Bernd Baumann, BSc

NetLoc: A UWB-based Localization System for the Internet of Things

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Bernhard Großwindhager

Ass.Prof. Dr.techn. MSc Carlo Alberto Boano

Institute for Technical Informatics
Head: Univ.-Prof. Dipl-Inform. Dr.sc.ETH Kay Uwe Römer

Graz, January 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Kurzfassung

Das Internet der Dinge (IoT) wird ein zunehmend wichtigerer Bestandteil unseres Alltags. Milliarden kabellos vernetzte, intelligente Objekte ermöglichen - unter anderem - die Entwicklung intelligenter Gebäude, Stromnetze und Städte. Miteinander vernetzte Objekte können zum Beispiel: (i) helfen schneller Parkplätze in Großstädten zu finden, (ii) durch Integration in Abfallbehälter die Effizienz der Abfallwirtschaft steigern, (iii) durch weltweite Netzwerke aus Feinstaub-Sensoren und Radioaktivität-Sensoren vor gesundheitsgefährdenden Umweltbedingungen warnen. Um die gesammelten Daten dieser IoT Geräte korrekt zu interpretieren, ist es unbedingt notwendig deren Standort zu kennen, weshalb Lokalisierung eine wesentliche Rolle in derartigen Anwendungen darstellt. Während die Positionsbestimmung im Freien weltweit durch Satellitensysteme (wie GPS) ermöglicht wird, gibt es kein etabliertes System zur Lokalisierung innerhalb von Gebäuden. In dieser Masterarbeit präsentiere ich NetLoc, ein ultra-wideband basiertes Lokalisierungssystem für IPv6-kompatible Netzwerke, welches trotz minimaler Infrastruktur präzise Positionsbestimmung ermöglicht. Eine neue energiesparende und preiswerte IoT Plattform wurde dazu entwickelt. Teil dieser Plattform ist der Decawave DW1000 UWB Transceiver, welcher es ermöglicht zentimetergenau Distanzen zwischen zwei Geräten zu messen. Mithilfe dieser Plattform wird ein Mesh-Netzwerk, bestehend aus vielen mobilen Geräten, generiert. Dazu wird das standardisierte RPL Routing Protokoll verwendet, um das Netzwerk zu verwalten und Daten im Netzwerk intern weiterzuleiten. Die RPL Nachrichten wurden insofern erweitert, dass Distanzen gemessen und diese Informationen zu einem zentralen Knoten gesendet werden können, ohne dabei zusätzliche Nachrichten schicken zu müssen. Des Weiteren wurde ein Lokalisierungs-Algorithmus entwickelt, der ohne zusätzliche Karteninformationen die Positionen aller im Netzwerk befindlichen Geräte bestimmen und visuell darstellen kann. Weil das Generieren von Positionsdaten verteilt im gesamten Netzwerk stattfindet, ist der NetLoc Lokalisierungs-Algorithmus nicht nur äußerst skalierbar, sondern die Positionsgenauigkeit steigt sogar mit der Anzahl an Nachbarn im Netzwerk. Eine experimentelle Evaluierung zeigt, dass trotz minimal nötiger Infrastruktur dezimetergenaue Positionierung möglich ist. Außerdem werden ausschließlich standard-konforme Kommunikationsprotokolle wie RPL verwendet, wodurch NetLoc kompatibel mit anderen IPv6-basierenden Systemen und damit erweiterbar und zukunftssicher ist.

Abstract

The Internet of Things (IoT) is rapidly becoming an integral part of our daily lives, with billions of wirelessly-networked smart objects empowering - among others - the development of smart buildings, grids, and cities. Such interconnected smart objects can, for example: (i) help us to find parking spaces in large cities, (ii) be embedded in waste containers to make waste management more efficient, and (iii) be used to form global networks of fine dust sensors and radioactive radiation sensors that alert us about hazardous environmental conditions. Location-awareness is a fundamental prerequisite to correctly interpret the data collected by IoT devices and therefore plays an essential role in such applications. While outdoors the problem of localization can be solved by using global navigation satellite systems (e.g. GPS), a well-established indoor localization solution for IoT devices has not yet been developed. In this thesis I present NetLoc, a localization system for IPv6-enabled networks based on ultra-wideband technology (UWB) that accurately determines the position of mobile devices despite requiring minimal infrastructure. To build NetLoc, a new low-power, low-cost IoT platform has been designed. This platform, which embeds the Decawave DW1000, a popular UWB transceiver capable of measuring centimeter-accurate distances, is used to form a mesh network consisting of multiple mobile devices. NetLoc uses the standardized RPL protocol to route data and manage such mesh networks. We the extended RPL messages used to build and maintain the mesh network in order to also measure the distance between adjacent neighbors and collect these distances at a central sink without the need of sending extra packets. We further develop a localization algorithm that allows a sink to locate each node in the network and visualize their positions without the need of prior knowledge on the environment. As the collection of location information is distributed throughout the network, NetLoc's localization algorithm is not only highly scalable, but its accuracy even increases with the number of neighbors. An experimental evaluation shows that, despite requiring only a minimal fixed infrastructure, NetLoc achieves decimeter-level positioning accuracy. Furthermore, the use of standard-compliant protocols such as RPL makes NetLoc compatible with other IPv6-based systems and hence makes it extendable and future-proof.

Acknowledgments

This master's thesis was conducted at the Institute for Technical Informatics at Graz University of Technology.

First and foremost, I would like to thank my supervisors Bernhard Großwindhager and Carlo Alberto Boano for their help and support during my work on this thesis. I wish to thank Bernhard for approaching me and introducing me to the fascinating field of ultra-wideband technology, as well as for his help and advice while designing the hardware.

I am deeply grateful to Carlo for guiding and helping me throughout my master's programme, especially during my work on this thesis. Thanks to his **extraordinary** courses I got interested in embedded devices and wireless sensor networks in the first place.

I thank my girlfriend, Elli, for her support, her motivation, and her patience with me. I would not be where I am today without her help.

Graz, January 2019 Bernd Baumann

Contents

List of Abbreviations

6LoWPAN	IPv6 over Low-power Wireless Personal Area Networks
ACK	Acknowledge
BLE, BTLE	Bluetooth Low Energy
BPM	Burst Position Modulation
BPSK	Binary Phase Shift Keying
CDF	Cumulative Distribution Function
CMSIS	Cortex Microcontroller Sofware Interface Standard
DAO	DODAG Advertisement Object
DIO	DODAG Information Object
DIS	DODAG Information Solicitation
DS-TWR	Double-Sided Two-Way Ranging
DODAG	Destination-Oriented Directed Acyclic Graph
ETX	Expected Number of Transmissions
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
ICMPv6	Internet Control Message Protocol version 6
IR-UWB	Impulse Radio Ultra-Wideband
IoT	Internet of Things
ISR	Interrupt Service Routine
LLN	Low-power Lossy Network
MAC	Medium Access Control
MCU	Microcontroller Unit
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PCB	Printed Circuit Board
PoE	Power over Ethernet
PRF	Pulse Repetition Frequency
RDC	Radio Duty Cycle
RPL	Routing Protocol for Low-power and Lossy Networks
RSS	Received Signal Strength
RTC	Real-time Clock
RTLS	Real-time Localization System
SS-TWR	Single-Sided Two-Way Ranging
TWR	Two-Way Ranging
TOF	Time of Flight
UC	Unicast
UWB	Ultra-Wideband

List of Figures

List of Tables

Chapter 1

Introduction

In recent years the market for smart and interconnected objects has grown constantly. The evolution of wireless sensor networks (WSN) led to an Internet of Things (IoT) consisting of billions of connected devices [35] and paved the way for attractive applications. One example of such applications are smart parking meters that support automatic payment if a car parks in their vicinity. Furthermore, these sensors help drivers to find a free parking space [56]. Another example of an IoT application is smart waste management, where sensors inside waste containers autonomously report the amount of waste such that the garbage collection process can be made more efficient [34]. There are also global sensor networks that inform and warn people about fine dust in the air [60]. In all these IoT applications, location-awareness plays an essential role.

Location-aware WSNs have been successfully deployed in both indoor and outdoor environments to monitor volcanic eruptions [84], track the movement and population of cane toads [45], or control efficient electric lighting in tunnels [11]. In these examples the devices' positions did not change during their lifetime, so there was no need to perform localization on a periodic basis. In the presence of mobile devices, instead, one needs to localize these devices periodically to correctly interpret their measurements. Outdoors, IoT applications can utilize localization systems based on cellular networks and Global Navigation Satellite Systems (GNSS) (e.g. GPS) to determine the position of a device. The IoT application *Safecast*, for example, uses GPS to link measurements of radioactive radiation levels to a specific location [7].

While many indoor applications requiring location information exist, they cannot rely on a globally available system. Satellite communication is hardly possible indoors and position measurements based on cellular networks are too coarse. As a result, indoor IoT applications typically implement a custom localization system that relies on its own infrastructure.

Indoor navigation systems like *Infsoft* [49] and *Mapsted* [16] aim to provide a real-time localization system (RTLS) that can track their user's movements on a map in real-time. Apart from indoor navigation, RTLSs are also used to increase security in large buildings and to lower labor costs. For example, the company *OpenRTLS* equipped a dutch hospital with a multipurpose RTLS consisting of thousands of devices [67]. By tracking the hospital's equipment, patients and staff in real-time, the system gives insights in asset utilization, provides secure access control, and lowers labor cost for inventory searching and counting.

1.1 Problem Statement

Depending on the specific requirements, the problem of locating a mobile device or letting a mobile device locate itself was solved differently in various applications. The three aforementioned examples by *Infsoft*, *Mapsted* and *OpenRTLS* show typical approaches. Because of their underlying technology and their implementation, these systems are limited either in terms of accuracy, infrastructure cost, or interoperability. To avoid isolated networks, the ideal solution uses only standardized communication protocols and locates devices at a high accuracy while requiring a minimal infrastructure, as discussed next.

1.1.1 Achieving Decimeter-Level Localization Accuracy

Especially in indoor environments, where objects tend to be closer together than outdoors, there are applications requiring sub-meter accuracy. The position accuracy that a system can achieve, however, heavily depends on the underlying technology.

The indoor navigation system by *Infsoft*, for example, uses a combination of microelectromechanical system (MEMS) sensors and received signal strength (RSS) measurements of WiFi access points and Bluetooth low-energy (BLE) beacons to estimate the position of a user's smartphone. With increased distance to the signal source, the RSS decreases: this allows client-based positioning at an accuracy of 5 to 15 meters. Such a distance is **not** accurate enough to find a specific item within a store or the correct check-in desk at an airport, but it allows to find the correct departure gate, a shop within a mall, or a car on a parking lot. To calculate the position based on distance measurements, the locations of these signal sources (WiFi access points, BLE beacons) must be known and a sufficient number of signal sources must be within range.

The *Mapsted* system provides a machine learning based approach. By training a classifier, the application recognizes if new RSS measurements of WiFi access points and BLE beacons are similar to previous measurements at a certain position on a map. This way, positions can be estimated with an accuracy of 2 to 5 meters.

Thus, an application that requires decimeter accuracy cannot rely on RSS measurements of WiFi, Zigbee or BLE signals. The aforementioned RTLS (OpenRTLS) deployed inside a hospital derives the positions of mobile devices from time-of-flight (TOF) measurements using ultra-wideband (UWB) technology. The latter is capable of locating every single mobile device on the site in real-time with an accuracy of 0.2 meters. The enabling technology for this accuracy is impulse-radio UWB (IR-UWB), which is the basis for one of the most accurate radio based indoor localization systems available today [18]. However, as one cannot reuse existing WiFi access points, there is a need to add infrastructure, which increases the cost, setup time, and labor-intensiveness of the solution.

1.1.2 Minimizing the Necessary Infrastructure

An important cost factor is indeed the setup and maintenance of the infrastructure, which must provide both network connectivity and localization capabilities. Localization is typically implemented by estimating the position from distance, or angle measurements to known reference positions. Devices that are placed at reference positions are referred to as anchors, devices that must be located are called tags. To operate correctly, tags must be in the communication range of anchors.

To save costs, both *Infsoft* and *Mapsted* make use of existing infrastructure in indoor environments like shopping malls and airports. Processing measurements and estimating the positions is not done on a server, but client-based on the user's smartphones. Thus, the system operator does not need to acquire and maintain tags. Only where the existing infrastructure is insufficient, additional BLE beacons are deployed. Since many of the access points that are used as anchors are not operated by the system provider, determining the positions of all access points can be infeasible. Furthermore, there is no guarantee for fixed anchor positions: WiFi access points or BLE beacons may indeed be removed or relocated, hence requiring a reconfiguration of the map or repeating the time consuming process of training a classifier. In very dynamic environments, the maintenance costs may outweigh the benefits of reused infrastructure.

OpenRTLS' UWB-based system has over 4000 anchors installed at a $350\,000\,m^2$ large area consisting of multiple buildings and parking lots. Thousands of tags were attached to wheel chairs, stretchers, infusion pumps, staff, patients and visitors. Anchors are tethered via Power over Ethernet (PoE). Special PoE compliant network switching hardware is needed to connect them. To cover larger areas with fewer anchors, long-range antennas are used. Horn antennas increase the communication range from tens of meters to several hundred meters. Locating a tag requires a scheduled two-way message exchange to at least three anchors, which reduces the scalability. Another problem is a decreasing update rate per tag if too many tags must communicate to the same anchors. Furthermore, the system possibly violates its real-time capability if the update rate of tag positions decreases too much. There are different solutions to this problem[1], one of them is allowing a tag that is already localized to communicate its position to neighboring nodes: this tag can then act as an anchor and reduce the traffic load of fixed anchors [88]. In general, one wants to minimize the necessary amount of anchor nodes and therefore the infrastructure cost, while maximizing the area that the system can cover.

1.1.3 Making use of Standard-Compliant Protocols only

The tags of the *OpenRTLS* system are not connected to the Internet, but used in an isolated localization system only. Hardware and software are both proprietary and cannot be combined with third-party systems. This vertical solution of having an isolated network that is accessed by a single application is inefficient and does not comply with the goal of an open Internet of Things. The customer depends on the vendor for maintenance and platform updates, a situation referred to as *vendor lock-in*. A standardized network protocol and open APIs would allow compatible platforms by other vendors to be integrated in the localization system.

Smartphone based applications like the ones by *Mapsted* and *Infsoft* have the advantage of Internet access via standardized communication protocols. The same situation applies for the *Safecast* example, where all nodes are accessible on the Internet and the system supports heterogeneous sensor nodes.

