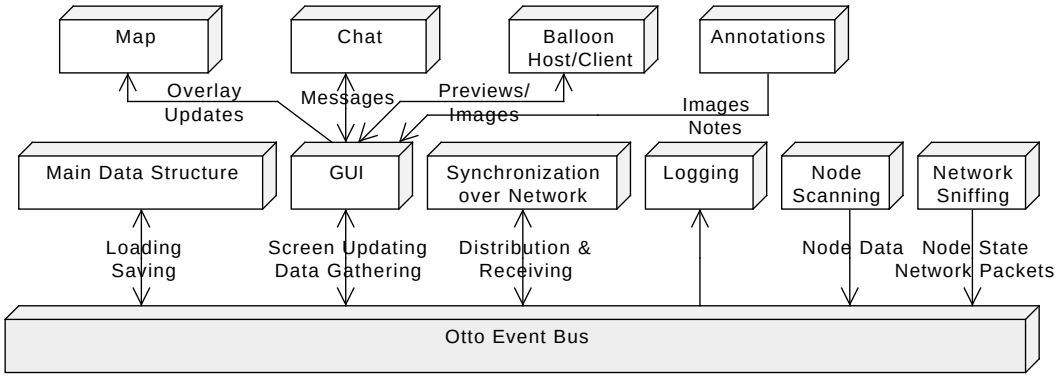

Figure 5.4 We use the Otto Event Bus [44] to distribute generated or received objects between components of our system. If the user generates new data it is saved by the main data structure, displayed on the GUI, triggers network synchronization, or displayed in an event log. Data objects received during synchronization are instantly passed along the message bus in the same way, updating the currently visible GUI views, and is saved in the data structure.

the orientation of the smartphone by aligning it to the sensor node and saving the current orientation of the device. To save the height above ground of the sensor node we calculate the difference from two manual barometric pressure measurements: one at ground level, and one at the height of the sensor node. From the pressure differences the height in meters is calculated.

- **AR Viewer**: To obtain a visual orientation where the sensor nodes are located we use the camera with an Augmented Reality layer using the Open Source library DroidAR [6]. The package `de.uvwxy.daisy.nodemap.ar` contains a factory to create the AR environment, which is launched as a separate Activity.

- **Audio Guide / Sound Finder**: The Audio Guide is a tool to help us find a location with sound. The idea originates from the related work SensorTune, see Section 3.1.5, that uses sonification to facilitate the placement of sensor nodes. The Audio Guide is and Android Intent that takes the following parameters: latitude, longitude, altitude, required accuracy, maximum distance to target, and the destination's radius. With these parameters the launched Activity generates sound patterns that correspond to the distance and the correct bearing of the user. The closer we move to the target, the faster and higher a repeated beeping is heard. When we walk into the wrong direction the beeping is shadowed by static noise. The volume level of the noise shadowing is mixed with a stereo channel. From the difference of the user's bearing and the correct bearing to the destination the noise and beeping is shifted between the left and right channel. This way we can hear into which direction the destination is located, and follow the right direction via audio.

- **Network Sniffer**: This tool uses our XBee driver from the package `de.uvwxy.xbee`, see Section 5.6. With the UI the user can send supported AT commands to the XBee device. This way we can request the RSSI or on which channel

to listen for WSN packets. The received packets are then parsed with the helper classes located in `de.uvwxy.daisy.sensornode`, as they are specific to the sensor nodes we used during our experiments. Different parsing behavior should be implemented and selected from this package.

5.4 The DAISY Library

The *DASIY* library contains the protocols, synchronization and connection management. With this library we can select which protocol to run when a connection to a participant succeeds. These are bootstrapping, synchronizing and the balloon/kite protocol. Furthermore, the library provides the interfaces to generate the data overlays for the map view. Additionally, it handles the background maps. If there are maps in the predefined *daisy_maps* folder, we generate a layer for each map.

5.4.1 Osmdroid Layers

The data layers display an icon for each annotation type, as well as the nodes and network traffic. When selecting an icon a pop-up view displays detailed information. This functionality is provided by the Osmdroid Bonuspack library. We extend its functionality to display different types of pop-up views by extending the `IOverlayExtractor` interface. In the implementation of the interface we define which icon to display for the different overlay types. Also, the interface provides functions to facilitate the extraction of title, text and description that is shown on the overlay bubble. Some of the overlay bubbles contain images. For this we adapted the implementation of the original `NodeBubble` class from the Bonuspack library. The modified class then checks what type of object is associated with the overlay icon, and changes the content of the bubble accordingly.

5.4.2 Daisy Protocol

The protocols behind bootstrapping, data synchronization, and the connection between the balloon/kite device and the remote image viewer is located in the `de.uvwxy.daisy.protocol` package. This is also where the main data structure is managed. The name of this class is `DaisyData`. This data structure keeps track of the sent, received and created data objects throughout the use of our system. The DaisyData object provides access to the stored ProtoBuf objects, as well as access to data needed by protocol messages, or to save and load the whole deployment data to a binary file. The contents of this this structure are received via the event bus. To add objects into this storage during runtime we need to write the objects onto the bus. This way they are stored and, possibly shown on the screen.

During the bootstrapping process the new participant obtains a clone of the main data structure of the communication partner. As we initially connect without any data in our own local data store, we obtain our initial data structure from this clone. Within this clone we have the ID and name of the deployment, to save it onto the phones memory.

The synchronization protocol works on the principle of exchanging the largest known sequence number of each known participant. This way each device can check which of these sequence numbers are missing among it's own data. In the next step, the device requests the missing sequence numbers. A *DataReply* answers this request, containing the missing objects. An example of the synchronization algorithm is shown in Figure 5.5.

The balloon protocol works similar to the synchronization protocol. Upon connecting, the client verifies that the other side is currently operating in balloon mode. The balloon protocol works on top of the other protocols. This way the flying device also answers to synchronization requests. A positive response from a device in balloon mode contains the available resolutions. Now, we can send control messages to set the delay between the captured images, the resolution and request image previews. This is either by requesting the latest preview, or all previews that are available so far. As soon as the previews are received, they are shown on the client's screen. Then, we can either open the preview in an image viewer, or request the original image. As soon as this is received the user is given the option to view the original resolution of the image. After disconnection, both devices continue to respond to synchronization requests.

5.5 The UVWXY Library

The *UVWXY* library simplifies the available Android API's. This way we can avoid writing boiler plate code, i.e., duplicate code that we need repeatedly with only little modifications.

5.5.1 Abstract Connection Layer

We implemented an abstraction layer on top of the existing TCP/IP sockets and Bluetooth sockets. This way we can handle Bluetooth and TCP/IP connections in the same way when working within the DAISY project. We provide three implementations of our abstract connection model, for Bluetooth, Wi-Fi Direct and TCP/IP. This way we have a unified way of discovering devices for all three connection types, see Figure 5.6. When a device is discovered using Bluetooth, Wi-Fi Direct, or TCP/IP, in the latter case DNS Service Discovery, we obtain an abstract `Peer` object with an address of the found device. As the peer object also has a type indicating what kind of address it provides, i.e., Bluetooth, Wi-Fi Direct, or TCP/IP, we can automatically switch to the matching implementation of our abstract connection.

5.6 XBee Driver

The XBee driver is based on the FTDI D2XX driver for Android [20]. This driver provides access to serial devices connected to an FTDI serial-to-USB chip. Thus, we obtain binary input and output stream through which we can communicate with a

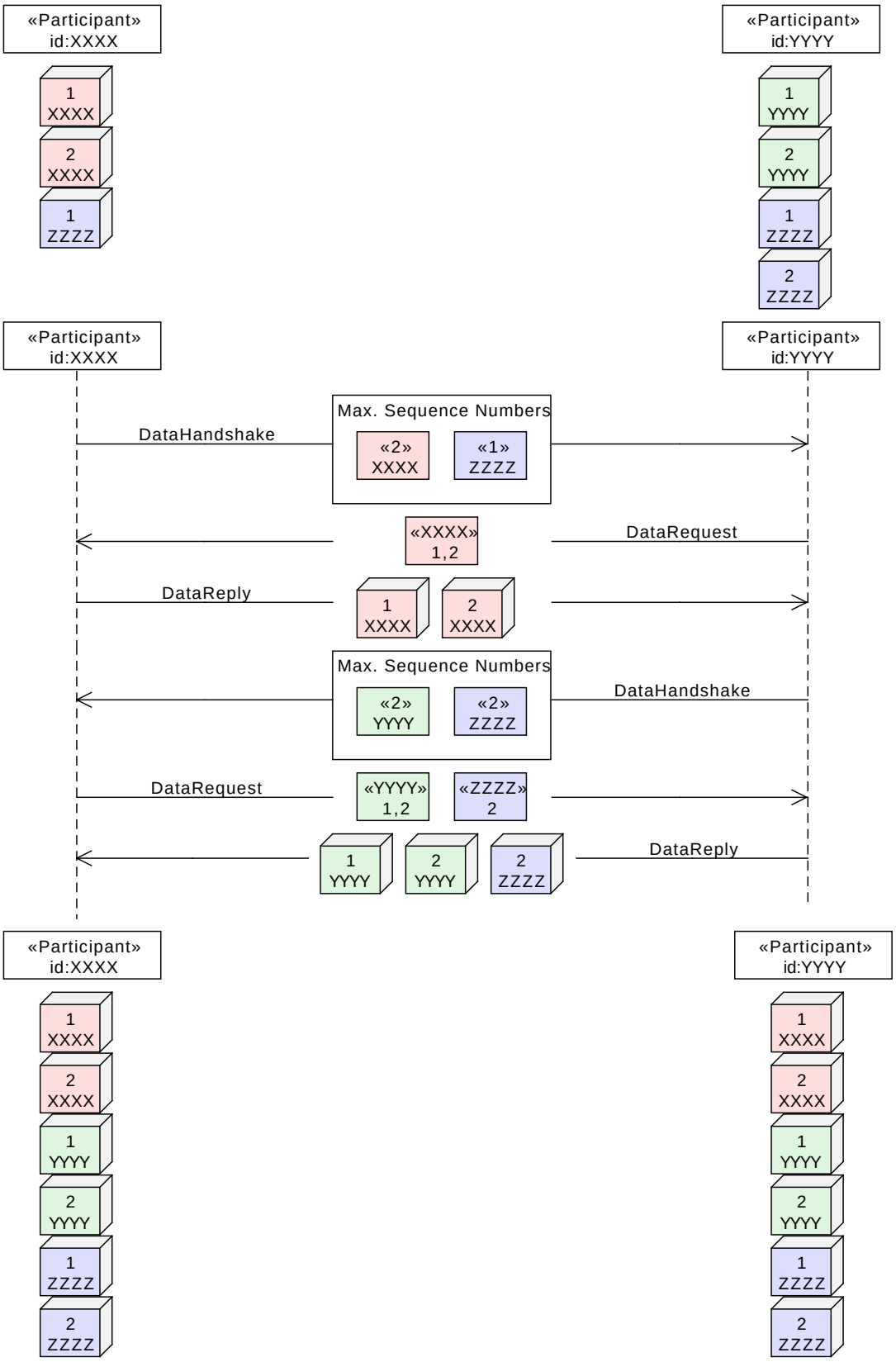

Figure 5.5 Before, during and after data synchronization. Each participants appends an ascending sequence number to each created data object. On synchronization the maximum sequence numbers of all owner ids are exchanged. With this each participant can lookup which data entries are missing and requests the missing data objects.