However, Safecast's hardware design is based on GPS which is too inaccurate and hardly available inside buildings, hence not suitable for indoor localization. The *Mapsted* and

[1] A protocol solution would be to use a combination of Time Division Multiple Access (TDMA) and Time Difference of Arrival (TDoA) methods to determine a tag's location [68]

Infsoft applications profit from a smartphone based solution in terms of standard compliance and Internet connectivity, but at the cost of accuracy and scalability. Despite a high density infrastructure and advanced positioning algorithms, the accuracy of RSS based position estimates cannot compete with an UWB-based solution. The goal is to provide a system where mobile devices communicate through standardized protocols and that achieves decimeter-level accuracy despite requiring a minimal infrastructure.

1.2 Contributions

In this thesis I present NetLoc, a location-aware IoT system based on UWB technology that embeds position-related information in its routing protocol. NetLoc uses a self-designed UWB platform containing a IEEE 802.15.4(a) compliant UWB transceiver allowing for precise distance measurements between any two devices, thus achieving localization at decimeter-level accuracy. Because of the properties of UWB, NetLoc is less prone to multi-path fading and cross-technology interference than solutions based on WiFi, BLE or Zigbee. Therefore, NetLoc is an ideal choice for an indoor localization system. Furthermore, NetLoc requires a minimal infrastructure to operate correctly. Only one single anchor node at a fixed, known position is required. All other nodes can move freely. Both communication and localization between devices take place in a multi-hop mesh network, which can be joined by devices ad hoc. Distances are only measured between local neighbors, making it highly scalable. Devices can be located even without a direct communication link to a fixed node. NetLoc's devices run the Contiki operating system for IoT devices [1]. The Contiki operating system provides standard-compliant implementations of the IPv6 network stack [33], the 6LoWPAN adaption layer [72], the RPL routing protocol [86] and a framer for IEEE 802.15.4 frames [47]. The system does not require custom hardware as long as the mobile devices support the same standardized network and routing protocols.

Distance measurements and data collection to perform localization are implemented in a highly efficient way. No additional packets need to be sent to estimate the positions of mobile devices. Instead, the existing DIO and DAO messages of RPL have been seamlessly extended in order to embed position-related information in the routing layer. The following sections summarize this thesis' contributions, which are: the design of a new, UWB-based IoT platform design that is efficient and low-cost (Section 1.2.1), a localization algorithm that just requires a single anchor (Section 1.2.3), a seamless extension of RPL allowing to gather location related data (Section 1.2.2), a localization application built on standard-compliant network and routing protocols (Section 1.2.4) as well as an experimental evaluation of the system and its localization performance (Section 1.2.5).

1.2.1 UWB IoT Platform

The first contribution of this thesis is the design of a UWB-based IoT platform. This new design is driven by the lack of a commercial UWB platform that satisfies a number of requirements, such as low-cost, low-energy, IoT OS support, and a SMA antenna connector. The desired platform should meet the requirements for IoT and positioning applications equally and allow individual power measurements of all components. The latter is important to evaluate different positioning methods and network protocols, since energy efficiency is a major design criterion for IoT systems. Furthermore, it would be benefi-

cial to have a modular hardware design that allows hardware upgrades and the exchange of platform components. Such a device does not exist among the investigated platforms, mainly because they were all designed to support RTLS applications, but ignored IoT requirements.

Our new hardware design is based on Decawave's DW1000 radio transceiver [22] and STMicroelectronics' STM32L152 ARM Cortex-M3 MCU [62]. This platform can be used to exploit all aspects of the DW1000 and the Contiki operating system. Its modular design allows the exchange of the UWB antenna and the extension of the hardware via standardized connectors. Still I have achieved a platform cost of less than 50EUR per node. Because the design allows voltage and current measurements of individual components, it especially supports researchers in the field of IoT and energy-constrained wireless communications.

The firmware for calibration is provided to ensure regulation compliance of the platform and to calibrate the transceiver. To operate this new platform, a DW1000 driver [15] and the Contiki operating system [77] were ported. Embedded applications were developed to verify correctness of distance measurements and correct route configuration.

1.2.2 Embedding Distance Information in RPL

The second contribution of this thesis is the extension of the RPL [2] protocol implementation, which allows gathering distance information at a single point in the network without the need of exchanging more packets. Appending timestamps to DIO packets allows the distances between nodes in a local neighborhood to be derived from TOF measurements. Because every node measures and saves the neighbor distances locally, the number of measurements per node does not scale with the network size but only with the number of neighbors. Extending DAO packets allows the collection of the network's distance information at a single point without sending additional packets. This procedure makes this localization application very energy-efficient. Since IoT devices are often battery powered, this is a significant advantage of NetLoc.

1.2.3 Localization Algorithm and Visualization

The third contribution of this thesis is the design and implementation of a localization algorithm. The underlying idea of this algorithm is the representation of a sensor network as a weighted graph. The weights of the edges correspond to physical distances between a pair of devices. If every node of the graph has a degree of three or higher, its two-dimensional representation is unique. To collect and process distance information efficiently, the application consists of two separate processes running simultaneously. One process to receive and store collected measurements and another process to parse this data, compute coordinates from distance measurements and visualize them in a two-dimensional coordinate system. Visualization and localization can include map information like objects and walls, but a map is not strictly required. Without a map, the network will be localized in a coordinate system with the anchor node in its origin.

[2] Refer to Section 2.1.4 for an explanation of the RPL protocol.

1.2.4 Standard-Compliant Protocols

The communication between devices in the NetLoc system complies to public standards, which means any other standard compliant IoT platform can join NetLoc's network. Contiki implements a standard-compliant IPv6 network stack that is optimized for constraint IoT devices. Furthermore, Contiki supports 6LoWPAN, an adaptation layer between the IPv6 network layer and the IEEE 802.15.4 link layer, which supports header compression and packet fragmentation. Regarding the hardware, NetLoc uses the DW1000, an IEEE 802.15.4(a) compliant IR-UWB radio transceiver. Furthermore, Contiki provides RPL, a routing protocol that is specifically designed for constraint IoT devices.

1.2.5 NetLoc Evaluation

The fourth contribution of this thesis is the evaluation of NetLoc's localization accuracy in laboratory experiments on a network consisting of nine devices. The position of every device in the network was estimated about 2200 times to allow a statistical analysis of the system's performance. Devices were distributed in a $63\,m^2$ room on fixed positions. The results show that over 90 percent of all estimates have an error of less than 30 cm, hence showing that NetLoc provides an accurate localization service in a cost- and energy-efficient network that requires little infrastructure.

1.3 Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces UWB technology and describes basic concepts about common IoT systems and localization algorithms that are needed to understand the contributions of this thesis. NetLoc's architecture and its functionality is described in Chapter 3. Chapters 4 and 5 describe in detail the hardware and software design, respectively. Chapter 6 illustrates the performance of NetLoc by showing the results of an experimental evaluation and the employed setup. Finally, Chapter 7 summarizes our conclusions and provides an outlook on future work.

Chapter 2

Background

Section 2.1 gives a short introduction to the field of IoT and an overview of their typical hardware and software characteristics. The Contiki operating system and its network stack, including the IPv6 and RPL protocols, are explained in Section 2.1.1 and Section 2.1.2, respectively. An introduction to UWB impulse radios and how they can be used to accurately measure distances is given in Section 2.2. Indoor localization concepts are explained in Section 2.3, with special focus on methods based on distance measurements. In Section 2.4 the relation between a network graph and its distance matrix is explained, which will be used in Section 5.2 to derive coordinates from distance measurements.

2.1 Constrained IoT Devices

The distinct requirements of IoT applications led to the development of specialized hardware, software, and communication protocols. One characteristic property of IoT devices is that they tend to be embedded into their environment or into objects. They are, for example, worn inside bracelets or fitness trackers [36], embedded into thermostats and heaters [64], spread across vast, hazardous areas to monitor radioactive radiation levels [7], or embedded in a city's streets to inform about free parking spaces [56].

Typically, such applications require their devices to run for a long time on battery. Thus, hardware, software and protocols are specifically designed to save energy and to extend the device's lifetime.

Hardware Requirements. Every device consists at least of a MCU, a sensor, a radio, and a battery. The radio's energy consumption is typically the highest of all components. The popular CC2650 for example has a maximum CPU power consumption of 3mW, but a maximum radio power consumption of about 30mW [79]. Consequently, the radio is switched off most of the time to save energy. The radio duty cycle (RDC) depends on the medium access control (MAC) protocol and its configuration. One example for an ultra-low power MAC protocol is Dozer, having a RDC of only 0.2 percent [9]. In such constrained devices, typical radios support only low data rates of 100kbps to 250kbps. MCUs usually have an embedded real-time clock (RTC), so they can power down into a sleep mode until an expired timer wakes them up again. To save energy, MCUs run at low frequencies, usually less than 50MHz. Their RAM size is in the order of hundreds of

kilobytes, the ROM has typically less than one megabyte. Except for some newer MCUs, there is no hardware support for memory protection. However, one advantage of these small MCUs, apart from consuming little energy, is that they are low-cost: an important aspect, considering that IoT applications can require networks consisting of thousands of nodes.

Software Requirements. These hardware constraints demand efficient software with a low memory footprint. Example operating systems for sensor networks are Contiki [30], Riot [5], TinyOS [80], and Zephyr [37]. They enable the development of portable applications for sensor networks by implementing hardware abstractions, task scheduling and network stacks. Modern programming languages are not suitable to write applications for constrained devices. With the exception of TinyOS, which requires a special compiler[1], the mentioned operating systems typically support C/C++ programming only.

Network Requirements. The efficiency of medium access control, network and routing protocols is crucial for the lifetime of IoT devices. In Ethernet and TCP/IP networks instead energy consumption plays a minor role and the network protocols are optimized for latency, throughput and reliability. Wireless communication links in sensor networks are lossy: this limits the optimal packet size and maximum data rate. The network protocols should not overreact on packet loss and prefer energy-efficiency over latency. Packet loss is expected in low-power and lossy networks (LLN) and overreacting on lost packets by updating routing tables would be a waste of energy.

MAC and routing protocols also have to consider that IoT devices communicate in mesh networks. This means that a single node needs to be able to act as a router if necessary and forward packets to its neighbors. Therefore, the network protocols must include a mechanism for neighbor discovery. Due to the small packet size of common technologies the protocol overhead has an even larger impact than in computer networks.

2.1.1 Contiki OS

The Contiki operating system for constrained devices was created by Adam Dunkels in 2004 and constantly improved since then [30]. Contiki itself, as well as its applications, are written in the C programming language. The Contiki network protocol stack can be seen in Figure 2.1. The network stack is accessed through generic interfaces, which are listed in the left column of Figure 2.1. The right column lists the specific protocol implementations that are used in NetLoc. Section 3.2.2 describes them in the context of NetLoc's architecture. Contiki's *Network Layer* object combines the implementations of the *Adaptation* layer, the *Network* layer, the *Transport* layer, and the *Application* layer. Note, that radio duty cycling is not part of the MAC layer, but is implemented as a separate layer instead. Typically, the task of MAC layer protocols is to avoid packet collisions by using radio functions to sense if the medium is free and implement a back-off mechanism if the medium is busy. MAC protocols, which are supported by Contiki are CSMA, an implementation of carrier-sense mutliple access that will re-transmit packets if collisions are detected, TSCH [28], and `nullmac`. The latter is used in NetLoc's network stack as

[1] Applications must be written in the nesC [39] programming language.

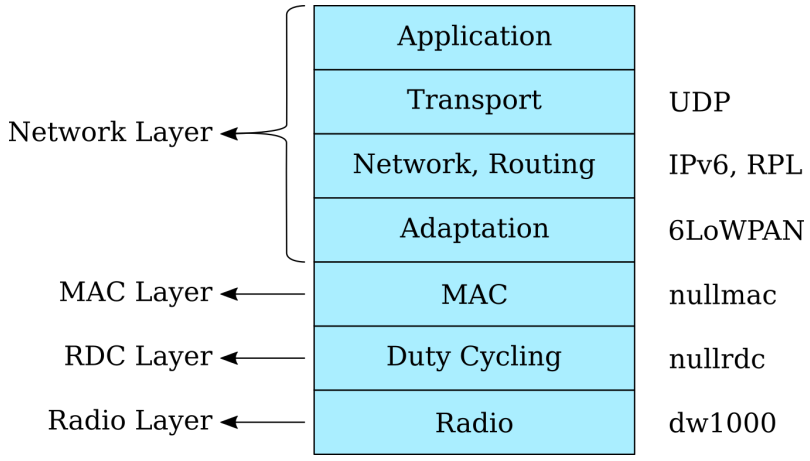


Figure 2.1: Contiki's network protocol stack [4].

shown in Figure 2.1. The RDC layer decides when to enable the radio and when to turn it off to save energy. Example RDC layers that are supported by Contiki are `cxmac` (an implementation of X-MAC [8]), `contikimac` [29], and `nullrdc`, which is used in NetLoc as shown in Figure 2.1. Contiki's main features are:

- Protothreads: non-preemptive, lightweight threads sharing a single stack [32].
- uIPv6 and 6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks): an IPv6 network stack with low memory footprint and support for neighbor discovery, stateless address auto-configuration, ICMPv6 and the 6LoWPAN adaptation layer, which implements a packet fragmentation mechanism as well as compression and decompression of IPv6 headers [33, 72]. Figure 2.1 shows 6LoWPAN as *Adaptation* layer below the IPv6 *Network* layer. The importance of IPv6 and 6LoWPAN for IoT networks is explained in Sections 2.1.2 and 2.1.3, respectively.
- RPL (IPv6 Routing Protocol for Low-power and Lossy Networks) in non-storing and storing mode: two operation modes of this routing protocol, specifically designed for low-power, lossy networks [82]. The location of this *Routing* layer within the network stack can be seen in Figure 2.1. Section 2.1.4 describes the RPL protocol and its design principles.
- Energest: an internal mechanism to estimate the energy consumption during execution [31].
- Cooja: a Java based simulator and packet analyzer.
- Coffee flash file system: Contiki's own implementation of an efficient file system design to manage files in a flash memory [81].

Figure 2.2 shows a block diagram of Contiki's software architecture[2].

[2]The `platform` block shows only a subset of the supported platforms and CPUs. The `user apps` block contains many more example applications, the displayed blocks are just some examples.