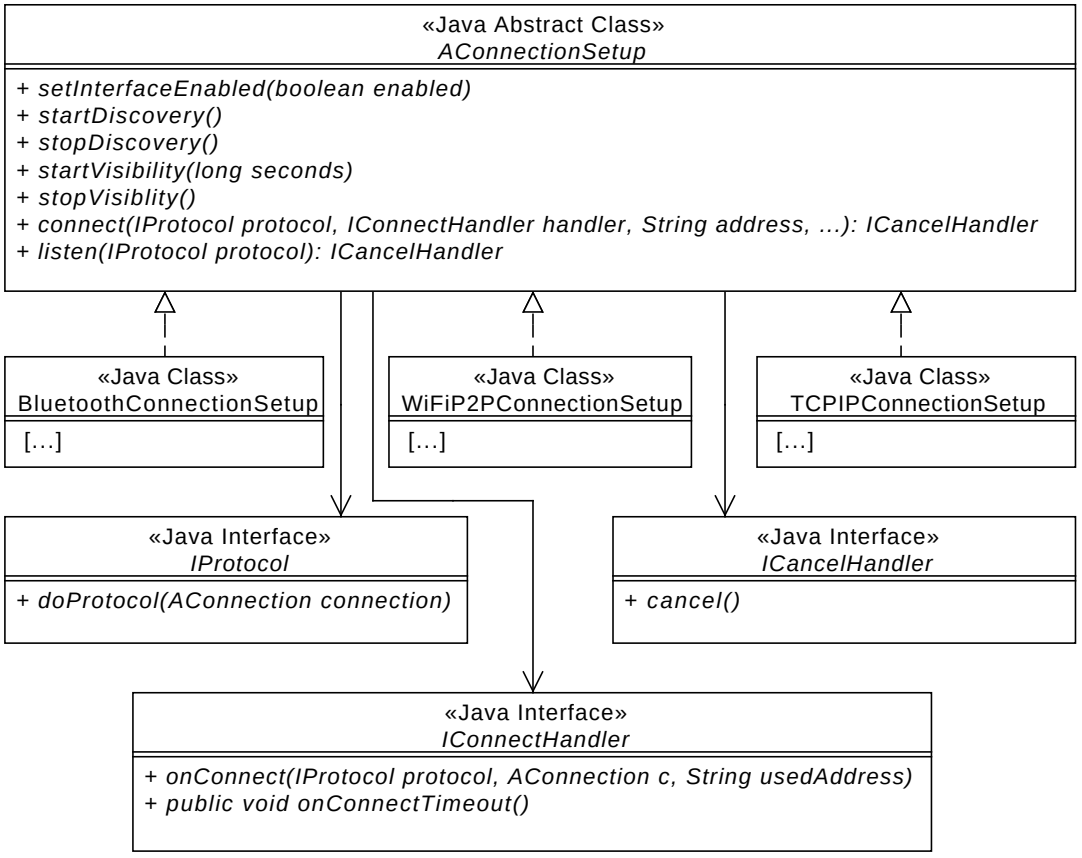

Figure 5.6 Overview of the connection setup abstraction. We switch between the respective implementations of the *AConnectionSetup* class to establish links with the different connection types. To achieve the same protocol behavior on these links, we provide the protocol to be execute on connection success, c.f. *IConnectHandler*. Protocols are implemented by performing their actions within the *doProtocol* function. The provided *AConnection* provides us the input/output streams of the underlying socket via *getIn()* and *getOut()*.

serial device connected via USB. Note, in our implementation this is only possible when the smartphone is USB-host capable.

Our driver provides a simple way of connecting to an attached XBee device. The following are the implemented functions of our `XBeeDevice_UART` implementation of the `XBeeDevice` interface:

- `bindToFirst()`: if there are multiple devices open an arbitrary device available.
- `boolean init()`: initialize the device, returns true if successful
- `boolean isOpen()`: check whether this instance is connected to an XBee device
- `Frame getNextAPIMessage()`: pop the next received API message from the driver's message buffer
- `setAPIMode2()`: enable API mode 2
- `setCommandMode()`: set the XBee into command mode
- `sendCommand(ATCommand atc, String params)`: send the given AT command with optional parameters
- `sendAPICommand(ATCommand atc, byte[] params)`: send the given AT command with optional parameters via API mode
- `sendAPIMessage(APIMessage apm)`: send the given APIMessage
- `close()`: close the connection to the XBee device

We access the XBee device through an implementation of the `XBeeDevice` interface. This way we can exchange the way of accessing the XBee device by providing different implementations. Another possibility is the USB Accessory mode, in connection with an XBee device connected to an Arduino ADK.

5.6.1 Parsing WSN Messages

To obtain the WSN's topology and sensor data we parse the messages received by the XBee module. The XBee driver uses the `FrameCallback` to return API messages. These are either returned when sending API commands to the module, or when messages are received. When the `FrameCallback` is triggered with an `APIMessage`, we post the contained onto the event bus. This way we can display debug information about API messages responses. Each API message is then translated to a `NodeCommunicationData` object. This is done within the `SensorNetworkMessageParser`, which contains all functions to convert an `APIMessage` that contains a WSN packet into an object of the type `NodeCommunicationData`.

The payload of our WSN packet is located after the TinyOS and CTP headers. The structure of the payload is shown in Figure 5.7. From the `nodeId` and `parentNodeId` fields we construct the network topology, that is displayed on the map. The payload also contains the sensor values of the node. After the parsing of the WSN packet, the converted object is passed onto the event bus. In this case the event bus facilitates the development of additional plug-ins using data from WSN messages. Any plug-in that requires these messages registers for objects of the type `NodeCommuincationData`. As soon as they are received by the XBee module, they are sent to all respective event bus subscribers.

```
typedef nx_struct SensorsDataCollectionMsg
{

  nx_int16_t nodeId;
  nx_int16_t parentNodeId;

  nx_uint16_t etx;
  nx_uint32_t time;
  nx_uint16_t batteryVoltage;

  nx_int16_t accelerationX;
  nx_int16_t accelerationY;
  nx_int16_t accelerationZ;
  nx_int16_t accelerationTemperature;
  nx_int32_t inclinationX;
  nx_int32_t inclinationY;

} SensorsDataCollectionMsg;
```

Figure 5.7 Payload structure of a WSN sensor data packet.

5.7 Summary

The implementation of *Daisy* focuses on portability, extensibility and re-usability of code. With the integration of Google Protocol Buffers, we do not need to manually write the code to serialize objects to save the deployment data, or within our protocols. As ProtoBuf compiles code for C, Python and Java, we can easily port our protocols and data structures to other systems. The definition of the objects exchanged within our protocols is given in the *Messages.proto* file. The Otto Event Bus, simplifies the exchange of information between protocols, user interface, and storing of data. The development of plug-ins is facilitated by this architecture; data updates are obtained via the event bus, without any knowledge about the underlying infrastructure. The visualization of annotations, nodes and network topology are realized with the *OSMDroid*, and *OSMdroid Bonuspack* libraries.

6

Evaluation

In this chapter we present the results of our work. We compare how our system improves sensor network deployment to our previous experiences. We discussed initial ideas for our work after a test deployment to measure the communication ranges of X-SLEWS sensor nodes. Although we consider the setup of the test deployment as simple, i.e. place 4 sensor nodes in a straight line along a road, it took us 3:30 hours to complete the deployment.

We presented our work at The Extreme Conference on Communication (ExtremeCom), and surveyed the participants regarding our system requirements. We deployed a WSN and used our system to gather and display information about the current status of the sensor network. Additionally, we created a map of the area using the tools presented in this thesis.

At the end of this chapter we discuss our results and highlight areas that require future improvement.

6.1 Pre-Deployment: Range Measurements

On the 5th of March 2013, we deployed a WSN to establish the communication range of the X-SLEWS sensor nodes developed by Matthias May [31].

6.1.1 Initial Survey

The deployment site is close to Orsbach, 5km north-west of Aachen. Along the "Schlangenweg" road, there is an 1.5km stretch of open space suitable for line of sight range measurements. Due to these observation, we considered it likely to keep track of participants during the deployment, e.g. where they were moving to deploy the nodes, by visual contact. An overview of the setup of network, is shown in Figure 6.1.

Figure 6.1 Overview of the initial phase of the range test. The nodes were placed along a road at increasing distances to the base station. After arriving at the site at 11:43 AM the base station was deployed at 11:52 AM. The following nodes were placed along the road in the following order with increasing distance:

- Node `101`: Timestamp: 01:14 PM, Location: 50.7909896 / 6.0122469
- Node `102`: Timestamp: 12:49 PM, Location: 50.7915931 / 6.0100736
- Node `104`: Timestamp: 12:56 PM, Location: 50.7922788 / 6.0078571

The data for this Figure was created from the timestamps and GPS tags of the pictures of the nodes we took during the deployment. Note, the images where not taken in the order the nodes were deployed.

6.1.2 Local conditions

Unfortunately, the GSM coverage of this area was not sufficient. We were not able to establish phone calls or mobile data connections along this road. This turned out to be one of the first obstacles during the deployment. At the day of the deployment the weather conditions were optimal. There was no overcast, and no wind, c.f. Figure 6.2.

Network Monitoring: After setting up the base station and connecting a laptop for debugging purposes, we realized that due to the good weather the screen was hardly readable in direct sunlight. Thus, the base station was moved closer to a line of trees, see Figure 6.1 in the bottom right corner. In the shade it was easier to read the display to monitor the state of the network.

6.1.3 Communication

As the laptop has to be connected to the base station, we assigned one person to be in charge of signaling the others if a node was received at the base station. We drove along the road, to slowly increase the distance of the first node to be deployed. We agreed on two waving patterns to establish whether to drive further to increase the distance or to stop and deploy the node. This ended up to be more difficult then we expected. After about 100 m we could not see what the person was waving against the contrast of the dark background of trees and the sun. Thus, we decided to drive to a distance of about 300 m, ignoring the person waving close to the trees, and deployed the first node (id 101). In the meantime the driver reversed back to the base station to check whether the node was communicating with the base station. As we could not establish over the distance what the person at the base station is signaling, we agreed on setting up the remaining nodes at equals distances between each other. This was facilitated by placing the sensor nodes at base of cement pillars lining the road in regular distances. This process was repeated until we established a single hop communication distance of 720 m.

6.1.4 Data Gathering

We agreed on noting the deployment times as well as locations, by capturing images including GPS tags with our smartphones. This data was later exchanged via e-mail in the days following the deployment. We noticed that one of the smartphones was set to a resolution of 640x480 pixels, making it hard to identify the nodes in the images after the deployment, see Figure 6.2. We approximated the distances during the deployment with the help of the mileage indicated by the car's tachometer. Initially we estimated the maximum communication distance to be 800 m. Later, after measuring using the GPS locations of the nodes, we established the longest link to be 720 m. The last node (id 103), was placed the furthest along the road.

Although the deployment seemed to be of a simple nature in theory, practically it took us 3.5 hours until we established a maximum communication distance on that day. Our undertaking took longer than initially planned. The following is a list of problems that we encountered during the deployment:

- No GSM
- Failure of communication, improvised "waving" ineffective despite open space
- Coordination of data gathering: node location and time unknown during deployment
- Misconfiguration of camera resolution
- No visible network topology

6.2 Survey on Requirements

During ExtremeCom we distributed a questionnaire among the participants. We asked them to rate each item from a list of requirements from "not at all helpful" to

Figure 6.2 During the range tests we recorded the deployment times and locations with our smartphones. Geo-tagged images were exchanged via e-mail in the days following the deployment. From left to right the images show the locations of the nodes 101, 102, and 104. The two fingers in the second image are an improvised annotation to note that this is the node with the id 10**2**. The original resolution of the images is 640x480 pixels, making it difficult to read the node ids on the cases.

"very helpful" regarding WSN deployment support. Out of a total of 19 participants 15 took part in our survey. Their background was delay tolerant networks, wireless sensor networks, and communication systems. Of those that answered the questionnaire, 5 had previously participated in other sensor network deployments. Their rating of the requirements is shown in Figure 6.3. As we can see, the emphasis of importance of their replies lies in the process of gathering node related information. Also, the display of maps with network and annotation information, together with the possibility to create aerial images, and additionally to be able to provide audio guidance to find nodes or places from the map are rated next to "very helpful" (4 out of 5 points, median rating). In average, opportunistic communication is rated exactly between "not at all" and "very" helpful. Augmented Reality is voted the least helpful with 2 out of 5 points. In contrast to the rating of all participants, the group experienced participants did not consider the augmented reality viewer, the audio guide and the retrieval of aerial images as only slightly important.

The results of the questionnaire confirm the requirements we established during the design of our system. Both, the process of gathering node related information as well as the display offline/online maps, including the network topology and map annotations are rated as important to very important for deployment support for wireless sensor networks.

6.3 Demo Deployment at ExtremeCom 2013

At the ExtremeCom 2013 we presented our system. Starting at 16:00h we had an hour to demonstrate our system. For a WSN deployment we took the same components used in the sensor network during the range tests. Thus, our network consisted of the base station, located inside a hut, and two additional sensor nodes.

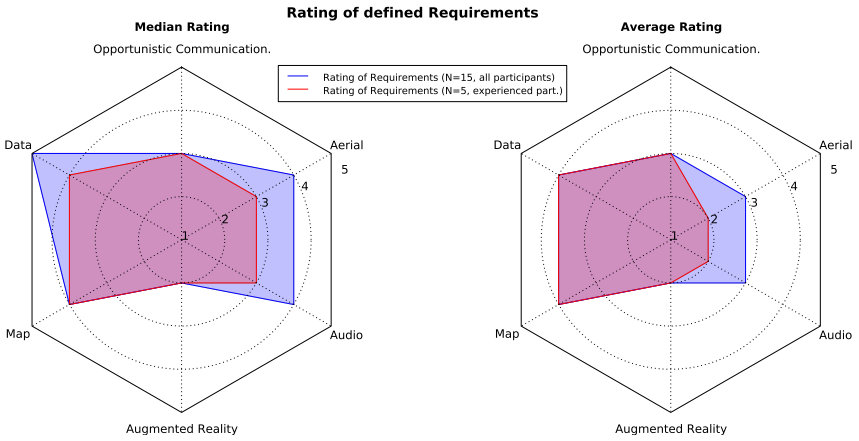

Figure 6.3 Evaluation of the requirements present in general. $N = 15$ represents all answers from the returned questionnaires, of which $N = 5$ have previous experiences with WSNs. Shown is the rating of the general helpfulness of the requirements. The Rating is given in 5 steps from "not at all"(1) helpful to "very"(5) helpful.The rated requirements are the following:

- *Opportunistic Communication*: Op. Com.
- *Process of gathering node related information*: Data
- *Map display (i.e. offline maps, display of network topology, messages, annotations)*: Map
- *Augmented Reality*: AR
- *Audio Guidance*: Audio
- *Balloon Mode / Balloon Client*: Aerial

The abbreviations given in the brackets are used for identification in the figures above.

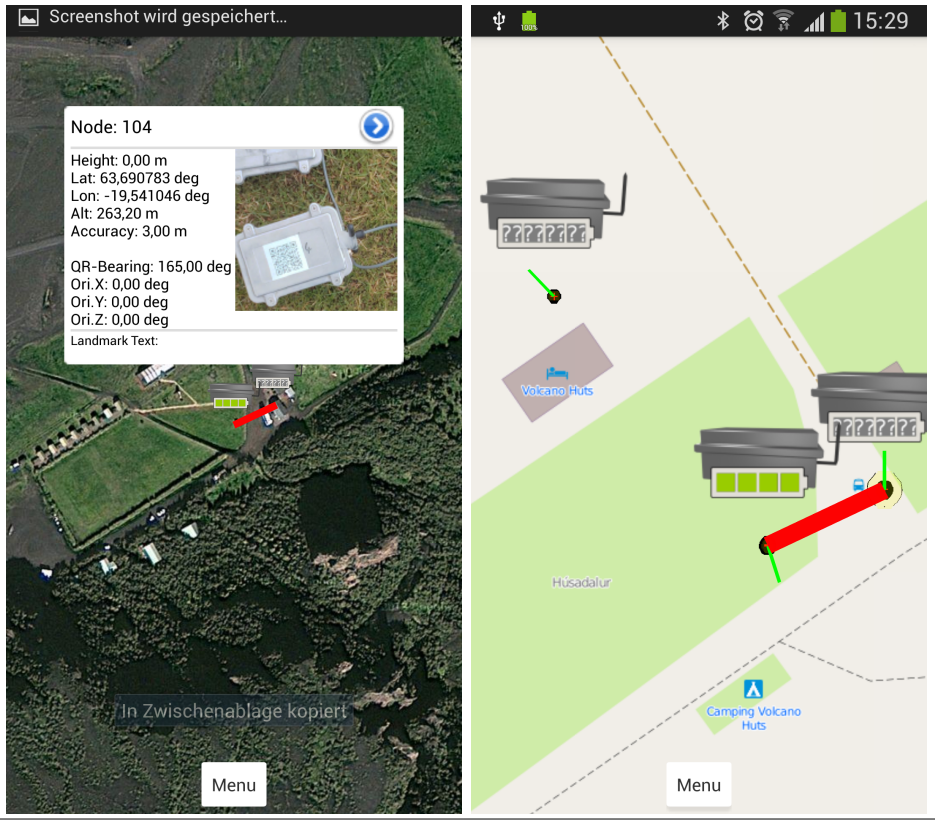

Figure 6.4 On the left we see the offline map using tiles from Google Maps. On the right we see the location of nodes and orientation of sensor nodes. The location is indicated with a red plus, surrounded by a small yellow circle indicating the accuracy of the location. The orientation is shown with a green line pointing away from the location marker. The battery state of node 104 is indicated as being full, as it is reporting 3.6 Volts.

Network Monitoring: Outside of the hut we placed an Android tablet on which users could watch the deployment's progress. To this tablet we connected an XBee to monitor the traffic of the sensor network. Figure 6.5 shows the network events as well as the deployment times of the nodes. On this device the participants could watch the progress of the deployment with the map of the area.

Offline Maps: We created an offline map archive of the local area around the deployment site as a base map. The archive was created with the *Mobile Atlas Creator* [32] using Google Maps satellite imagery as the source. This was then uploaded to our demonstration device. A screenshot displaying this map is shown in Figure 6.4.

6.3.1 Data Gathering/Monitoring

One person then deployed two sensor notes at distances of 30 m (node 104) and 86 m (node 101). The first node to be placed was the node with the id 104 at 16:25:41. Then the user came back and added the location information of the base station at 16:31:32 and enabled the network sniffer. From this we can see that at 16:32:33 node 104 is already sending data to the base station, see Figure 6.5. This means, that after 7 minutes we could verify that the placed node was integrated in to the

network. After, this the last node (id 101) was deployed at 16:37:48, see Figure 6.6 for the location of nodes and the GPS track of the user deploying the devices. The bearings of the deployed are 165 degrees for node 104, and 315 degrees for node 101, c.f. Figure.

6.3.2 Aerial Maps

During the deployment we also launched a smartphone attached to a kite to capture aerial images. The construction was improvised within the first 10 minutes of the demo. The smartphone was attached 30 cm below the kite with a single line, wrapped in a plastic bag. For the camera we cut out a hole into the bag.

The total flight time was 32 minutes, starting at 16:17h. At 16:32h someone not knowing the smartphone was used during a demo, called the number of the device. This caused our system to crash at 16:32h. We noticed this at 16:46h as we lowered the kite down to check our system, because we could not establish a connection to the kite anymore. We restarted our software and launched the kite at another location again. During the initial phase we were able to download the previews from the kite, and request the original images on demand.

In total, 306 images where captured during the demo. Among these images, 150 are black. These images were captured in a jacket pocket after the smartphone was taken from the kite, after 40 minutes. From the other 156 images, 25 where sharp enough to be useful for map creation. Again, from them only 10 images were used for map stitching with MapKnitter, due to overlapping regions and/or to high angles.