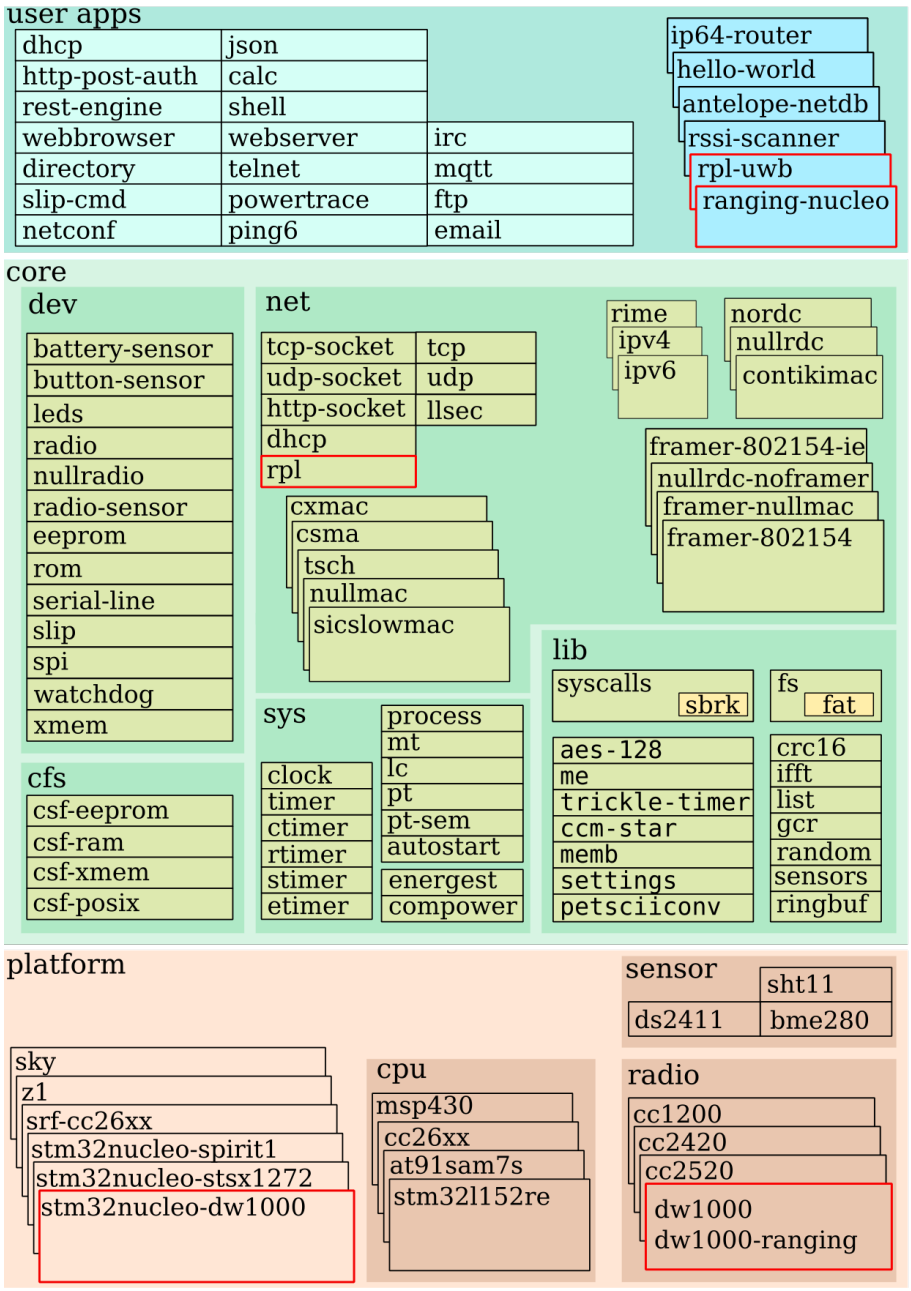


Figure 2.2: A block diagram of Contiki's software architecture. Blocks that are of special interest for the NetLoc system have red borders. Overlapping blocks exclude each other. Non-overlapping blocks can exist concurrently. User applications (`rpl-uwb`, `hello-world`, ...) and application layer protocols (`telnet`, `mqtt`, ...) are on the top of the diagram in the `user apps` block. Contiki's kernel, its coffee file system `cfs`, network- and MAC protocols, as well as timer and hardware abstractions are part of the `core` block. The `platform` block contains all hardware dependent modules; higher-level modules are hardware-independent.

2.1.2 IPv6

The rising number of IoT devices requires a network protocol with a large address space. IPv4 addresses are exhausted since 2011 [46]. Whilst Network Address Translation (NAT) technology could be used, it breaks the end-to-end principle of the IP architecture. The IPv4 successor (IPv6) is hence the logical choice of a network protocol for IoT devices because of its large address space. The 32 bit long IPv4 addresses allow only about four billion addresses (2^{32}), while an IPv6 address is 128 bit long, which yields 2^{128} different addresses. Apart from providing standardized network access the IPv6 network protocol is required by RPL, which implements routing and the localization mechanism, making IPv6 essential to the NetLoc system.

IPv6 packets, however, cannot be embedded into IEEE 802.15.4 frames, due to their limited frame size. IPv6 mandates a maximum transmission unit (MTU) of 1280 byte. Unfortunately, IEEE 802.15.4 frames support a MTU of only 127 byte.

2.1.3 6LoWPAN

The 6LoWPAN adaptation layer addresses this problem and provides methods for efficient header compression and fragmentation of IPv6 packets. 6LoWPAN allows, amongst others, to compress the 40 byte IPv6 header to a minimum of 3 bytes. To make use of the 6LoWPAN layer the radio must be compatible to the IEEE 802.15.4 standard. Because the DW1000 radio transceiver is IEEE 802.15.4 standard compliant, the NetLoc system can utilize 6LoWPAN and the IPv6 network protocol.

2.1.4 RPL

RPL is the routing protocol for IPv6-based LLNs specified in RFC 6550 [86]. This routing protocol defines packets and methods how to route packets through such a mesh network from any endpoint to any other endpoint. Typically, wireless sensor networks have a mesh topology. At least one node within the network has the special function of a data sink. In IoT applications the sink is typically also a border router, a link between the sensor nodes in the mesh network and the Internet. The RPL routing protocol defines packets and methods how to route packets through such a mesh network from any endpoint to any other endpoint.

The routing topology is tree-shaped with the network's border router in its root. Every node in the network other than the root chooses one neighbor as its parent based on a link metric. Exemplary link metrics are hop count and expected number of transmissions (ETX). An objective function defines how every link must be evaluated based on the link metric. An example of an objective function is the *Minimum Rank with Hysteresis Objective Function* (MRHOF) [40]. Based on the ETX, MRHOF allows RPL to find routes that minimize the ETX between any two nodes within the network. The RPL protocol forms a routing tree that optimizes the objective function.

Topology Configuration. The RPL protocol is designed to form a Destination-Oriented Directed Acyclic Graph (DODAG) topology. Packets are forwarded along directed routes that are loop free (Directed Acyclic Graph). The graph is called destination-oriented, because every node in the network can send packets along a directed path to the sink node.

Because link metrics are expected to change over time, the topology is constantly maintained. That way, the routing tree will adapt to changes like an increase or decrease of the link quality or a node failure caused by a depleted battery.

The DODAG configuration starts with the selection of a root node by the network administrator. Each DODAG instance must have exactly one root node. There are three ICMPv6 RPL control messages defined to configure the routes [86, Chapter 6].

1. The **DODAG Information Solicitation (DIS)** is sent by all nodes except the root to probe the local neighborhood for DODAGs. Nodes receiving a DIS respond with a DIO message.

2. The **DODAG Information Object (DIO)** informs nodes about the DODAG configuration parameters and which parent node to choose. It is sent downwards in the DODAG towards its leafs. A DIO is sent as a response to a DIS or because the **Trickle Timer**[3] expired [55]. It contains a rank field to determine the nodes rank within the tree and to compute parents. The root has the smallest rank, downwards along the routes the rank must increase in order to avoid forming loops.

3. The **DODAG Advertisement Object (DAO)**[4] is sent towards the root to propagate information upwards in the DODAG. The root node requires this information to contact leaf nodes.

Triggered by the Trickle timer, DIOs are sent aperiodically in increasing intervals if the link quality does not change and in short intervals if the links are unstable.

RPL supports two modes of operation, storing and non-storing mode. In storing mode, routing tables are saved locally on the nodes, whereas in non-storing mode IPv6 source routing defines the path.

Storing mode saves bandwidth, whereas non-storing mode saves memory. Once nodes joined a DODAG in storing mode, they send **unicast** DIOs instead of broadcasting DIOs. IEEE 802.15.4 link-local unicast packets are typically acknowledged. This communication pattern matches the one sketched in Figure 2.8. By appending the T_{reply} interval to the ACK packet, the sender can calculate the distance between him and his parent node. NetLoc exploits this property of the storing mode to increase the efficiency of distance measurements. To estimate positions, all distance measurements must be collected at a single point. DAO messages, which are sent to the sink node by all other nodes, are exploited to efficiently collect distance information. By appending a list of $(neighbor, distance)$ tuples to the payload of each DAO packet, the sink node gets all the necessary information to localize the network without sending additional packets.

2.2 Ultra-Wideband (UWB)

A radio signal qualifies as a UWB signal if its bandwidth is larger than 500MHz or larger than 20% of its center frequency. The basic idea is to trade transmit power for bandwidth.

[3] A timer that adapts its period to network density and inconsistent (in this case) routing information. The timer interval is increased or decreased exponentially. It quickly reacts in dynamic networks, but avoids unnecessary traffic on static routes.

[4] Additionally there is the DAO-ACK message. DAO-ACKs are optional, thus not further discussed.

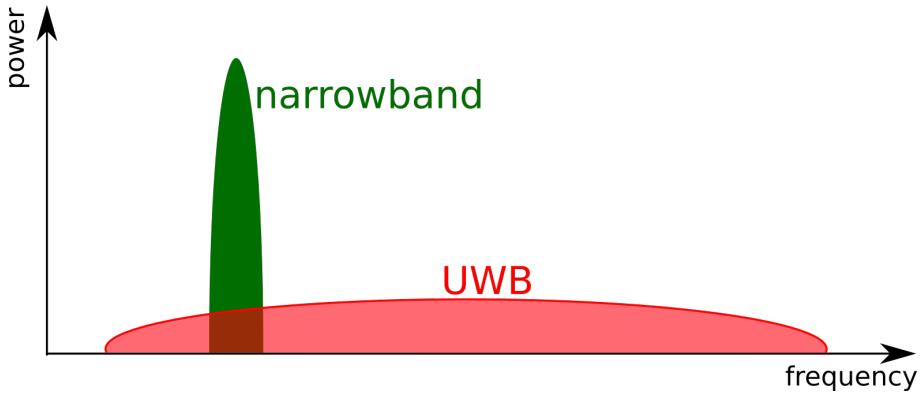


Figure 2.3: Transmit power and bandwidth of a narrowband signal compared to a UWB signal

Instead of transmitting a narrowband signal at a high power, UWB signals are transmitted with a very low power spectral density as sketched in Figure 2.3.

A UWB transceiver has multiple advantages compared to a narrowband transceiver. The most used wireless technologies (WiFi, Bluetooth/BTLE and Zigbee) share the same 2.4 GHz ISM band and are inherently narrowband. Therefore, they are highly susceptible to multi-path fading and cross-technology interference. Due to the low effective isotropic radiated power (EIRP) of less than -41.3dBm/MHz and the short pulses in its baseband representation, UWB signals cause little interference. To narrowband and broadband receivers, these signals have a noise-like character. The high bandwidth makes UWB signals almost immune to jamming.

The UWB transceiver used in the NetLoc architecture is the IEEE 802.15.4(a) compliant Decawave DW1000 (see Section 2.2.1). Its physical layer is defined as High Rate Pulse Repetition Frequency Ultra-Wideband Physical Layer (HRP UWB PHY) in the IEEE 802.15.4(a) standard [47, chapter 16]. Frames consist of a series of a large amount of very short pulses (impulse-radio). The DW1000 supports pulse repetition frequencies (PRF) of $16\,MHz$ and $64\,MHz$. Figure 2.4 shows an exemplary pulse shape having a length of $2ns$, which results in a 500MHz bandwidth. Due to its very short pulses, UWB technology is very robust against multi-path fading. The latter is a serious problem for narrowband transceivers, especially in indoor environments. Another advantage of the HRP UWB PHY is that concurrent transmissions are less prone to cause collisions because of its modulation scheme. The modulation of the data is a combination of binary phase-shift-keying (BPSK) and burst position modulation (BPM). The burst duration is much shorter than the interval that it is in and its position is scrambled according to a time-hopping scheme [47, Chapter 16.3].

Another consequence of the short pulses in impulse radio UWB (IR-UWB) transceivers is a very high time-domain resolution, which allows precise time-of-flight (TOF) measurement of transmitted packets. Knowing the TOF, the distance d between two nodes can be calculated by Eq. 2.1, where c is the speed of light in air.

$$d[m] = TOF[s] \cdot c \left[\frac{m}{s}\right] \tag{2.1}$$

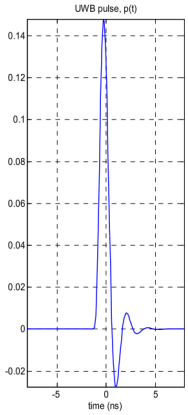


Figure 2.4: UWB pulse shape according to IEEE 802.15.4(a) [47, Figure 16-13].

Channel Number	f_c $[MHz]$	B $[MHz]$
1	3 494.4	499.2
2	3 993.6	499.2
3	4 492.8	499.2
4	3 993.6	1 331.2
5	6 489.6	499.2
7	6 489.6	1 081.6

Table 2.1: IEEE 802.15.4 UWB channels supported by the DW1000 [22], the corresponding center frequencies f_c and bandwidths B. The maximum receiver bandwidth of the DW1000 is 900 MHz, although the transmitter supports higher bandwidths.

2.2.1 Decawave DW1000 UWB Radio

The Decawave DW1000 is the first low-cost IEEE 802.15.4(a) compliant UWB transceiver integrated circuit (IC) [22]. Its features include accurate timestamping of received and transmitted frames. The timestamp precision of approximately 15 ps allows accurate TOF measurements, hence distance calculation errors of less than 10 cm. Its highest supported data rate of 6.8Mbps is faster compared to typical IEEE 802.15.4 radios, which leads to shorter channel acquisition times. The DW1000 supports the 6 RF channels shown in Table 2.1, which center frequencies range from 3.5 GHz up to 6.5 GHz and transmit bandwidths above 1 GHz. These high frequencies make the antenna and platform design very challenging, because circuit geometry, material and parasitic side effects play an essential role. Decawave published multiple guidelines about calibration and hardware design to deal with these problems [20, 21, 23, 24].

2.2.2 UWB Frame Structure

Figure 2.5 shows the structure of a physical frame data unit. The first two fields, preamble and start-of-frame delimiter (SFD), define the synchronization header (SHR). The detection of a valid SFD is the event reflected in the reception timestamp. However, the

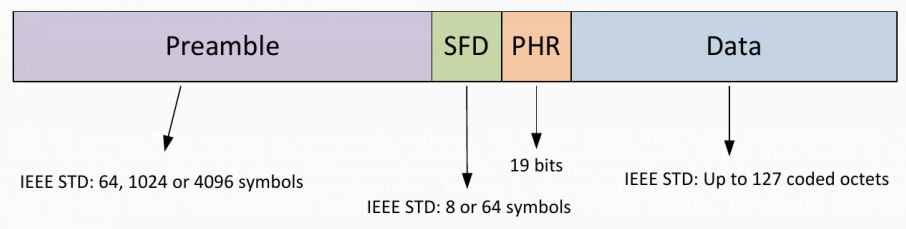


Figure 2.5: UWB PHY frame structure supported by the DW1000 [26, Figure 31].

Bits: 0–1	2–8	9	10	11–12	13–18
Data Rate	Frame Length	Ranging	Reserved	Preamble Duration	SECDED

Figure 2.6: Physical Header Field Format [47, Figure 16-6]

transceiver does not save the timestamp unless the ranging flag in the physical header (PHR) is set to 1. Figure 2.6 shows the fields of the PHR including the ranging flag at offset 9. After the PHR the frame contains the transmitted data. It typically contains a MAC protocol data unit (MPDU) consisting of a MAC header, MAC payload and a two-byte MAC footer that contains the frame check sequence.

2.2.3 Time-of-Flight Measurements

The TOF of a packet is the time difference between the start of a transmission and the start of the reception (t_1 and t_2 in Figure 2.7, respectively). Figure 2.7 shows a sketch of a packet that is transmitted by node A at a time t_1 and detected by a receiver node B at a time t_2. The definition of TOF is given by Eq. 2.2. Node A must send the timestamp t_1 to node B, so that B can calculate the TOF. To this end, both transceivers need a precise clock, the ability to start transmissions at a definite time t_1, and to report the timestamp t_2 of a packet reception. Eq. 2.2 implies that both transceivers have synchronized clocks and the same timebase, which requires a mechanism to synchronize receiver and transmitter.

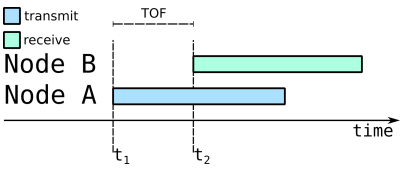


Figure 2.7: TOF of one packet.

$$TOF = t_2 - t_1 \tag{2.2}$$

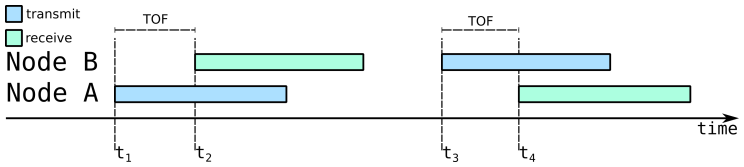


Figure 2.8: Single-Sided Two-Way Ranging.

$$TOF = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} = \frac{T_{round} - T_{reply}}{2} \tag{2.3}$$

2.2.4 Two-Way Ranging

The process of measuring the physical distance between two points based on a signal transmission is called ranging. Two-way ranging means that the signal transmission from which the distance is derived is sent both ways, first from an initiator to a target and then from this target back to the initiator. There are two common methods that do not require a receiver and transmitter to have the same timebase.

Single-Sided Two-Way Ranging. The TOF can be derived from the round-trip times. Figure 2.8 shows a sketch of a communication between two nodes A and B and the calculation of the TOF in Eq. 2.3. Because the time interval T_{round} is measured on node A and the interval T_{reply} is measured on node B, there is no need for a common time base. This is also known as single-sided two-way ranging (SS-TWR). The value of time interval T_{reply} must be sent to node A, who can then calculate the TOF. This type of TOF measurement has one drawback. Because the clock frequencies of node A and node B are not exactly the same, a systematic measurement error is added to the result. In practice, there will always be a clock frequency offset, meaning the clock frequency on one node is higher than on the other node. This phenomenon is also known as clock drift, because the time values of two devices drift apart from each other. Reasons that lead to a frequency offset are for example low accuracy quartz crystals and temperature differences between both nodes [71].

Double-Sided Two-Way Ranging. One way to reduce clock drifts is to use a more stable clock source like a temperature-compensated crystal oscillator (TCXO). Because of its high energy consumption and costs, this is not recommended for battery-powered devices. Another solution is the double-sided two-way ranging (DS-TWR) method, shown in Figure 2.9. An additional message allows the combination of two round-trip times, which reduces the error caused by a clock drift [51]. Node A must send the values of $T_{round,1}$ and $T_{reply,1}$ time intervals in the second transmission, so that node B can calculate the TOF.

2.3 Indoor Localization

State-of-the-art localization systems determine the position of an object in various ways. One way presumes a given radio map with measurable information in it. The most likely position on this map is then calculated based on measurements. An example of such a

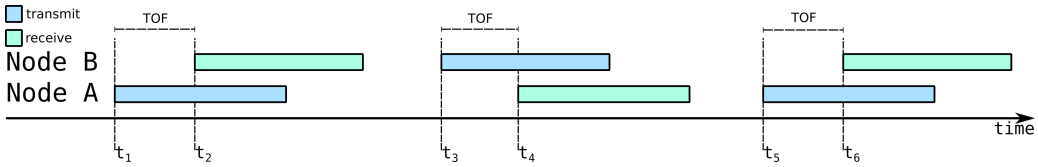


Figure 2.9: Double-Sided Two-Way Ranging.

$$TOF = \frac{(t_4 - t_1) \cdot (t_6 - t_3) - (t_3 - t_2) \cdot (t_5 - t_4)}{(t_4 - t_1) + (t_6 - t_3) + (t_3 - t_2) + (t_5 - t_4)} = \frac{T_{round,1} \cdot T_{round,2} - T_{reply,1} \cdot T_{reply,2}}{T_{round,1} + T_{round,2} + T_{reply,1} + T_{reply,2}} \tag{2.4}$$

system was mentioned in Chapter 1: a map of expected RSS values in an indoor environment is compared to current RSS measurements. A trained classifier then finds the most likely position on that map. Since the classifier is static, it is assumed that the map information does not change over time. This method does not necessarily rely on RSS measurements, but can be applied to magnetic measurements [13], image data (Visual Positioning Systems) [87], Lidar measurements, or a combination of sensor measurements as well.

Another way is to have static anchors deployed at known positions in the environment beaconing their location [61]. The position is then determined by measuring distances and/or angles to these anchors. Localization based on angles is called triangulation, based on distances is called trilateration or multilateration. One refers to triangulateration if localization is based on both, angles and distances. The following sections discuss distance estimations (Section 2.3.1) and trilateration algorithms (Section 2.3.2) in more detail.

2.3.1 Distance Estimation

There are several ways to measure distances. Centimeter accuracy can be achieved with Lidar and ultrasonic-based distance measurements. To minimize the hardware cost and the energy consumption, the preferred way to estimate the distance between two nodes in wireless networks is to use the communication interface, which is in most cases a radio. Distances can be calculated from the RSS or from the time-of-flight (TOF) of a transmission. RSS-based techniques are limited in their accuracy, especially indoors where one has to deal with multi-path fading as well.

If accurate time measurements are possible, a good alternative to RSS-based methods is measuring the time-of-flight. Section 2.2.3 explains this method in more detail.

2.3.2 Trilateration

If distances to anchors and anchor positions are both known, the problem of localization can be reduced to a geometric problem. For two-dimensional localization, the position lies at the intersection of circles, for three-dimensional localization at the intersection of spheres. 3D localization requires at least four anchors to get a unique solution, 2D localization only three. Figure 2.10 shows a sketch of four intersecting circles with anchors (red dots) in their center. The tag's position is marked with a blue dot. In reality the situation of a unique

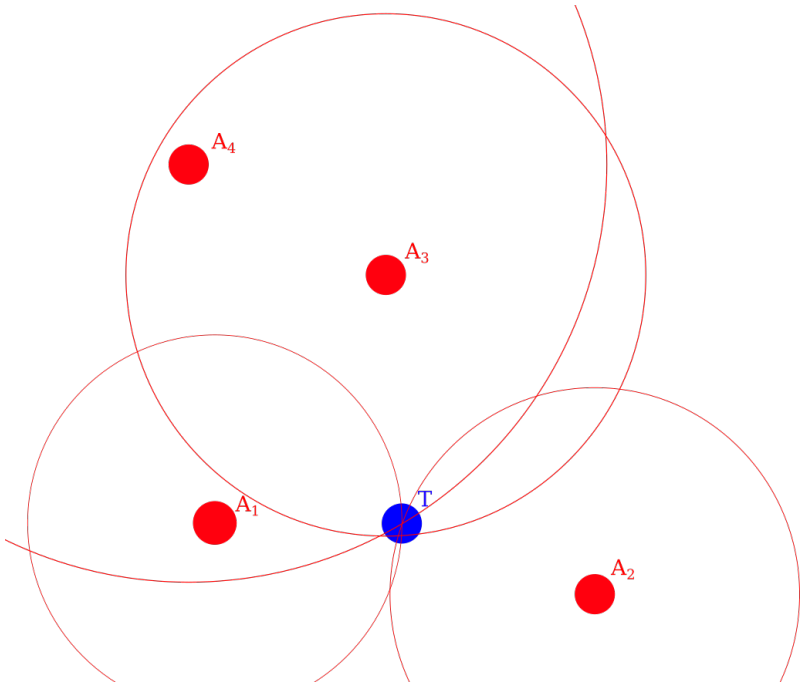


Figure 2.10: Two-dimensional positioning based on four distance measurements. The red dots and circles represent anchors ($A_1 - A_4$) and their distance measurements, respectively. The blue dot is the calculated position of the tag (T), where all circles intersect.

solution where all circles overlap is highly unlikely, because of distance measurement errors. Practical localization algorithms must deal with these uncertainties [85, 89].

2.4 Distance Matrix

In graph theory, a distance matrix consists of distance values between pairs of nodes in a graph. The distance can be a logical value, the weight of the edges, a cost value or a metric. In this thesis, only Euclidean distance matrices are of interest, where elements hold the physical distance in meter between to nodes of a network. A distance matrix D has the following characteristics.

- It is symmetric. For any nodes i and j their distances are equal in both directions $d_{ij} = d_{ji}$.
- All elements are positive.
- The diagonal elements are zero $d_{ii} = 0$.
- If there is no edge between two nodes i and j, their distance is zero $d_{ij} = 0$.

Figure 2.11 shows an example graph and its corresponding distance matrix. The four nodes of the graph in Figure 2.11 have edges to all their neighbors, so only the diagonal elements of Table 2.2 are zero.

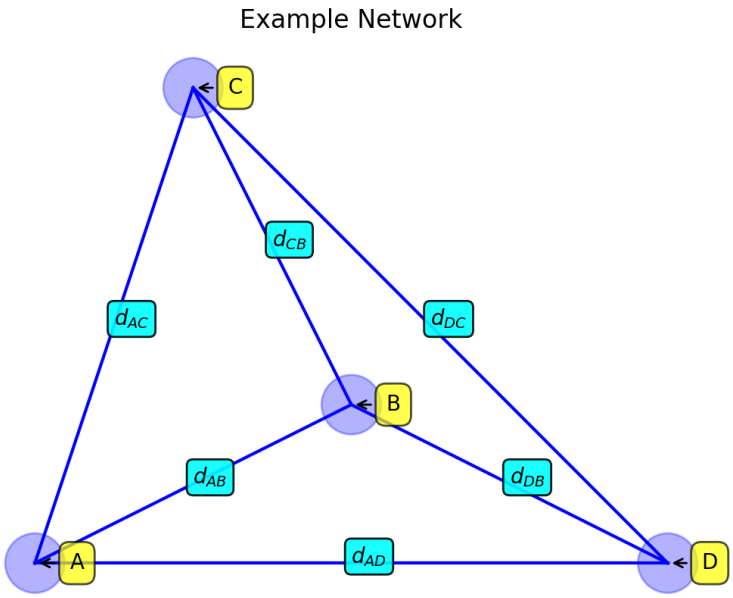


Figure 2.11: Example network graph consisting of four nodes and five edges.

	A	B	C	D
A	0.00	d_{AB}	d_{AC}	d_{AD}
B	d_{AB}	0.00	d_{CB}	d_{DB}
C	d_{AC}	d_{CB}	0.00	d_{DC}
D	d_{AD}	d_{DB}	d_{DC}	0.00

Table 2.2: Distance matrix of the example network shown in Figure 2.11.

Chapter 3

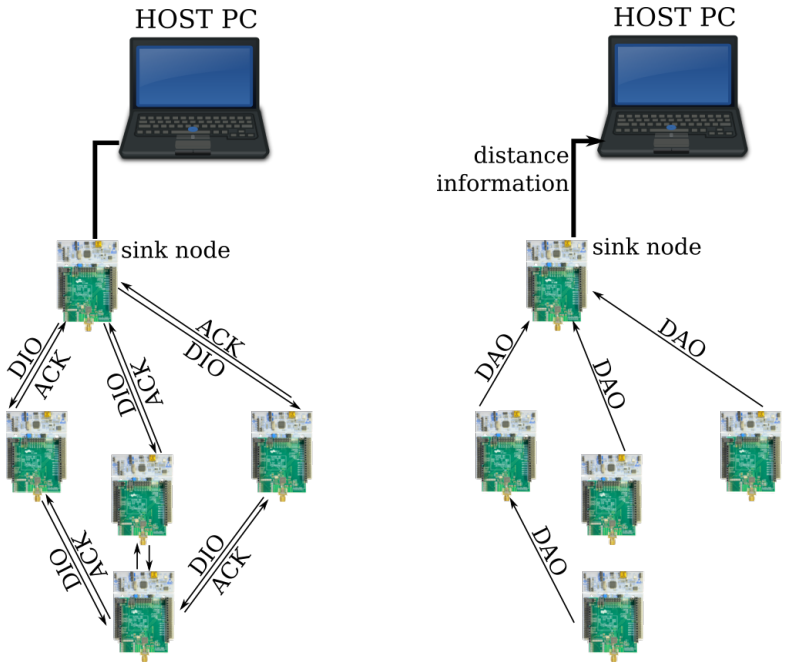
NetLoc Architecture

This chapter presents NetLoc, a system to localize the position of all the nodes in a network and gives a functional overview of its software and hardware components. Section 3.1 illustrates the process of determining and visualizing the position of each node in a multi-hop mesh network. An overview of the system's functional blocks follows in Section 3.2.