The average resolution of the aerial images used during the stitching process lies between 3.0 and 30.0 cm / pixel. The average is at resolution of the final map lies at 11.0 cm/pixel. This resolution lies in the high resolution range of Google Maps (10 - 15 cm/pixel) [11], see Figure 6.9.

With each image we capture the following information:

- GPS: latitude, longitude, altitude, accuracy, speed, bearing, location provider, time
- Accelerometer: three axes in $\frac{m}{s^2}$
- Compass: azimuth, pitch, roll in degrees
- Barometer: pressure in millibar Hg
- Battery charge in percent
- Remaining free memory in bytes

The data recorded throughout the demo session are shown in figs. 6.10 to 6.16.

6.4 Survey Results of our System

In addition to the rating of the requirements, we asked the participants to evaluate our work. In the second part of the questionnaire the participants answered how our our system fulfilled the list of requirements. The answers are given with points

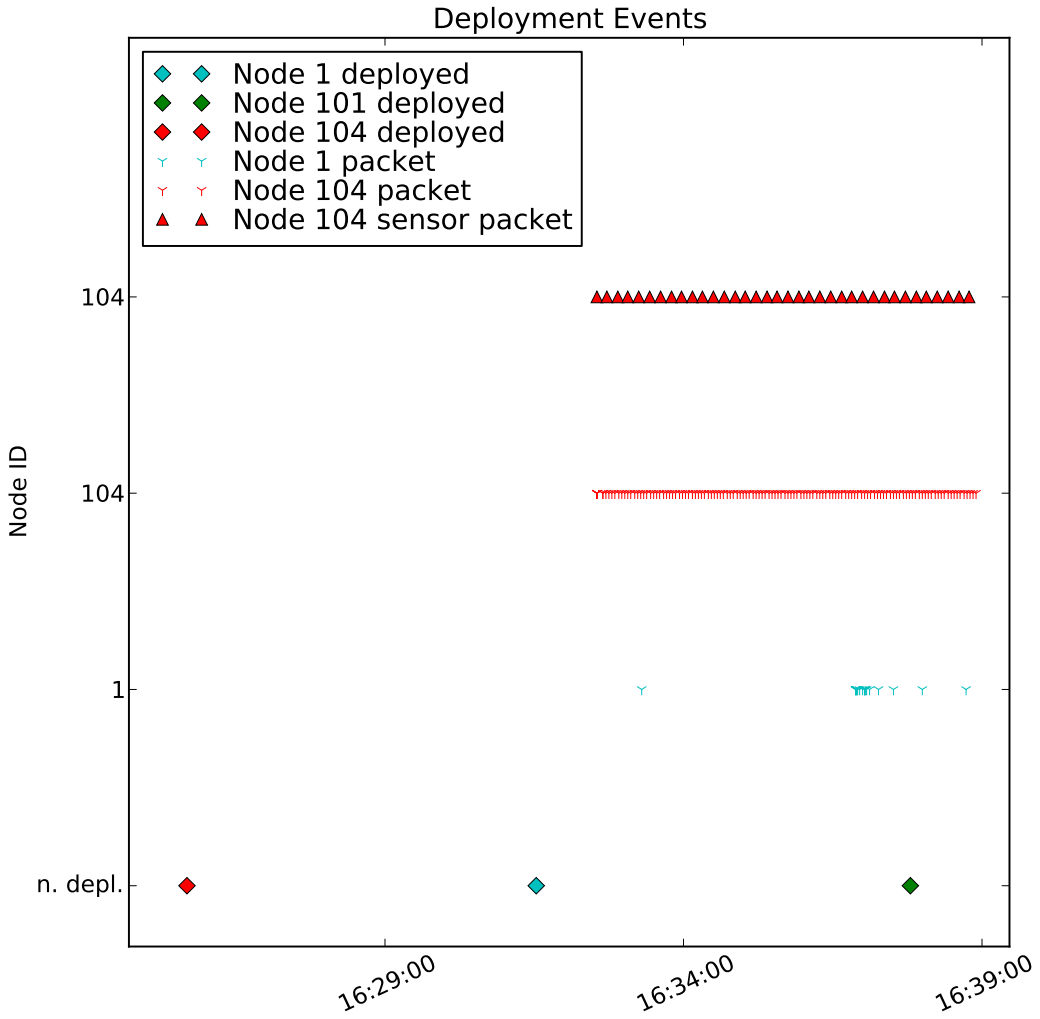

Figure 6.5 Time line of events of the demo deployment at the ExtremeCom 2013. The first node to be deployed was node `104` at 16:25. After that the location of the base station was added at 16:31 although it was already running. At 16:37 the second sensor node `101` was added. We can see the regular intervals of sensor data sent to the base station every 10 seconds beginning at 16:32.

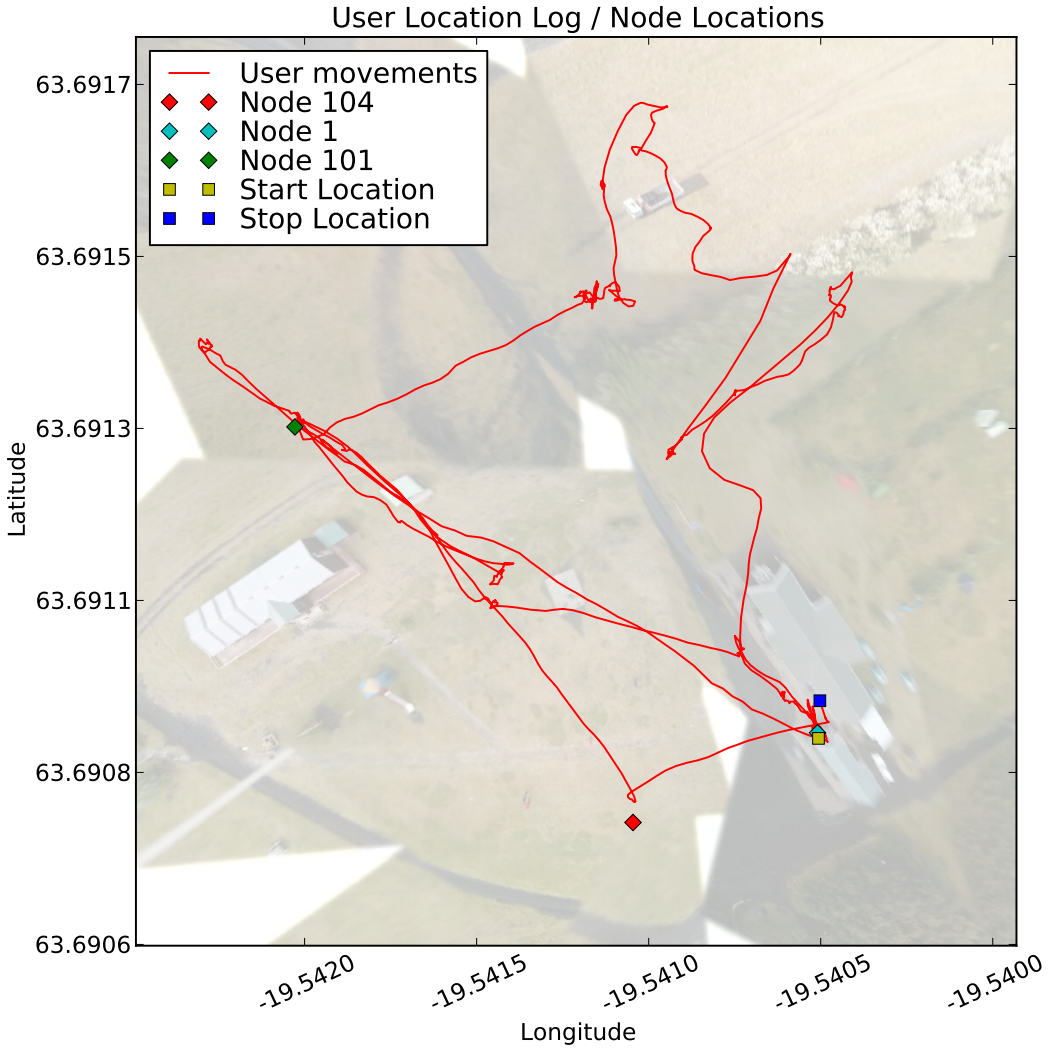

Figure 6.6 GPS Location track of the deploying user during the demo deployment at the ExtremeCom 2013.

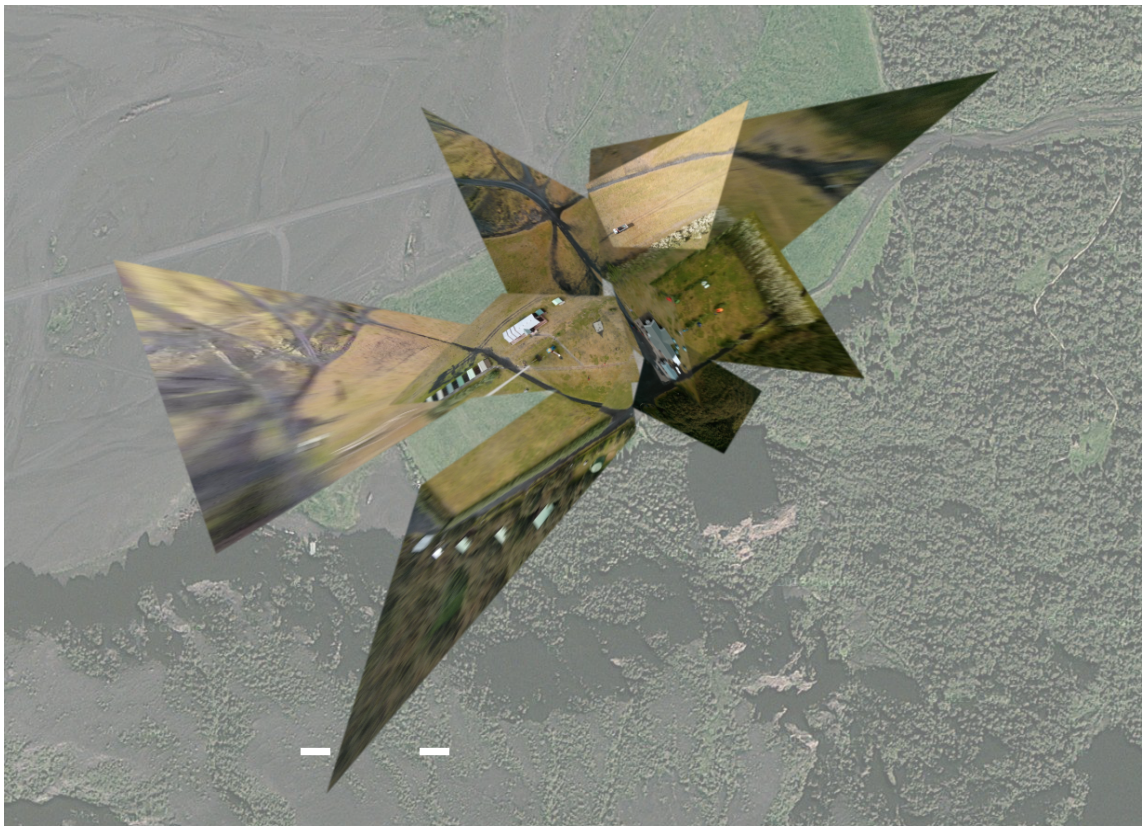

Figure 6.7 Coverage of the rendered map created with aerial images taken by a smartphone attached to a kite.