3.1 NetLoc: Localization Process

The process to determine a device's position starts with the creation of a new RPL instance by the network's sink node as described in Section 2.1.4. Client nodes start probing their surroundings for existing RPL instances and join a DODAG if one is found. The Trickle timer causes nodes to send DIO packets from time to time. Figure 3.1a shows the exchange of packets during this phase, where DIO packets are sent to neighboring nodes. The latter respond with IEEE 802.15.4 ACK packets containing the necessary timestamps to calculate the TOF and therefore the distance, which matches the communication pattern of SS-TWR as shown in Figure 2.8. Although distance measurements are more accurate when using the DS-TWR method, the SS-TWR has the advantage that it can be embedded in RPL without sending additional packets. In Section 5.1.3 an alternative method to sufficiently increase the accuracy of distance measurements is described. In Figure 3.2 the TOF measurement through modified DIO and ACK packets is presented. Node A initiates a TOF measurement by sending a DIO to node B having the ranging flag set active. When receiving this DIO packet, node B reads the reception timestamp t_2 from its radio and calculates the transmission timestamp t_3. Node B then generates an IEEE 802.15.4 ACK packet and appends the timestamps t_2 and t_3 to it. The ranging flag of this ACK packet is set active and scheduled for the delayed transmission at time t_3. As Node A receives the ACK packet from node B including the timestamps t_2 and t_3, the timestamps t_1 and t_4 are read from the transceiver and the TOF is calculated according to Eq. 2.3. From the TOF, the distance is calculated using Eq. 2.1. As shown in Figure 3.1a, the distance measurement is done by all nodes in the network to collect distance information to local neighbors.

Next, the network's distance information is collected by the sink node. This is done by appending the neighbors addresses and measured distances to DAO packets, which are then sent by all clients to the sink node. Figure 3.1b shows this process of collecting DAO



(a) Measuring distances between pairs of nodes through DIO packet transmissions.

(b) Collecting distance information in the sink node by appending it to DAO packets.

Figure 3.1: Packet exchange to measure the TOF of DIO packets (a) and then collect the distance information in the network's sink by appending it to DAO packets (b).

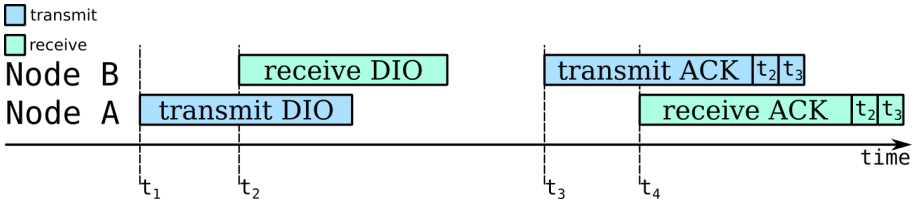
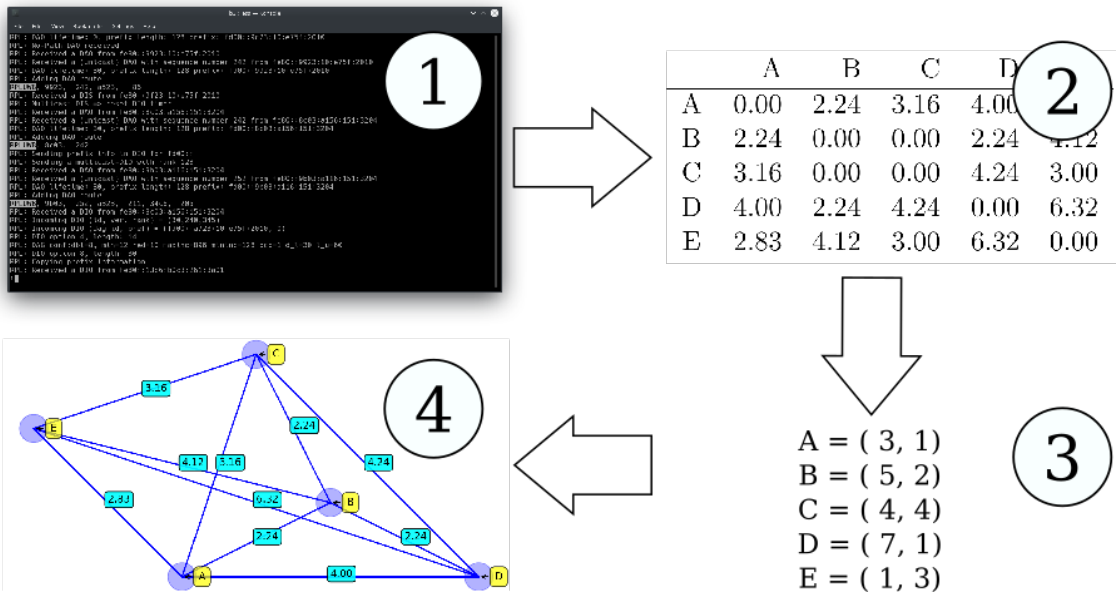


Figure 3.2: Principle of NetLoc to derive distance information. Node A sends a DIO packet to node B with an active ranging flag in the PHR of the packet. Node B recognizes this flag and appends the reception timestamp of the DIO packet (t_2) and the transmission time of the ACK packet (t_3) before sending the ACK.



	A	B	C	D	
A	0.00	2.24	3.16	4.00	
B	2.24	0.00	0.00	2.24	4.12
C	3.16	0.00	0.00	4.24	3.00
D	4.00	2.24	4.24	0.00	6.32
E	2.83	4.12	3.00	6.32	0.00

Figure 3.3: Processing steps for locating nodes of a network based on distance information. (1) parsing the sink node's log file for distance information to (2) create and maintain a distance matrix of the network. This distance matrix is then used by a localization algorithm to (3) calculate each node's coordinates so that the nodes and their links can be (4) visualized.

packets (and therefore the distances information) at the sink node. The sink node forwards the distance information via UART interface to a host PC.

The rest of the localization process takes place on the host PC. The four main processing steps are shown in Figure 3.3. A localization application parses a log file and extracts the network's distance information. Based on the latter, a distance matrix is created and constantly updated as soon as the sink receives new DAO packets. Every time the distance matrix changes, the localization algorithm starts calculating the coordinates of all nodes in the network. Both nodes and their links are then visualized graphically.

Localization and visualization do not necessarily require a map. If no map is provided, the coordinate system of the network will have the sink node in its origin. All other nodes will be located relatively to the sink node's position. Providing a map does not change the localization algorithm, but transforms the whole network in such a way, that it is fitted in the map. In addition to the (optional) map information, three more conditions are required, as explained in Section 5.2.5.

3.2 NetLoc: Building Blocks

As explained in Section 2.1.4, the sink node can act as the network's border router as well. A border router can connect the local wireless network to the Internet by providing an IPv6 address prefix during address configuration that can be routed globally. Each node in the network would then get a global unicast address. This way nodes are able to send locally measured distance information to a data processing center, instead of collecting and

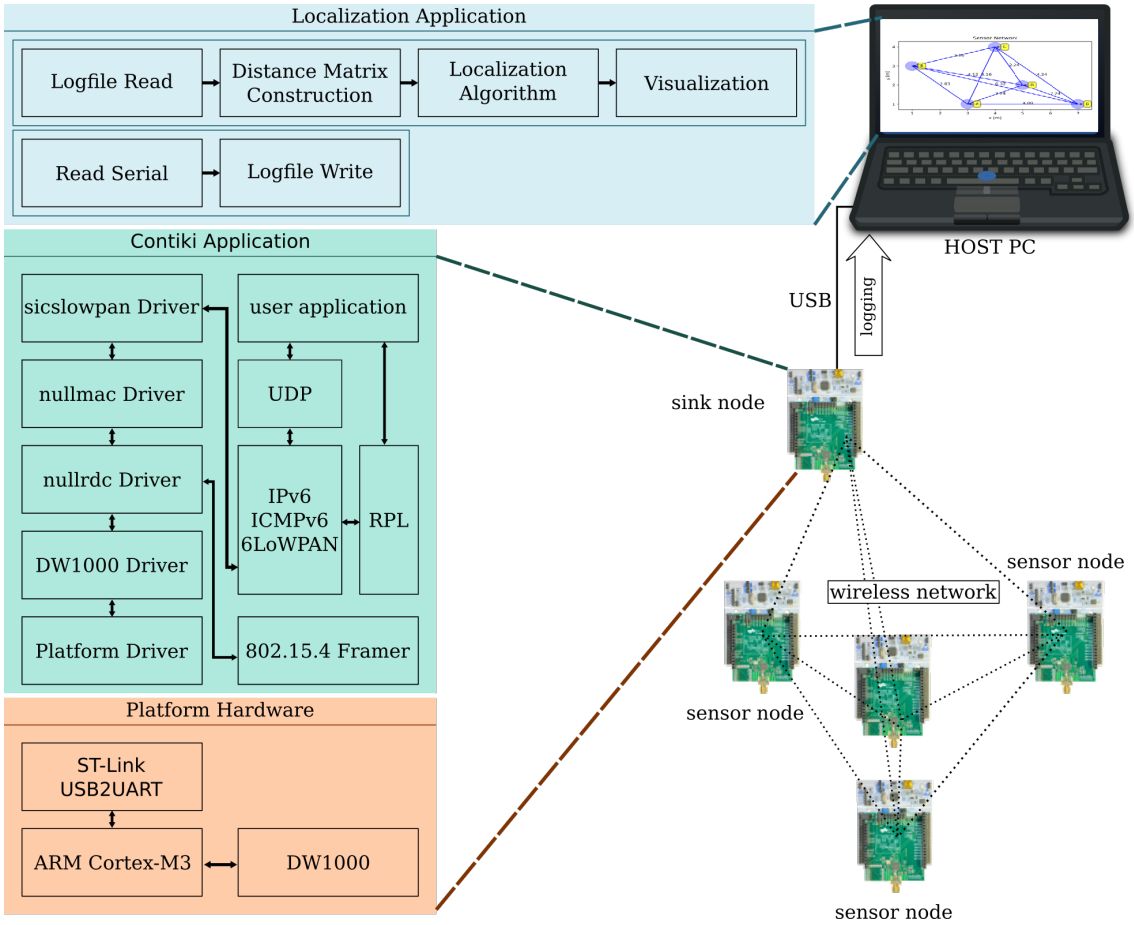


Figure 3.4: Block diagram of NetLoc's architecture.

processing the distance information on a computer in the local network. If the network consists of too many nodes to efficiently process all distance information locally on a single computer, the localization application can benefit from a border router significantly. However, configuring the sink node as a border router was not in the scope of this thesis. The network's distance information is forwarded by the sink node through a tethered serial connection to a host PC. The host PC computes and visualizes the position of each node in the network based on this distance information. Figure 3.4 shows a block diagram of the NetLoc architecture. The building blocks of NetLoc can be grouped into the following three main components.

Platform Hardware. This component contains the UWB hardware, consisting of the NUCLEO-L152RE [76] microcontroller development board and the UWB extension board including the DW1000 radio. The microcontroller board contains the ARM MCU and an independent USB interface converting the MCU's serial UART communication to USB. The Contiki application runs on the ARM MCU, the DW1000 radio is used for communication and TOF measurements and the UART serial bus is used to transfer distance information to the host PC. Section 3.2.1 lists the component's blocks and their functionality.

Contiki Application. Each node in the network runs the Contiki operating system, which is described in Section 2.1.1. There are two different roles that a node can have in the network: the role of a server and the role of a client. The server collects information, the clients gather information to send it to the server. One of these roles must be assigned to each node. A Contiki application performs role assignment and starts the user application process, represented by the second main component as shown in Figure 3.4. The key features that enable localization are embedded in the DW1000 driver and inside the RPL network stack. The DW1000 driver calculates the distances from the measured TOF. Debug output and distance information is written to a UART serial bus. To perform localization, only the sink node's output must be captured. Its blocks are described in Section 3.2.2.

Localization Application. The application implementing the computational-heavy optimization algorithm is illustrated as the top-most component in Figure 3.4. The localization application runs on a PC that is connected to the network's sink node using a USB cable[1]. The localization application's task is to calculate the node positions and to visualize them. The network graph is defined by the distance information in the sink node's log file as shown in Figure 3.3. The sink node's log file contains enough information to locate all node relative to the sink node. Optionally, to improve the visualization of the network, the network can be located in a map coordinate space. If map coordinates are required, the sink node's position on the map and the map itself must be provided upfront. Section 3.2.3 describes the functionality of the localization application in detail.

[1]Note that instead of using a USB cable, the distance information can be forwarded to the host PC via WiFi or Bluetooth as well. It was convenient to use a USB cable, which also powers the sink node in this case.

3.2.1 Platform Hardware

A detailed description of the platform design and its performance is given in Chapter 4. Although multiple UWB-based platforms are available, none could satisfy all the requirements of the NetLoc system, such as the required of low-energy consumption and the requirement of low-cost hardware (the full list of requirements can be found in Section 4.1). The hardware design allows the DW1000 to be used not only for localization, but also as a communication interface. Here, the essential components of the hardware platform are listed.

Decawave DW1000. Section 2.2.1 already gave a short summary of the DW1000's features. It was primarily chosen for NetLoc's platform because it allows precise distance measurements. Furthermore, it performs well in multi-path rich environments, like indoors.

The DW1000 fulfills the requirements for accurate two-way ranging (TWR) as described in Section 2.2.3:

- It has a precise clock. The resolution of the timestamps is 15.65 picoseconds.
- Upon message reception the timestamp is saved and can be read by the MCU.
- Delayed transmission is possible by setting a timestamp that lies in the future. The radio will then wait for its system time to reach this timestamp's value and start transmitting at this exact time. This way, the transmission timestamp can be included in the packet.

ARM Cortex-M3. The STM32L152RE is an ARM Cortex-M3 MCU that exhibits a good trade-off between computational power and energy efficiency. It runs the Contiki application, described in Section 3.2.2.