Figure 6.8 Comparison of the created map from aerial images (left) and satellite images provided by Google Maps (right).

Figure 6.9 The average resolution of this image is 3.7 cm/pixel. The kite was flying at a height of about 32 meters (precision 3 m). In comparison, Google Maps provides resolutions of up to 10 - 15 cm/pixel [11, p. 266].

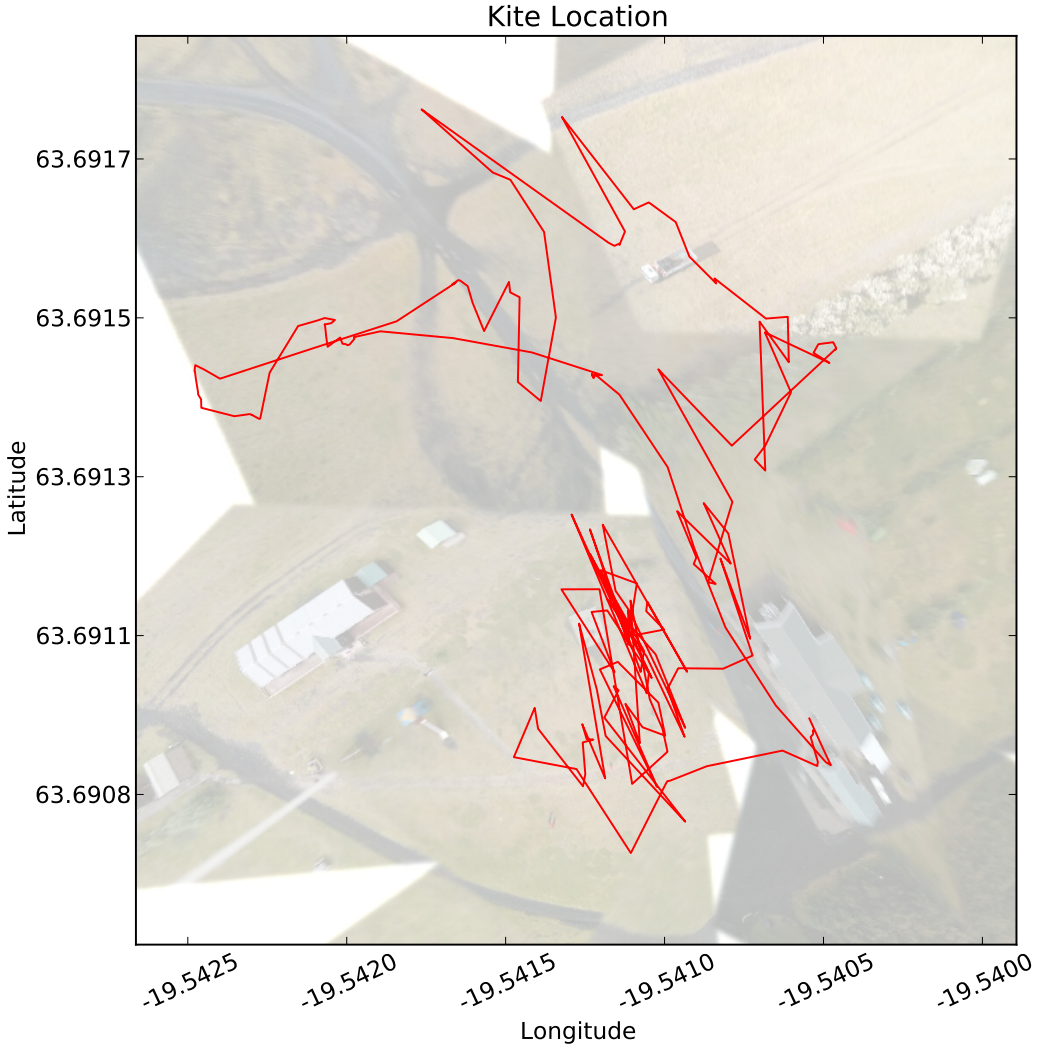

Figure 6.10 Location track of the kite during the demo session. The background map in this figure is created with the images captured by the kite during this flight. The start is to the left of the bottom right hut, in the lower center of the map. After 35 minutes the kite was carried further north and launched closer to the truck. The location tracking is done via GPS.

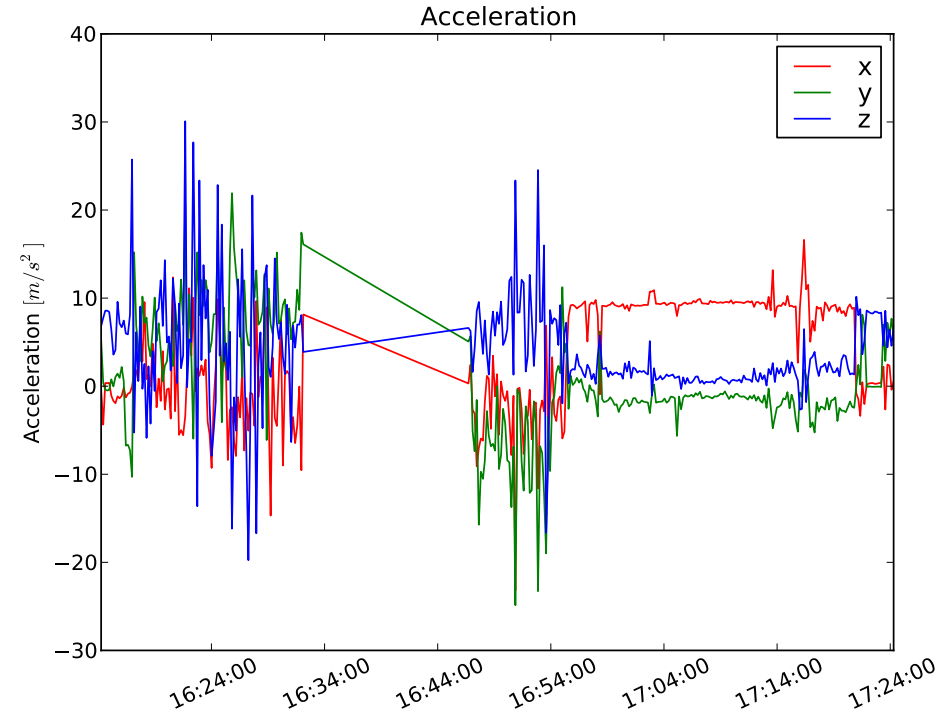

Figure 6.11 Acceleration log of the kite during the demo session. With each captured image our system saves the accelerometer values of the smartphone. At 16:32h our system crashed due to an unhandled voice call. At 16:47h the kite was checked and the software restarted.

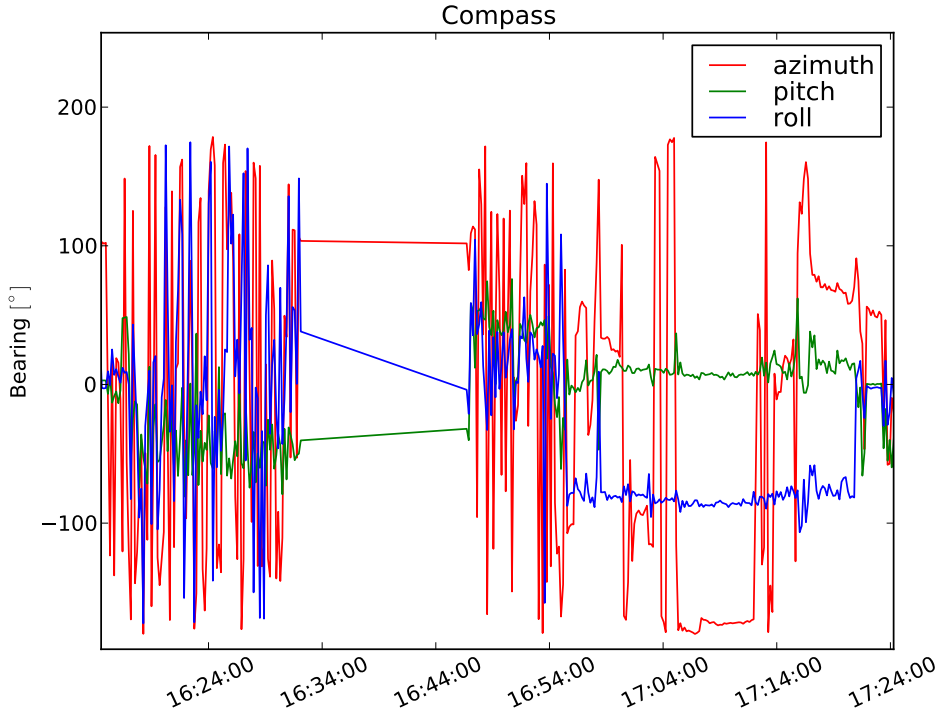

Figure 6.12 Compass log of the kite during the demo session. With each captured image our system saves the compass orientation of the smartphone. At 16:32h our system crashed due to an unhandled voice call. At 16:47h the kite was checked and the software restarted.

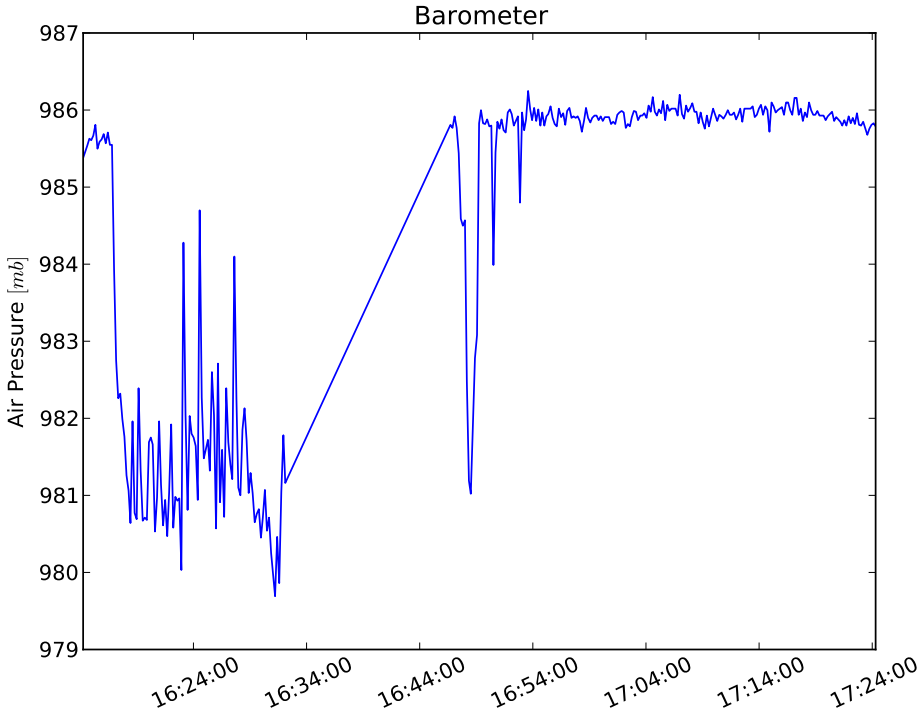

Figure 6.13 Barometer log of the kite during the demo session. With each captured image our system saves the barometer value of the smartphone. At 16:32h our system crashed due to an unhandled voice call. At 16:47h the kite was checked and the software restarted.

from 1 to 5, i.e. between "poor quality" and "excellent quality", Figure 6.17 shows the average and median ratings of the participants, in comparison to the initial requirement ratings. The median quality rating of all features lies at 4 points. The average results differ slightly, with the kite/balloon mode, audio guide, and the AR view scoring 3 points. The results show, that in average our system scores better in the process of node data gathering, displaying of information on the map, and opportunistic communication.

During the demo we demonstrated the system to the participants of the conference. This included the live previews of the kite images, as well as the live sensor data received from node 104. Prior to the conference our application was distributed with the other demo applications among the participants. From those, only 3 installed the application on their device.

6.5 Discussion

In this section we will discuss the results of our work. We divided the related work we presented into two categories: surveying and map making tools. This is due to our work being closer in line to the former, as it is to the latter. Still, our results show that we provide valuable data that can be used by map making tools.

Surveying: The approaches that are in closest relation to our system are *iButton Assist*, *idNotebook* and the *GSN Project*. All three's goal is to collect information

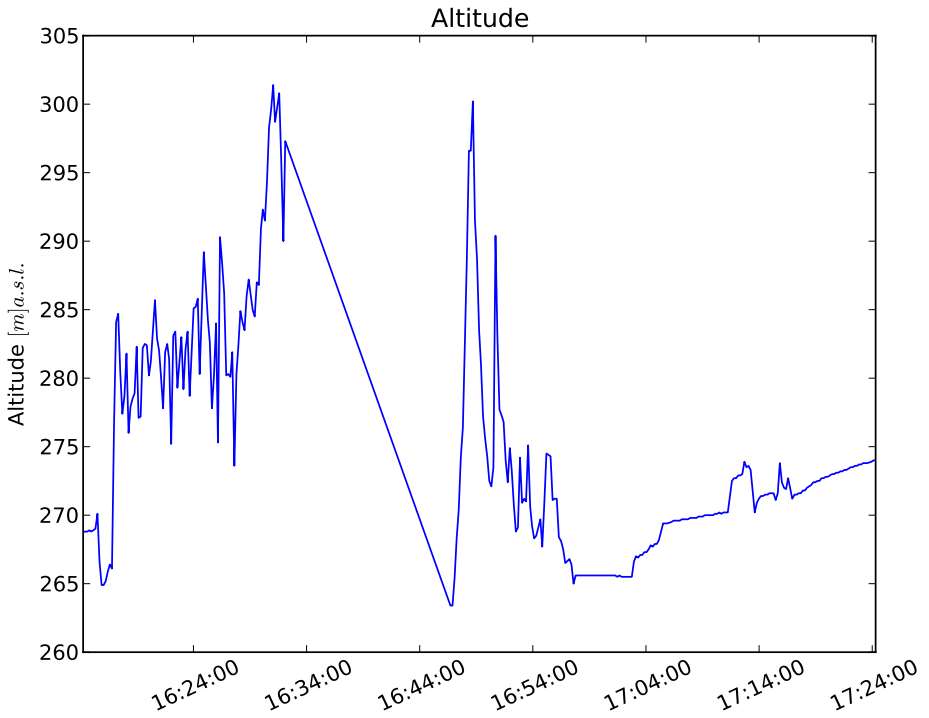

Figure 6.14 Altitude log of the kite during the demo session. With each captured image our system saves the altitude a.s.l. of the smartphone. This is provided by the GPS receiver. At 16:32h our system crashed due to an unhandled voice call. At 16:47h the kite was checked and the software restarted.

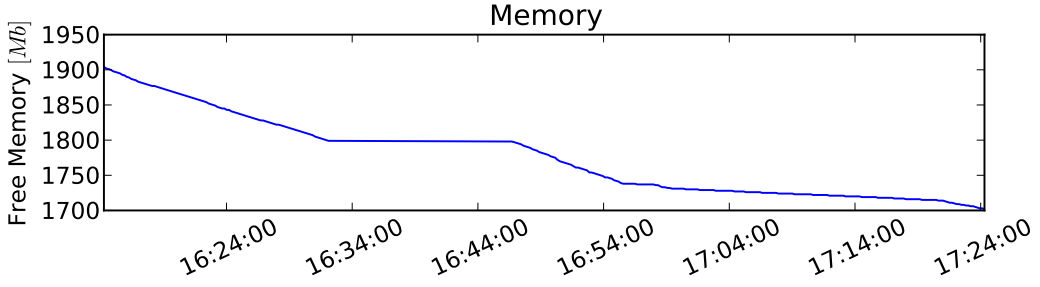

Figure 6.15 Decreasing of free memory during the demo session. At 16:32h our system crashed due to an unhandled voice call. At 16:47h the kite was checked and the software restarted.

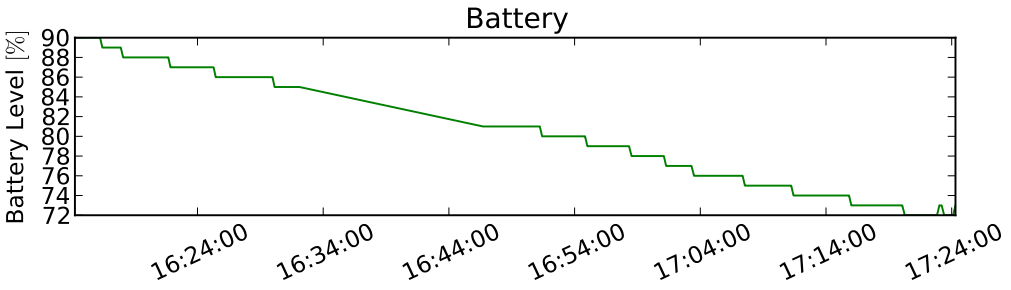

Figure 6.16 Battery log of the smartphone during balloon mode. After 70 minutes of operation 20% of the total capacity was used. This gives us an up time of about 5h50 on a fully charged battery. We used a Google Nexus 4 during this test. At 16:32h our system crashed due to an unhandled voice call. At 16:47h the kite was checked and the software restarted.

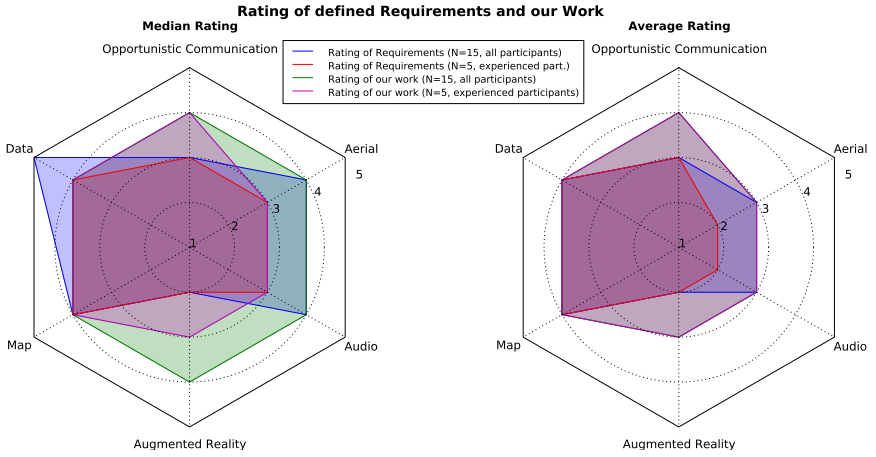

Figure 6.17 Evaluation of the requirements, and the realization of the requirements in our system. $N = 15$ represents all answers from the returned questionnaires, of which $N = 5$ have previous experiences with WSNs. Shown is the rating of the general helpfulness of the requirements and the rating of our system. The Rating is given in 5 steps form "not at all"(1) helpful to "very"(5) helpful for the general consideration of the features and the quality of our work rated from "poor"(1) to "excellent"(5) regarding those features. The rated requirements are the following:

- *Opportunistic Communication*: Op. Com.
- *Process of gathering node related information*: Data
- *Map display (i.e. offline maps, display of network topology, messages, annotations)*: Map
- *Augmented Reality*: AR
- *Audio Guidance*: Audio
- *Balloon Mode / Balloon Client*: Aerial

The abbreviations given in the brackets are used for identification in the figures above.