ST-Link Programmer and USB2UART. The NUCLEO-L152RE [76] board supports three interfaces: a programmer for the STM32L152RE MCU, a debugger and a serial-to-USB converter. It is used as a power supply for the board too. The serial interface is accessed on the host PC via the *Read Serial* block as described in Section 3.2.3.

3.2.2 Contiki Application

This section gives an overview of the functional blocks in the *Contiki application* component of NetLoc as shown in Figure 3.4. All nodes in the network, including the sink node, are programmed with the same application binary. Implementation details are given in Section 5.1.

User Application. This user application implements three processes, one to assign a role to the node, a second one to control the sink node's behavior as a server, and a third one to control a node's behavior as a client. The role assignment process is started automatically after a reset. Based on the node's hardware address, it starts either the server process or the client process next. The server process initializes the node as the RPL root, the client process sends probes to its neighbors and tries to join a RPL network. Each process is described in Section 5.1.1.

UDP. The server process and the client process open UDP connections, allowing the application to exchange messages with other IPv6 nodes through UDP sockets.

RPL. The RPL routing protocol generates control packets to form a routing tree and support a multi-hop topology, as described in Section 2.1.4. RPL maintains the routing table that is used to select a parent from the parent set in order to form the destination-oriented tree structure. Three functions of Contiki's RPL implementation were extended:

1. `dio_output()`: before sending the DIO packet, the *DW1000 Driver* state is changed through its generic `NETSTACK_RADIO.set_value` interface function to enable the ranging features of the DW1000.

2. `dao_output()`: the list of $(neighbor, distance)$ tuples is appended to the regular DAO packet.

3. `dao_input_storing()`: if the DAO packet is received by the sink node, the appended $(neighbor, distance)$ tuples are extracted. The sender's address, the DAO packet's sequence number, and the $(neighbor, distance)$ tuples that list the distances between the sender node and its parent nodes are then forwarded to the host PC to be further processed. Collecting this information from all nodes in the network allows the localization application on the host PC to estimate the each node's positions.

IPv6 and ICMPv6. The blocks *RPL* and *UDP* are based on IPv6 and ICMPv6. The IPv6 neighbor cache keeps track of link-local neighbors, their reachability state, whether or not they can route IP packets, and their IP addresses. In the NetLoc architecture, this neighbor cache plays an important role, as it was extended to save the physical distance to each neighbor.

802.15.4 Framer. All packets that are sent or received by the Contiki application are IEEE 802.15.4 compatible frames. This block is responsible for setting the *ACK-Required* flag in the frame control field (FCF) if the packet has a unicast destination address. Otherwise, the receiving neighbor would not respond with an IEEE 802.15.4 ACK frame. Because of this ACK response, the traffic pattern matches the one for TWR (see Section 2.2.4), which means unicast packets and their ACK responses can be modified to calculate the distance between two nodes.

sicslowpan Driver. This is the network driver of the NetLoc system. It implements 6LoWPAN, making it possible to use IPv6 on this IEEE 802.15.4(a) compliant platform.

nullmac Driver. This is the default MAC driver for developing new applications in Contiki. It is a simple pass-through driver that does not manipulate a packet.

nullrdc Driver. The `nullrdc` driver is the link between the radio driver and higher layers. It checks link layer addresses and accesses the *802.15.4 Framer* to create and parse frames.

The acronym stands for *null radio duty cycle* meaning the radio is not duty cycled and stays switched on. In terms of energy consumption, this is unacceptable for an IoT device. Currently, Contiki does not offer a RDC or MAC layer that is suitable for UWB communication. Medium access control is a challenging task for UWB impulse radios, because clear-channel assessment and carrier sensing techniques cannot be applied on UWB signals. This problem is out of scope of this thesis, but possible solutions are discussed in Section 7.1.3.

DW1000 Driver. This radio driver calls the DW1000 API functions to configure and control the DW1000 UWB transceiver. The driver was ported from the EVB1000 platform implementation [15]. It accesses the transmit and receive buffers on the radio IC to actually send and receive data. The procedure for transmitting data packets changes if ranging is active. A flag in the transmit control register sets the ranging bit in the physical header, so the receiving node recognizes a ranging frame and remembers its reception timestamp [26].

Reception of frames is signaled by a hardware interrupt. An interrupt service routine (ISR) takes care of error checking and executing the proper callback functions in the driver. In case the ranging flag was set, the type of frame is checked and immediately processed. This is necessary because longer response times would increase the measurement error [19]. Regular data frames are copied to Contiki's packet buffer so higher layers can continue processing it. Detailed information of the driver port and its modification is given in Section 5.1.3.

Platform Driver. The platform driver includes: CPU specific implementations, board specifications, pin and bus assignments, interrupt assignments, timer initialization, the Cortex Microcontroller Software Interface Standard (CMSIS), the Nested Vector Interrupt Controller (NVIC) interface, the hardware abstraction layer (HAL) of the board, the button driver, the LED driver, UART, as well as SPI and I2C interface functions. Towards this goal, the Contiki implementation of the Spirit1 platform, which uses the same NUCLEO-L152RE MCU board, was ported to the NetLoc system [77].

The following segments of the platform driver are essential to the NetLoc system:

- The SPI bus that connects the STM32L152 MCU to the DW1000 radio IC is initialized.
- The DW1000 interrupt pin is linked to a callback function and its priority is configured.
- UART is configured to enable serial communication between a host PC and the MCU. A boot message including the node address is printed on the serial bus, which triggers the localization application to start logging to a new file (see Section 3.2.3). Furthermore, during runtime, the distance measurements are printed to this bus interface.
- The DW1000 radio is initialized with its default configuration.

Section 5.1.4 lists the details of the platform port.

3.2.3 Localization Application

The localization application consists of two separate processes. Its building blocks are grouped together in Figure 3.4. One process gets its input from the serial data output of the network's sink. After every hardware reset of the sensor node, the process creates a new log file with the nodes address as the filename. The task of the second process is to parse that log file and run a localization algorithm based on the information in it. Both processes can run simultaneously or sequentially.

Logfile Write. The first task of this block is to create a new log file every time the sink node reboots. During the boot process, it prints the node's address, which will be extracted to generate the log's filename. The block's second task is to append data to the log file and keep it updated as new measurements are received. Section 5.2.2 describes this block in more detail.

Read Serial. The sink node will print all its output to a serial bus. At the host PC the serial device must be configured with the correct settings so the output can be decoded correctly and passed on to the *Logfile Write* block. Section 5.2.1 contains details about the interface configuration.

Logfile Read. Reading the logfile can be done on a previously saved file, or online while it is written. The task of this block is to scan the log file line by line and to extract relevant information in a way that allows further processing. Node addresses are saved as a list of strings, distances as floating point variables in the distance matrix. The implementation of this block is explained in Section 5.2.3.

Distance Matrix Construction. The sink collects measurements from each node separately and combines each measurement in a distance matrix. The distance matrix D is a way of representing the network graph, where each element d_{ij} holds the distance between the nodes i and j as explained in Section 2.4. Each time a new measurement is received, the matrix is updated. Section 5.2.4 describes how this process is implemented.

Optimization Algorithm. The algorithm calculates all the coordinates relative to the sink node's coordinates. If it is required to visualize the network on a map, that map must be provided upfront. To transform the coordinate vectors into a map space, additional information is needed: (i) the displacement vector of the sink node, (ii) the rotation angle between both spaces and (iii) the relative location of three points to each other. The latter is needed to check if the transformation includes mirroring. If all information is given, the algorithm calculates the map coordinates for all nodes. The algorithm is explained in detail in Section 5.2.5.

Visualization. The visualization function creates a graphical representation of the sensor network on a two-dimensional plot. The following details are shown in the plot:

- One dot for each sensor node that has been located. The address of each node is displayed next to its position.

- The links between nodes drawn as edges of the network graph, including their distance in meter.
- Optionally, a map including important obstacles like walls. A map is not required for localization, but it can be added the visualization.

The input for this block is a list of all nodes and their coordinates, a list of all edges and their lengths, and optionally, map information to be included in the plot.

Chapter 4

Hardware Design

This chapter presents the hardware design of NetLoc's mobile devices. Typical hardware properties of constrained IoT devices were already discussed in Section 2.1. In Section 4.1 the specific hardware requirements to realize the NetLoc system, many of which generally apply to IoT devices as well, are described. An analysis and evaluation of existing platforms regarding the requirements is given in Section 4.2. In Section 4.3 a new platform design is presented that meets the requirements listed in Section 4.1. As part of the new platform a printed circuit board (PCB) design was developed including the Decawave DW1000 UWB radio transceiver, which is the first available low-cost IR-UWB radio transceiver[1]. This new board design is validated in Section 4.4, proving that its RF performance is equivalent to Decawave's own reference design.

4.1 Requirements

The following list of requirements ensures an IoT-driven, low-cost hardware design, which can be upgraded in the future and adapted to specific use cases, while avoiding an isolated solution resulting in vendor lock-in.

Low-energy. The platform should be energy efficient enough to be powered by a small battery for a reasonable long time. This is a common requirement for devices employed to build IoT applications.

Individual power measurements. Since energy efficiency is a requirement, it is also of interest to measure the energy consumption of individual platform components. The requirement is to analyze each individual platform component (like an MCU or a radio) separately.

Low-cost. Instead of a few powerful and expensive computing devices, IoT applications require a high number of cheap devices. This requires a single device to be low-cost.

[1] By the time this thesis was written, the DW1000 was the only available IR-UWB transceiver.

Open-source. The hardware design and the firmware has to be shared with the research community. Researchers should profit from this work and accelerate their own development of UWB-based applications. Free and full access to all design sources also simplifies replication of our measurement and test results.

IoT OS support. The platform should support a common IoT OS. The open-source operating system Contiki is preferred [1], as it allows us to re-use implemented communication protocols such as IPv6, RPL and UDP.

Multi-purpose solution. Existing platforms use specialized designs (e.g., tag or anchor) depending on the role within the network [10, 67, 73, 83]. Some are fixed anchors with a tethered power supply. Others are designed to be battery-powered to around freely. Such a design is inflexible in dynamic and mobile networks and prone to failures. Furthermore, it prevents a dynamic role assignment of individual nodes, since specific roles (like anchor or tag) are restricted to specific hardware designs. Therefore, the platform should be a multi-purpose solution.

Exchangeable antenna. Some applications require directional antennas, while others rely on an omni-directional antenna [41]. Supporting one single antenna is a considerable constraint for wireless devices. Therefore, the antenna of the transceiver should be exchangeable. By providing an SMA connector it is possible to select the most suitable antenna type for each application individually. In a real-time location system (RTLS) two types of nodes exist: anchors (with a static, known position) and (tags with a mobile, unknown position). While an omni-directional antenna is a good choice for mobile nodes, anchors are often fixed on walls or in corners. An anchor's performance can be improved by using a directional antenna.

Expandability. It can be cumbersome to replace or change existing hardware designs. Therefore, it should be possible to extend the platform with additional sensors. In positioning applications, an inertial measurement unit (IMU) and a barometer are beneficial to achieve higher accuracy. The design should allow adding (and removing) hardware components on demand.

Interface for data logging. IoT links are known to be lossy. Therefore, a standalone logging option is required to overcome loss of sampled sensor data. Furthermore, to re-run and analyze experiments offline, it is required to store measurement data. This logging option can be an IEEE 802.11 transceiver as a back-channel or a local flash memory.

These requirements can be grouped into three fields of interest: IoT-specific, research-specific and positioning-specific (columns of Table 4.1). For IoT applications the requirements IoT OS support, low-energy and low-cost must be met. For research purposes the possibility for individual power measurements, open-source designs and exchangeable antennas are required. An exchangeable antenna, expandability and a data logging interface are in the interest of positioning systems. To support further research in the field of

IoT	Research	Positioning
IoT OS support	Open-source	IMU (9 DoF), barometer, ...
Low-energy	Individual power measurements	Standalone logging option
Low-cost	Multi-purpose solution	Multi-purpose solution
	SMA antenna connector	SMA antenna connector

Table 4.1: Requirements for the UWB evaluation platform.

	EVB1000	OpenRTLS	Ciholas
IoT OS support	✓	–	–
Low-energy	–	✓	–
Low-cost	–	–	–
Open-source	–	–	✓
Individual power measurements	✓	–	–
Multi-purpose solution	✓	–	✓
Exchangeable antenna	✓	–	✓
Expandability/IMU	–	✓	✓
Interface for data logging	–	✓	–

Table 4.2: Requirements coverage of the platforms EVB1000, OpenRTLS and Ciholas.

location-aware IoT applications, a platform is needed that fulfills all the listed requirements.

4.2 Analysis of Existing DW1000-Based Platforms

There exist numerous platforms integrating the DW1000 UWB transceiver in their design. The applications of these platforms are industrial [67, 73], scientific [54], for hobbyists [10] and embedded smartphone systems [70]. I evaluated the eligibility of all platforms in terms of the requirements summarized in Table 4.1.

All analyzed platforms focus on localization applications. The UWB transceiver was hardly used as a communication interface to exchange data, except by Sewio [73]. The platforms in [52, 66, 73, 83] use the DWM1000 module[2]. The platforms in [10, 14, 54, 59, 67] use their own RF design. The following three platforms seemed most suitable for research and development, and for experimenting. Table 4.2 shows which requirements were met by these platforms.

EVB1000 [59]. This is Decawave's own reference design for the DW1000. It allows to measure the current of the DW1000 individually. The platform is a multi-purpose solution, requiring no specialized second board design for positioning. It has a SMA connector allowing to exchange the UWB antenna. Contiki has been ported to this platform and a radio driver for the DW1000 was implemented [15]. The current drain of the microcontroller ($47\,mA@72\,MHz$ in active mode and $1.2\,mA@125\,kHz$ in idle mode) is relatively

[2]The DWM1000 [27] is a module that embeds the DW1000 and includes a non-exchangeable, small, omni-directional chip-antenna.

high for an IoT device. The cost of 473.11 EUR[3] for 2 boards makes it too expensive for large-scale networks. It has no interface for data logging and it does not provide connectors to add sensors like an IMU to the existing design.