	Online Maps	Offline Maps	Annotations	Collaboration Support
Daisy	+	+ Balloon Mode	+ Text/Voice/Image	+ Opp. Com.
Geopaparazzi	+	+	+ Text/Voice/Img./Sketch	−
OSMTracker	+	−	○ Predefiend OSM Entities	−
iButton Assist	−	−	−	−
idNotebook	−	−	−	−
GSN Project	−	−	−	−
SensorTune	−	−	−	−

	Data Export	Network Monitoring	Node Data Collection
Daisy	+	+ Sensor Data / Topology	+ ID/Loc./Height/Orient./Text/Img.
Geopaparazzi	+	−	−
OSMTracker	+	−	−
iButton Assist	+	○ Directly from Device	+ ID/Loc./Sensor Data/Img.
idNotebook	+	−	+ Manual: ID/Loc.
GSN Project	+	+ Display of uploaded Data	−
SensorTune	−	−	−

Figure 6.18 Comparison table of related sensor network deployment and map annotation tools. Our work (Daisy), is the only approach to provide opportunistic sharing of created data during the deployment of WSNs. We monitor the network topology during deployment, and display live sensor data from sniffed WSN traffic.

about WSNs. The former two are systems that are used during deployment, to collect location information about nodes. *iButton Assist* also provides a method of reading sensor data from the deployed devices by connecting them to the smartphone. The *GSN Project* can be used in the field as long as you have a mobile data connection to upload and/or view the collected data. What these tools do not provide, is a way of localizing the user on a map or in relation to the deployed nodes. None of them provide live topology information *during* the deployment. Experience has shown, this is a much needed information in understanding how the system is communicating, and where to eliminate problems arising from wireless signal propagation issues.

Map Making: We can directly use the aerial images obtained with our system during deployments. Once stored on the client device, we can add them to deployed nodes, or include them as separate image annotations in the map. Our system does not provide any stitching functionality, as the related works shown in the Chapter 3. These processes are designed for use on personal computers or laptops. Thus, to created maps we need to provide a machine with *MapKnitter* or *MapCruncher* installed.

Looking at the sensor data from figs. 6.10 to 6.16, we believe it is possible to automatically select feasible images for map stitching. Using the sensor data, blurred or images with the wrong angle can be discarded automatically and thus minimizing the required time needed to review images in detail. Although, the installation of the smartphone below the kite was very unstable, c.f. figs. 6.10 to 6.14, we were still able to create a map of the deployment area, see Figure 6.7. Still, this can be greatly improved in the future by installing the device in a more stable fashion below the kite.

Maps: Satellite images from sources such as Google Maps are a good starting point for offline maps and as a photographic overview of the deployment site. Unfortunately, these are not guaranteed to be up-to-date, and in the desired resolution. With a balloon or kite and our system configured in the respective mode we can instantly access aerial images of the deployment site. This happens during surveying, deployment and maintenance. We have shown, that with our system and the right tools, we can create maps of the deployment area. Although it depends on the quality of the camera, and how stable it is attached to a balloon or kite, the quality of the resulting maps is close to the high resolution images of for example Google Maps, at a very low cost using off-the-shelf devices.

Additionally, as long as balloon device is idle it maintains the role of a deployment participant. Thus, it also synchronizes with other participants, providing a possible range extension to others below the balloon device. This way it is also possible to add an XBee node to the balloon/kite and monitor the WSN from higher altitudes. In this case, the topology and sensor data of the network is distributed among peers as soon as they synchronize with the balloon/kite.

Opportunistic Communication: The synchronization process between participants is only triggered as soon as a participant adds new data to the deployment. If the connection to another participant fails, synchronization request is not repeated. In this case, the user has to manually requests our system to do so. Also, if there are multiple possibilities to connect between two participants, e.g. via Bluetooth and Wi-Fi, both are made without any heuristics using distance, or available bandwidth.

This is not ideal, as Wi-Fi provides a higher bandwidth and should be preferred over Bluetooth. The exchange of high volumes of data, beginning at size of tens of megabytes should be done with the highest bandwidth, and thus prefer faster networks. On the other side the exchange of meta data, i.e. the association of images belonging to a certain sensor node, or simple messages, can be done via Bluetooth without significant delays. In this case the amount of data exchanged during synchronization is in the range of 10 to 100 kilobytes.

The results from the questionnaire confirm our position that this is an important technology for WSN deployments. Also, the evaluation of the questionnaire shows our system satisfies these requirements. Additionally, our work was prized with the "Coolest Demo" award during the conference, the highest of three prizes at the end of the conference.

6.6 Summary

Comparing the pre-deployment with the demo deployment at ExtremeCom 2013 we see that we now have a system that would have greatly improved the speed of progress during the pre-deployment. During the estimation of the first node locations we could have observed the messages of the sensor nodes on the map and seen how they were communicating. Instead of driving back to the base station, to check up if the nodes were still communicating via single hop links, the visualization of the network topology using our system would have shown us the communication patterns. The evaluation of a range test is done in the field now. By selecting a deployed node on the map, we see the distance to the node. The same holds for the deployment of the first nodes. After selecting the base station we could have monitored the distance to the station and deployed nodes at precise distances.

We successfully tested our system during the demo at ExtremeCom 2013. We recorded the location of the deployed WSN as well as the sensor data transmitted to the base station. The evaluation of the questionnaire not only confirms that we established correct requirements, it also shows that our work fulfills these requirements regarding WSN deployment support.

7

Conclusion

Our system provides collaborative communication on the deployment site, employing network technologies provided by current smartphones running the Android operating system. With our system we move away from *trial and error* approaches and provide a more efficient, collaborative method to incrementally deploy a WSN. We provide elaborate data to non-expert end users, that allows them to deploy a network and assess its current state.

We implemented an Android application, providing the user a simple method of starting a deployment and begin with recording her movements and actions. All images, messages to other users, voice and text annotations are shown on the map. The user can select them and view the information present at the respective location. With a simple interface the user can further select which information layer to show on the map. The nodes are shown on the map at the location where they are deployed. With the addition of an XBee radio module attached to the smartphone via USB, we can sniff the sensor network data. The received information is displayed on the map and distributed among other users of our system that are in communication range of the radio systems we use for data synchronization. This way a user can instantly see the battery state, as well as a live view of the sensor data the node is sending. During testing phase of our system we noticed that the nodes we were given for testing purposes had faulty sensors.

With the balloon mode, we can stream images from the bird's eye view, giving us an additional overview of the area. With these images we can create maps that are then distributed among the participants. For this the created map archive only needs to be copied to a single device. Upon synchronization the map is uploaded and copied to the next device. This way it is sufficient for a single person to prepare our system for a deployment. Upon bootstrapping the required maps and information are exchanged, and further participants added to the system.

The data collected by our system can be easily exported and accessed through various programming languages, whilst maintaining compatibility between protocol versions. This only requires a compilation of the ProtoBuf definition to the respective

programming language. The same way our protocols can be easily ported to other platforms. Considering the given possibilities we believe that our work can be extended and further improved to provide a powerful set of tools to assist during the survey, deployment and maintenance of wireless sensor networks.

7.1 Future Work

While this thesis has laid the groundwork for a collaborative system to support the deployment of sensor networks, we see room for future improvements. In the following we list aspects of our system that we would like to improve in the future:

- **Opportunistic communication improvements:** Synchronization between a participant and all others is only triggered when someone adds new data to our system. If someone is not reachable and returns within communication range, no synchronization attempts are established as long as no new data is generated. In this case the synchronization process between the outdated users has to be done manually. In terms of quality of experience we need a system that tries to synchronize in intervals, without flooding the network with requests, as well as conserving battery power.
- **Data Interpretation:** The data we collect provides us with clues about the state of the network. Our system could be extended to provide automatic assistance to the end user. Possible scenarios could be a warning signal, when nodes run low on power, are separated from the network, or not connected at all. From the sniffed WSN traffic we know when and *where* an packet was received. A possible addition would be a high-level interpretation that signals where to place a sensor node. For example, a traffic light indicating the current signal quality of the node's neighborhood.
- **Support for further WSN types:** Our system currently only supports the X-SLEWS sensor network platform. Other hardware is currently not supported for the display of live sensor network data. We provide a standalone XBee driver, paving the way for the inclusion of further 802.15.4 based WSNs.
- **802.15.4 Communication between participants:** With our driver we can also send and receive messages directly between devices. Thus, a further improvement would be to include a simplified synchronization protocol that could be communicated over the WSN or directly between participants. Note, the message size for communicating this way is limited to 102 bytes, and requires an additional layer to exchange data.
- **802.15.4 Communication with nodes:** With the XBee driver it would be possible to communicate directly with a sensor node. Either to request further data, such as routing tables, or even reprogram it.
- **Smartphone based WSN sink:** It would be feasible to port the base station to a smartphone. This would significantly reduce the size of the base station, as well as allowing the introduction of temporary mobile base stations as we are walking through the environment. A smartphone as a base station could also provide additional debugging options via the display.

- **Bug fixing:** Our system is a prototype, thus it is not entirely tested. Only parts of the libraries are covered with JUnit tests. To avoid bugs in the future, this needs to be completed. Further more, we uncovered errors in 3rd party libraries we use in our system. Among the major bugs are Out-Of-Memory-Exceptions, and segmentation faults within the Bluetooth sub-system.

Bibliography

[1] 3rd Generation Partnership Project. 3gpp - lte@ONLINE, Sept. 2013. http://www.3gpp.org/Technologies/Keywords-Acronyms/LTE.

[2] Andrea Antonello. Geopaparazzi @ONLINE, Sept. 2013. https://code.google.com/p/geopaparazzi/.

[3] Avvenuti, M., Cassano, L., Cesarini, D., and Mandala, S. Simulation of automatic weather stations for the energy estimation of sensing and communication software policies. In *In Proceedings of the 5th Extreme Conference on Communication* (2013), ACM.

[4] Barrenetxea, G., Ingelrest, F., Schaefer, G., and Vetterli, M. The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), ACM, pp. 43–56.

[5] Beutel, J., Gruber, S., Gubler, S., Hasler, A., Keller, M., Lim, R., Talzi, I., Thiele, L., Tschudin, C., and Yücel, M. The permasense remote monitoring infrastructure. In *ISSW*, vol. 9, pp. 187–191.

[6] Bitstars. Droidar mobile locationbased augmented reality framework for android@ONLINE, Sept. 2013. https://github.com/bitstars/droidar.

[7] Bluetooth Special Interest Group. Bluetooth basics @ONLINE, Sept. 2013. http://www.bluetooth.com/Pages/Basics.aspx.

[8] Bluetooth Special Interest Group. Bluetooth fast facts @ONLINE, Sept. 2013. http://www.bluetooth.com/Pages/Fast-Facts.aspx.

[9] Boano, C. A., Wennerström, H., Zúñiga, M. A., Brown, J., Keppitiyagama, C., Oppermann, F. J., Roedig, U., Nordén, L.-Å., Voigt, T., and Römer, K. Hot packets: A systematic evaluation of the effect of temperature on low power wireless transceivers. In *In Proceedings of the 5th Extreme Conference on Communication* (2013), ACM.