OpenRTLS [67]. This platform focuses on real-time positioning in industrial and medical environments. The tags run on batteries and their micro-controller has a low power consumption of 24 mA in active mode and 2 μA in idle mode. They include an IMU and a barometric sensor in their design as well, but no data logging interface. UWB antennas can be exchanged on anchors, but not on tags. Anchor nodes require a tethered power supply and embed a second wireless interface (2.4 GHz, IEEE 802.15.4 compatible) as well as an Ethernet connector. Both designs are not supported by an IoT OS. While the tag (uNemo) costs only 43 EUR, the anchors are more expensive (375 EUR). This platform is unsuitable for research, because neither its hardware design nor its firmware is open-source. Power measurements of single components are not supported and multiple hardware designs are used instead of a multi-purpose platform solution.

Ciholas [14]. This platform provides all hardware design files of its DWUSB board, but no firmware sources. The board has an SMA connector for the UWB antenna, an integrated IMU and a pressure sensor. Although the application focuses on RTLS, there is no separate design for anchors. The DWUSB is a multi-purpose solution. The costs per board of 180 EUR are too high to build large IoT networks. Its 120MHz ARM Cortex-M4 MCU has an idle mode current drain of 6.9 mA and 38.0 mA in active mode. For IoT devices this energy consumption is not efficient enough. Furthermore, there is no IoT OS support for this platform and the latter has no interfaces for data logging. Individual power measurements of this board's components are not supported by its design.

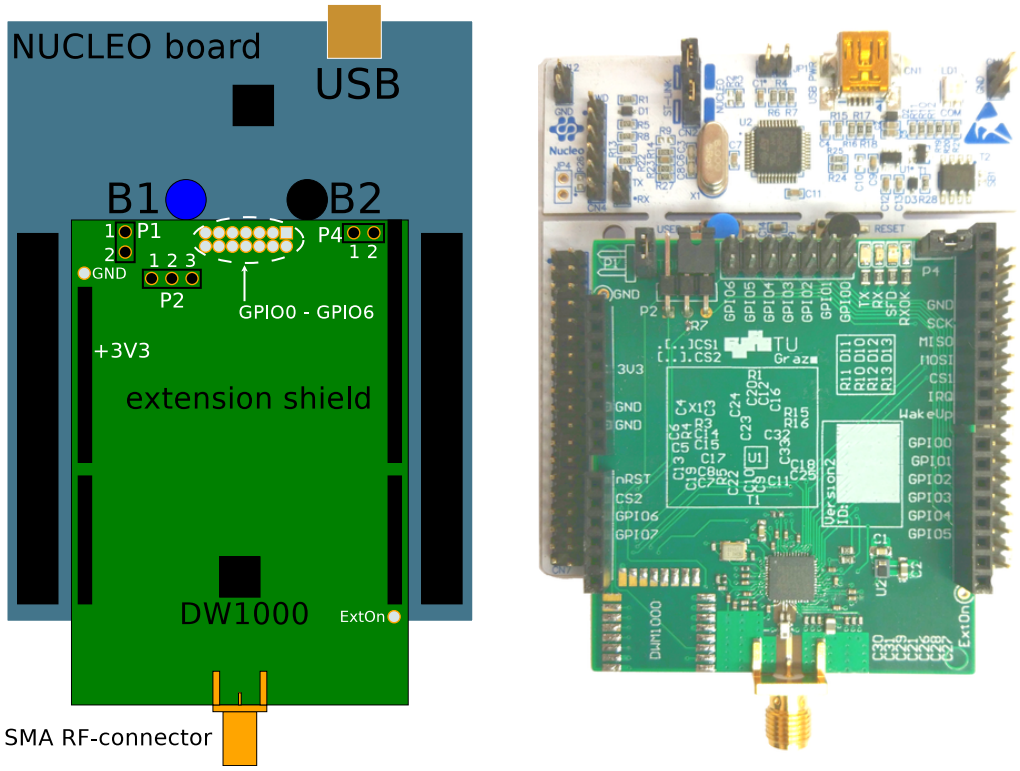
Since none of the analyzed platforms fulfilled all the requirements, a new platform was designed. The resulting modular platform design allows faster future research on UWB. The platform design is described in Section 4.3. Its RF performance is equivalent to the EVB1000, Decawave's own reference design. Details about the PCB design of the UWB board are described in Section 4.3.2. The calibration results of the UWB board are shown in Section 4.4.2.

4.3 Design

Section 4.3.1 lists all requirements from Section 4.1 and describes how they are met in our design. The UWB platform consists of the following main components:

- NUCLEO-L152RE development board
  This board from ST Microelectronics combines a ST-Link programmer with an ARM Cortex-M3 STM32L152RE MCU evaluation board. The NUCLEO-L152RE board embeds the MCU, programming and debugging interfaces, a USB connector, the voltage conversion and power supply circuits. For user inputs, the board provides a push-button (B1 in Figure 4.1a). Its state is readable through a GPIO pin on the MCU. To reset the MCU, there is a second push-button (B2 in Figure 4.1a).

[3] Distributor www.digikey.com



(a) Block diagram of the UWB extension shield stacked on top of the NUCLEO-L152RE board.

(b) Picture of the UWB extension shield stacked on top of the NUCLEO-L152RE board.

Figure 4.1: The UWB extension shield stacked on top of the NUCLEO-L152RE board.

- UWB extension shield
  This is a new design of an extension board compatible with the Arduino header format. It provides access to the DW1000 transceiver and includes an RF design to a 50Ω SMA connector. The specifics about the design of this board are described in detail in Section 4.3.2.

- X-NUCLEO-IDW04A1 [74] *(optional)*
  It extends the platform with the 802.11 b/g/n transceiver SPWF04SA and a micro-SD card slot. This extension board's useful features are optional and not required in the NetLoc system, therefore it is not used in this thesis.

- X-NUCLEO-IKS01A1 [75] *(optional)*
  This motion MEMS extension shield includes a 3D accelerometer, a 3D gyroscope, a 3D magnetometer, a pressure sensor (barometer), a humidity sensor and a temperature sensor. This board's senors are especially useful for localization systems as discussed in Section 7.1. Although the hardware design supports this board, the software implementation and the use of these sensors was out of scope for this thesis.

Figure 4.1 shows a block diagram (Figure 4.1a) and a picture (Figure 4.1b) of the UWB platform without the optional boards. The NUCLEO-L152RE board can be extended

through connectors that are compatible to the Arduino format [3], as well as through the less flexible but more powerful MORPHO connectors. Additional hardware for programming the MCU is not necessary since an ST-Link programmer is already included. Another advantage is the flexibility to switch to different, compatible boards of the same product family (NUCLEO64), if necessary. If the application demands more computational power, more memory, or less energy consumption, the board can be exchanged with one that better fits the application requirements. Other extension shields can be stacked on top of the UWB board to complement the existing platform. This choice of hardware components sufficiently fulfilled our requirements from Table 4.1. The following detailed description explains the design decisions in the context of the requirements.

4.3.1 Fulfillment of the Requirements

IoT OS Support. The Contiki operating system supports a variety of different processors and IoT platforms. The choice to use the STM32-based Nucleo board NUCLEO-L152RE brings multiple benefits, one of them being the support for both the STM32L152RE ARM-based Cortex-M3 CPU and the NUCLEO-L152RE platform. The driver for the DW1000 transceiver is not included in the official Contiki repository. However, an implementation exists [15].

Low-energy. Of all hardware components in wireless sensor nodes, the radio transceiver has the highest energy consumption. The chosen UWB transceiver DW1000 is no exception: in some operational modes, the average current consumption exceeds $150mA$. For small battery powered devices this is a problem. To reduce the DW1000's overall power consumption, an external DC-DC converter is used as proposed in [22, section 7.2]. A crystal oscillator without temperature compensation was used, because of its higher energy efficiency.

The Cortex-M3 CPU of the NUCLEO-L152RE is low-power and supports sleep states as well as deep-sleep states [62]. The CPU has a current draw of $7.55\,mA@32\,MHz$ in active mode and $21.5\,\mu A$ in low-power sleep mode, which makes it suitable for battery-powered IoT devices.

The DW1000 supports driving LEDs on four GPIO pins to indicate certain states of operation (transmitting, receiving, preamble received and start-of-frame delimiter received). This is useful for debugging, but it consumes more energy. Therefore, the LEDs are included in the design but can be disabled by removing jumper P4 (top-right corner of the extension shield as shown in Figure 4.1a). If there is no need for the LEDs on the platform, they can be left out on the board without affecting the functionality.

The choice of using an external UWB antenna has an impact on energy efficiency as well as a higher antenna gain can increase the quality of a wireless communication link at the same energy budget.

Low-cost. The NUCLEO-L152RE platform is a low-cost development platform that requires no additional programmer. The ST-Link software used to flash the micro-controller is open-source and can be downloaded from [78]. The firmware and the Contiki OS are open-source too. Components like IMUs, IEEE 802.11 transceiver or other sensors can be added to the platform on demand and can be left out to reduce the overall cost of the

platform. The PCB manufacturing cost of 17.52 EUR[4] accounts for the largest portion of the overall platform cost. The components per board sum up to 29.90 EUR[5] (a detailed list of all components can be found in Appendix D). A platform without the optional WiFi and MEMS boards would cost 47.42 EUR. Including the WiFi board (18.79 EUR) and the MEMS board (12.51 EUR) the platform costs 78.72 EUR.

Open-source. The software ST-Link, which is required to program and debug the NUCLEO-L152RE board, is released under the BSD license. The Contiki OS source code is released under the Contiki open-source license. The hardware design of the UWB extension shield was created using the Altium Designer software [57]. All design files and Gerber outputs are open-source as well. The UWB extension shield is designed to fit on the Arduino header, a common development platform that is also open-source hardware.

Individual power measurements. The UWB extension shield was designed to allow power measurements. To measure the current consumption of the UWB extension shield, one can connect a current probe to jumper P1 (shown in Table A.1 and Figure 4.1a). In default mode, the jumper P1 must be shortened to connect the $+3.3\,V$ supply pin from the Arduino header and the VCC plane from the UWB shield.

To measure the supply voltage of the extension shield, the voltage probe must be connected to the $+3.3\,V$ supply pin and to the GND pin. To connect different types of probes, the GND test point next to the P1 jumper can be used.

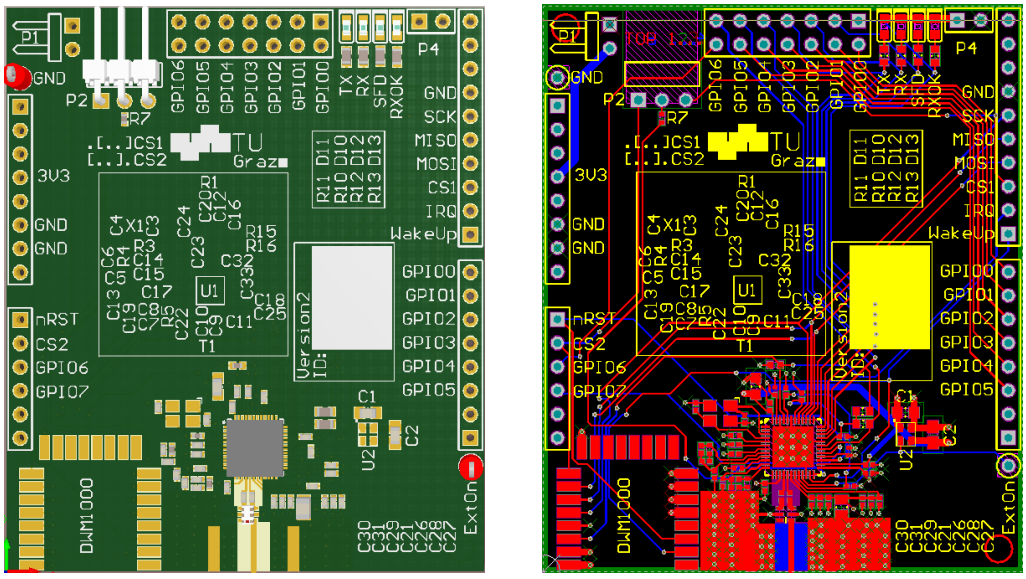
Multi-purpose solution. The extension shield can be used in combination with different other MCU-boards. Nevertheless, it is possible to build sensor networks with wireless sensor nodes that all consist of the same hardware components. One is not forced to use specialized designs for different roles (like anchors or tags) in the network. If the application demands different hardware components in different nodes, the same UWB extension shield can be used in every sensor node. The SMA antenna connector simplifies the multi-purpose solution.

Exchangeable antenna. The SMA-RF connector allows experimenting with all antennas that have a female SMA connector. The user can choose an antenna design depending on the application. By using an RF switch, one can use multiple antennas in the same design [41].

Expandability. The decision to use the DW1000 on an extension shield was made to expand the hardware later on. The platform's micro-controller can be changed without altering the UWB extension shield by stacking the extension shield on different MCU boards. The chosen MCU board family (NUCLEO) offers two separate header formats for stacking - ST's own MORPHO header format and the widely used Arduino-compatible header format. The advantage of the Arduino header connectors is that it is supported by a large community. The UWB extension shield is designed to allow stacking of other

[4] 10 pieces costed 175.20 EUR. The price heavily depends on the ordered quantity. The cost of ordering a single PCB is 126.42 EUR.

[5] We ordered all components at www.mouser.com.



(a) Top view of UWB extension shield.

(b) PCB Designer view of UWB extension shield.

Figure 4.2: UWB extension shield design.

extension shields. Stacking two UWB extension shields is considered in the design and described in Section A.1. The P2 header (shown in Table A.1 and Figure 4.1a) was added to support two different SPI chip-select lines (SPI-CS). This way, the same SPI bus can be used by another shield that is stacked on top of the UWB extension shield. The header pins of the serial UART and I2C buses were left unused in the UWB extension shield, so that other extension shields can make use of them. In particular the WiFi shield [74] and the sensor shield [75] by ST are supported in combination with the UWB extension shield.

Interface for data logging. An interface for data logging can be stacked on top of the UWB extension shield. This can be a reliable second wireless interface like IEEE 802.11, or an offline mass storage to collect all test data for offline analysis. The WiFi shield X-NUCLEO-IDW04A1 [74] supports both by providing access to an on-board micro SD card.

4.3.2 Production Specifics of the UWB Extension Shield

Figure 4.2 shows our design of the UWB extension shield with Figure 4.2a showing the top view. In Figure 4.2b the copper elements of the top layer are colored in red and the copper elements of the bottom layer are colored in blue.

Decawave provides its own design guide for integrating the DW1000 in a hardware design [58]. This hardware design guide and Decawave's own reference design for the DW1000 (the EVB1000 evaluation platform [59]) were kept in mind for the UWB extension shield design.