[10] Costanza, E., Panchard, J., Zufferey, G., Nembrini, J., Freudiger, J., Huang, J., and Hubaux, J.-P. Sensortune: a mobile auditory interface for diy wireless sensor networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 2317–2326.

[11] Di Palma, V., Periton, D., and Lathouri, M. *Intimate Metropolis: Urban Subjects in the Modern City*. Routledge, 2008.

[12] Digi International Inc. Xbee® 802.15.4, device connectivity using multipoint wireless networks @ONLINE, Sept. 2013. http://www.digieurope.de/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#specs.

[13] ESRI Geographic information system company. Arcgis - mapping and spatial analysis for understanding our world@ONLINE, Sept. 2013. http://www.esri.com/software/arcgis.

[14] European Telecommunications Standards Institute. Mobile technologies gsm @ONLINE, Sept. 2013. http://www.etsi.org/technologies-clusters/technologies/mobile/cellular-history.

[15] European Telecommunications Standards Institute. Universal mobile telecommunications system @ONLINE, Sept. 2013. http://www.etsi.org/technologies-clusters/technologies/mobile/umts.

[16] Fernandez-Steeger, T., Arnhardt, C., Walter, K., Hass, S., Niemeyer, F., Nakaten, B., Homfeld, S., Asch, K., Azzam, R., Bill, R., et al. Slews—a prototype system for flexible real time monitoring of landslides using an open spatial data infrastructure and wireless sensor networks. *Geotechnologien science report 13* (2009), 3–15.

[17] Fernández-Steeger, T., Ceriotti, M., Bitsch Link, J. Á., May, M., Hentschel, K., and Wehrle, K. "and then, the weekend started": Story of a wsn deployment on a construction site. *Journal of Sensor and Actuator Networks 2*, 1 (2013), 156–171.

[18] Fernández-Steeger, T., Ceriotti, M., Bitsch Link, J. A., May, M., Hentschel, K., and Wehrle, K. "And Then, the Weekend Started": Story of a WSN Deployment on a Construction Site. *Journal of Sensor and Actuator Networks 2*, 1 (2013), 156–171. http://www.mdpi.com/2224-2708/2/1/156.

[19] Friedman, R., Kogan, A., and Krivolapov, Y. On power and throughput tradeoffs of wifi and bluetooth in smartphones. *Mobile Computing, IEEE Transactions on 12*, 7 (2013), 1363–1376.

[20] Future Technology Devices International Ltd. D2xx drivers@ONLINE, Sept. 2013. http://www.ftdichip.com/Drivers/D2XX.htm.

[21] Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., and Razafindralambo, T. A survey on facilities for experimental internet of things research. *Communications Magazine, IEEE 49*, 11 (2011), 58–67.

[22] Google Inc. Android compatibility downloads@ONLINE, Sept. 2013. http://source.android.com/compatibility/downloads.html.

[23] Google Inc. Protocol buffers@ONLINE, Sept. 2013. https://developers.google.com/protocol-buffers/docs/overview.

[24] GSN Project. Introduction - gsn@ONLINE, Sept. 2013. http://sourceforge.net/apps/trac/gsn/wiki/Introduction.

[25] Humanitarian OpenStreetMap Team. Mapmill : Help the civil air patrol sort images@ONLINE, Sept. 2013. http://mapmill.hotosm.org/.

[26] Jeffrey Warren. A hot-or-not styled crowdsourcing engine for sorting raw map imagery @ONLINE, Sept. 2013. http://mapmill.org/.

[27] Jeffrey Warren. Mapknitter: Use public laboratory's map knitter to upload your own aerial imagery and combine it into a geotiff and tms/openlayers map.@ONLINE, Sept. 2013. https://github.com/jywarren/mapknitter.

[28] Langendoen, K., Baggio, A., and Visser, O. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International* (2006), IEEE, pp. 8–pp.

[29] Martinez, K., Hart, J. K., and Ong, R. Deploying a wireless sensor network in iceland. In *GeoSensor Networks*. Springer, 2009, pp. 131–137.

[30] Maxim Integrated. Overview of ibutton® sensors and temperature/humidity data loggers @ONLINE, Sept. 2013. http://www.maximintegrated.com/app-notes/index.mvp/id/3892.

[31] May, M. X-SLEWS: Developing a Modular Wireless Monitoring System for Geoengineering Wide Area Monitoring. Master thesis, RWTH Aachen University, March 2013.

[32] Mobile Atlas Creator Community. Mobile atlas creator: Prepare online maps for your mobile device@ONLINE, Sept. 2013. http://mobac.sourceforge.net/.

[33] Monteiro, M., Goldman, A., and Amantea, G. Wip: A framework to evaluate routing protocols on mobile networks using realistic scenarios. In *In Proceedings of the 5th Extreme Conference on Communication* (2013), ACM.

[34] NeuroSky Inc. Mindwave mobile: Myndplay bundle@ONLINE, Sept. 2013. http://store.neurosky.com/products/mindwave-mobile.

[35] Nicolas Guillaumin. Osmtracker @ONLINE, Sept. 2013. https://code.google.com/p/osmtracker-android/.

[36] OSMDroid Community. Osmbonuspack: A third-party library of (very) useful additional objects for osmdroid@ONLINE, Sept. 2013. https://code.google.com/p/osmbonuspack/.

[37] OSMDroid Community. Osmdroid: Openstreetmap-tools for android@ONLINE, Sept. 2013. https://code.google.com/p/osmdroid/.

[38] PermaSense Consortium. Permasense field sites @ONLINE, Sept. 2013. http://www.permasense.ch/field-sites.html.

[39] Prof. Mauro De Donatis. Beegis digital field mapping@ONLINE, Sept. 2013. http://www.beegis.org/.

[40] Public Laboratory for Open Technology and Science. Mapknitter - about@ONLINE, Sept. 2013. https://github.com/jywarren/mapknitter.

[41] Quantum GIS Project. Quantum gis@ONLINE, Sept. 2013. http://www.qgis.org/index.php.

[42] Refractions Research. udig user-friendly desktop internet gis@ONLINE, Sept. 2013. http://udig.refractions.net/.

[43] Society, L. S. C. I. C. Ieee standard for local and metropolitan area networks, part 15.4: Low-rate wireless personal area networks (lr-wpans). *IEEE Std* (2011).

[44] Square. Otto: An enhanced event bus with emphasis on android support@ONLINE, Sept. 2013. http://square.github.io/otto/.

[45] Szewczyk, R., Polastre, J., Mainwaring, A., and Culler, D. Lessons from a sensor network expedition. In *Wireless Sensor Networks*. Springer, 2004, pp. 307–322.

[46] TIK Institute, ETH Zurich. ibutton assist @ONLINE, Sept. 2013. https://play.google.com/store/apps/details?id=ch.ethz.iassist.

[47] TIK Institute, ETH Zurich. idnotebook@ONLINE, Sept. 2013. https://play.google.com/store/apps/details?id=ch.ethz.idnotebook.

[48] Vasilescu, I., Kotay, K., Rus, D., Dunbabin, M., and Corke, P. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd international conference on Embedded networked sensor systems* (2005), ACM, pp. 154–165.

[49] Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., and Welsh, M. Deploying a wireless sensor network on an active volcano. *Internet Computing, IEEE 10*, 2 (2006), 18–25.

[50] WiFi Alliance. Discover and learn @ONLINE, Sept. 2013. http://www.wi-fi.org/discover-and-learn.

[51] WiFi Alliance. Wi-fi direct @ONLINE, Sept. 2013. http://www.wi-fi.org/discover-and-learn/wi-fi-direct.

[52] WiFi Alliance. Wi-fi faq: Is wi-fi direct the same as ad hoc mode? @ONLINE, Sept. 2013. http://www.wi-fi.org/knowledge-center/faq/same-ad-hoc-mode.

[53] ZXing Community. Zxing: Multi-format 1d/2d barcode image processing library with clients for android, java@ONLINE, Sept. 2013. https://code.google.com/p/zxing/.

A

Appendix

Questionaire about DEMO: Opportunistic Deployment Support for WSNs

1. **Did you use our Android application?** ○ No. ○ Installed on own device. ○ Used other device.
2. **How relevant do you consider our system for WSN deployments in general?** Not at all ○—○—○—○—○ Very
3. **How helpful do you consider each part of our system for WSN deployments?**
   - Opportunistic communication — Not at all ○—○—○—○—○ Very
   - Process of gathering node related information — Not at all ○—○—○—○—○ Very
   - Map display (i.e. offline maps, display of network topology, messages, annotations) — Not at all ○—○—○—○—○ Very
   - Augmented Reality — Not at all ○—○—○—○—○ Very
   - Audio Guidance — Not at all ○—○—○—○—○ Very
   - Balloon Mode / Balloon Client — Not at all ○—○—○—○—○ Very
4. **How do you rate our current work?**
   - Opportunistic communication — Poor ○—○—○—○—○ Excellent
   - Process of gathering node related information — Poor ○—○—○—○—○ Excellent
   - Map display (i.e. offline maps, display of network topology, messages, annotations) — Poor ○—○—○—○—○ Excellent
   - Augmented Reality — Poor ○—○—○—○—○ Excellent
   - Audio Guidance — Poor ○—○—○—○—○ Excellent
   - Balloon Mode / Balloon Client — Poor ○—○—○—○—○ Excellent

The following 3 questions only apply if you have participated during at least one sensornetwork deplyoment before.

5. **How was the availability of infrastrucure for data communication during the deployment(s)?**
   - GSM/3G — None ○—○—○—○—○ Excellent
   - WiFi AP(s) — None ○—○—○—○—○ Excellent
   - Other: _____ — None ○—○—○—○—○ Excellent
6. **How did you collect information about the deployment (e.g. location of nodes, network topology)?**

   _____

   _____

   _____
7. **How did you communicate and share collected data with others during the deployment?**

   _____

   _____

   _____
8. **Do you plan to use our system at some point in the future?** Not at all ○—○—○—○—○ Very
9. **Are you interested in contributing to our system?** Not at all ○—○—○—○—○ Very
10. **Further comments:**

    _____

    _____

    _____

    _____

Figure A.1 The questionnaire used for the evaluation of our work during ExtremeCom 2013.

Figure A.2 Our system was awarded the "Coolest Demo" award at ExtremeCom 2013. This is the trophy. The length is approx. 40cm.