As discussed earlier, the manufacturing of the PCB is the main cost factor. The

following parameters affect the PCB cost and should hence be considered:

- Size (area);
- Number of Layers;
- Quantity;
- Surface finish;
- Track width;
- Via diameter;
- Material of dielectric;
- Color of solder stop;
- (optional) Solder stencil[6].

Although the cost should be minimized, we decided to make the following trade-offs to ensure a high quality design.

Track width. A smaller track width leads to higher production costs. The minimum track width of the PCB design is limited by the pad width of the DW1000, which is $200\mu m$.

4-Layer layout. The choice of a 4-layer layout instead of a cheaper 2-layer layout had two reasons: (i) to decrease the width of the RF transmission lines (described in Section 4.3.6) and (ii) to simplify a uniform power supply while placing the components close to each other. The placement of the decoupling capacitors is crucial to the performance of the DW1000. A 4-layer layout allows to place them very close to the DW1000's pins. The second and third layer in the 4-layer layout are assigned to be a VCC plane (positive voltage power supply) and a GND plane (negative voltage power supply), respectively. That way, these two nets are accessible through vias anywhere on the PCB. Section 4.3.4 gives detailed description of the layer stackup.

Board size. A larger area increases the costs of the extension shield. The minimum width of the PCB results from using the Arduino header connector, which is $50.26mm$. To have a safe distance between the SMA connector, the RF transmission lines and the underlying NUCLEO-L152RE board, the length of the board is $60mm$. That is $10.74mm$ longer than the minimum length forced by the Arduino header.

Electroless Nickel Immersion Gold (ENIG). This type of surface plating is more expensive compared to other techniques. One of its advantages is the constant surface planarity. The surface thickness must be considered when calculating the dimensions of the RF transmission lines. A deviation of the surface thickness would lead to an incorrect width of the transmission lines, hence changing the transmission line impedance.

[6] When soldering in a reflow-oven, a stencil is helpful to deposit the soldering paste onto the SMT pads.

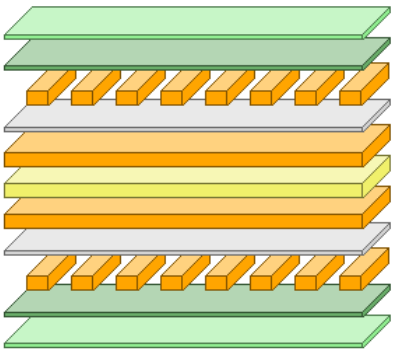
Manufacturer. We chose the manufacturer Multi-CB [63] to produce our PCBs. Although the layer stackup recommended by Decawave [58, Figure 5] was not offered, an individual layer stackup was possible. Detailed information about thickness and material of each layer was available and was considered in the calculation of the transmission line width. The layer stackup is described in detail in Section 4.3.4.

Components placement. Possible positions for the antenna port were the top or the bottom edge of the PCB. The bottom edge was chosen to increase the UWB antenna's distance to all other electronic components. Placing the DW1000 in the middle of the board would allow good access to all 48 of its pins. To keep the length of the transmission lines as short as possible (and therefore the chance of impedance deviations), the DW1000 was instead, placed closer to the bottom edge as shown in Section 4.3.6. One design aspect that was not obvious before we soldered the UWB shields was the correct alignment of the stencil. The pads of R7 (close to P2) were very helpful to align the stencil on the PCB.

4.3.3 Custom SMA Footprint Library

The SMA connector must not be through-hole mounted, but should be edge-mounted. A through-hole mounted connector would add an unwanted stub to the transmission line. In our design the footprint of the SMA connector is customized to avoid impedance discontinuities. The pad width of the center pin that carries the signal is matched to have the same width as the transmission line. Furthermore, the pad length of the signal pin was increased until the edge of the PCB so that there is no gap between the transmission line and the PCB edge.

4.3.4 PCB Layer Stackup



Name	Material	Thickness [μm]
Top Overlay	Ink	
Top Solder	Solder-Stop	10
Component Side	Copper	35
Dielectric	2 x Prepreg 7628	180+180
Ground Plane	Copper	18
Dielectric	FR-4 Core	700
Power Plane	Copper	18
Dielectric	2 x Prepreg 7628	180+180
Solder Side	Copper	35
Bottom Solder	Solder-Stop	10
Bottom Overlay	Ink	

Table 4.3: Thickness and Material of the 4-Layer PCB Stackup [57].

Table 4.3 shows the layer names, the chosen FR4 types and their thickness. In total, the PCB has a thickness of $1526\,\mu m$. A symmetric layer stackup was demanded by the manufacturer. Especially the dielectric between the component side, where the transmission lines are located and the underlying GND plane is of interest. Using two layers of

Prepreg 7628 with a combined thickness of $360\mu m$ came closest to the recommendation in [58]. The manufacturer Multi-CB also provided information about the Prepreg that will be used for production. In our case, this was the DE104 by Isola Group [42]. According to the DE104 datasheet, the **permittivity** at $5GHz$ is $\epsilon_r = 4.32$. These two values - the thickness and the permittivity - will affect the dimension of the transmission lines (as described in Section 4.3.6).

4.3.5 Placing Decoupling Capacitors

The decoupling networks must fulfill two purposes. First, they must filter noise and other high frequency signals. Second, they act as energy buffers. To effectively filter high frequency signals, the capacitance should be small and the component must be placed close to the signal source.

The current path must be considered when placing the capacitors. A decoupling is only possible if the pads of the capacitor are positioned between the power source and the DW1000 pad. Figure 4.3 shows capacitor C22 to illustrate the correct placement of decoupling capacitors.

The power amplifier pins VDDPA1 and VDDPA2 of the DW1000 are decoupled by a network of 7 capacitors to avoid noise propagation to other parts of the board. See [58, chapter 6.3] for a detailed explanation on how to place them.

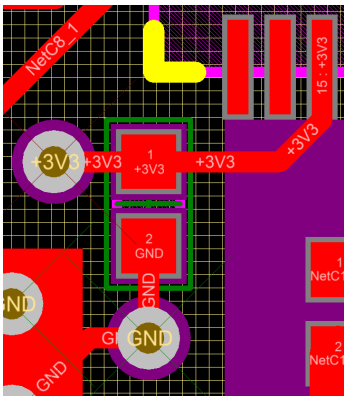


Figure 4.3: Correct decoupling capacitor placement.

4.3.6 Transmission Line Design

Designing the transmission lines can be challenging, because even small miscalculations could have a negative impact on the RF performance. We included a backup solution in our PCB design in case of unforeseen problems, which is described in Section 4.3.7.

Figure 4.4 shows the transmission lines on the UWB shield. The differential transmission lines connecting the DW1000's RF_P and RF_N pins to the capacitors C9 and C10 must have 100Ω. The second differential pair connects the capacitors to the UWB Balun and must have 50Ω. Finally a 50Ω single-ended transmission line connects the Balun to the SMA signal pin. There is a rule of thumb in RF design saying that a connection in a network must be treated as a transmission line if it is longer than $\lambda/10$. Assuming the

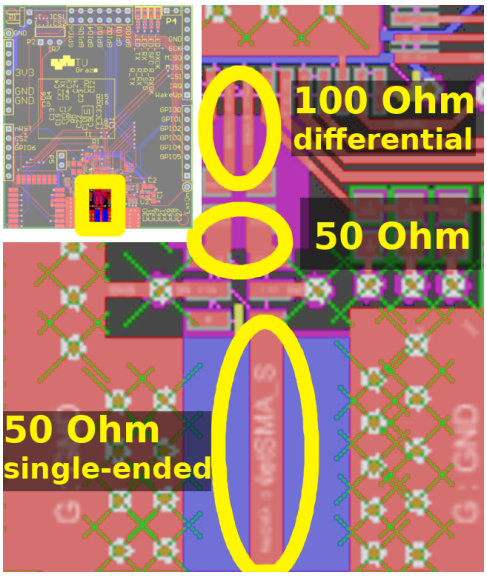


Figure 4.4: RF transmission lines.

Net	Length in mm
100Ω differential	1.18
50Ω differential	0.35
50Ω single-ended	**5.4**

Table 4.4: Lengths of (potential) transmission lines.

highest frequency on these transmission lines is 10GHz, this would lead to a length of $3mm$ to be considered.

$$\frac{\lambda}{10} = \frac{1}{10} \cdot \frac{speed\ of\ light}{frequency} = \frac{1}{10} \cdot \frac{299\,792\,458 \frac{m}{s}}{10 \cdot 10^9 \frac{1}{s}} = 3mm \tag{4.1}$$

For the same reason, the components on the UWB shield have the housing size of **0402**[7]. A bigger housing would require to treat that component like a transmission line. According to Eq. 4.1, the transmission lines can be treated as normal connections as long as they are shorter than $3mm$. This was taken into account when designing the PCB. In the final design, the transmission lines seen in Figure 4.4 had the lengths listed in Table 4.4. Line impedance matching by calculating the correct width was neglected for connections shorter than $3mm$. So only the connection from the Balun to the SMA connector was treated as a transmission line. The model to calculate the correct width of the 50Ω single-ended transmission line is illustrated in Figure 4.5. T is the thickness of the copper on the component side. As mentioned before in Section 4.3.2, there are fabrication methods that guarantee an almost constant thickness. H is the thickness of the dielectric. In this case, it is a $360\mu m$ thick FR-4 (see Table 4.3) having an $\epsilon_r = 4.32$ at $5\ GHz$ [42]. Another rule of thumb says that the space between the shielding GND planes (labeled S in Figure 4.5) and the transmission line should be twice the transmission line's width. There are several

[7] 0402 is a standard package size for SMT components. The dimensions are 1.0mm x 0.5mm.

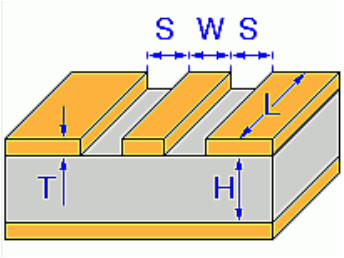


Figure 4.5: Scheme for transmission lines with co-planar wave guide and ground plane.

Tool	Result for W in μm
Altium 17 [57]	**700**
Multi-CB Online Tool [63]	636
PCB Calculator [53]	710
NI AWR TX Line [50]	672
Saturn PCB [48]	740

Table 4.5: Results for W calculated by different tools.

vias added to the GND planes on both sides of the 50Ω transmission line to improve the shielding of the transmission line.

The area around the transmission lines must be free from solder stop and printed text, because these layers are not taken into account when calculating the trace width of the transmission line.

Transmission line calculator tools. The Altium Designer software includes some helpful tools to design transmission lines [57]. By providing the layer stackup shown in Table 4.3 and defining the desired line impedance, Altium automatically calculates the line width. There are several other tools available to calculate the width of a transmission line. Their results are listed in Table 4.5. Even though the results are different, their deviation is still within the manufacturing tolerances. We took the value calculated by Altium Designer for our design.

4.3.7 DWM1000 Footprint

The UWB extension shield design includes a backup solution by soldering a DWM1000 module [27]. The footprint was added to the PCB layout in case the RF transmission line design of the DW1000 is faulty or shows insufficient performance. That way the PCBs would still be of use. Another advantage is that the DWM1000 and the DW1000 can be used in the same design. This can be useful for comparing the performance of the DWM1000's omni-directional UWB antenna with individual designs. Listening on different RF channels at the same time is also possible. Both transceivers share the same SPI bus, but the DWM1000's chip select line is connected to the **CS2** pin on the board. In order to use both transceivers simultaneously, the DW1000's must use the **CS1** connector for its chip select line. This can be achieved through setting jumper **P2** as described in Section A.1.

PHY Setting	Value
Channel	5
Pulse repetition frequency	16 MHz
Preamble symbol repetitions	128
Data rate	6.8 Mbps
Payload size (including MAC header)	12 Bytes

Table 4.6: Default test configuration of the DW1000 radio on power-up.

4.3.8 Crystal

The DW1000 uses a 38.4 MHz quartz crystal to synthesize the frequencies for RF TX, RF RX and all digital blocks. Since ranging and localization applications rely on precise time measurements, the accuracy of the crystal is crucial for their performance. Ambient temperature changes have a strong impact on the clock drift of the quartz crystal. This can be omitted by using a temperature-compensated crystal instead of a simple one. In Section 4.3.1 it was mentioned that we are not using a TCXO in favor of a lower energy consumption. A low frequency tolerance and frequency drift of $\pm 10ppm$ are recommended in the DW1000 datasheet alongside with three crystals by different manufacturers [22]. Because none of these three recommended crystals were available for purchase at design time, we decided to use Epson's TSX-3225 in the design [17]. The TSX-3225 is a 10ppm crystal oscillator with the same specifications as the RSX10 used by the EVB1000 [59].

The DW1000 offers a crystal calibration mechanism to compensate for the initial frequency offset. The procedure to trim the crystal is described in detail in Appendix C.1.9.

In the PCB layout, the crystal must be placed as close as possible to the XTAL1 and XTAL2 pins. To avoid noise interference by the DC-DC converter, the latter was placed on the opposite side of the DW1000, as recommended in the hardware design reference guide [58].

4.4 Hardware Validation and Calibration

To validate the UWB extension shield's correct functionality, and evaluate its RF performance, a test firmware was developed allowing to configure the radio. A description how to use the firmware and execute these tests can be found in Appendix C. Decawave provides a set of recommended production tests for products containing the DW1000 transceiver [21]. The tests are divided into the following three groups:

1. operational tests, where no RF functionality is tested;
2. RF measurements that require a Spectrum Analyzer;
3. RF measurements that require a compatible reference device.

All test setups consist of the NUCLEO-L152RE board with the extension shield stacked onto its Arduino connectors.

The following voltages and electric currents were measured using the Fluke 87-V Digital Multimeter. Transmit power and crystal frequency were calibrated using the Rohde-Schwarz FSQ26 Spectrum Analyzer.

4.4.1 Operational Tests

An important indicator for a valid board design including the DW1000 is the idle-mode current consumption, which must stay below 25mA. The current consumption of the board

Current after 10s	Voltage
[mA]	[mV]
18.88	3270

Table 4.7: Idle current measurement.

in idle mode was within the expected range as shown in Table 4.7. The SPI test executed successfully as well as the GPIO strobe test, the RSTn test, the EXTON test and the WAKEUP test. The results of the maximum current consumption are listed in Table 4.8. The operational tests showed a maximum current consumption that was much higher than expected (shown in Table 4.8). The reason for that is not clear yet. A design flaw in the UWB shield is not suspected, because the same high current consumption can be measured using the DWM1000 too.

4.4.2 Transmitter Calibration

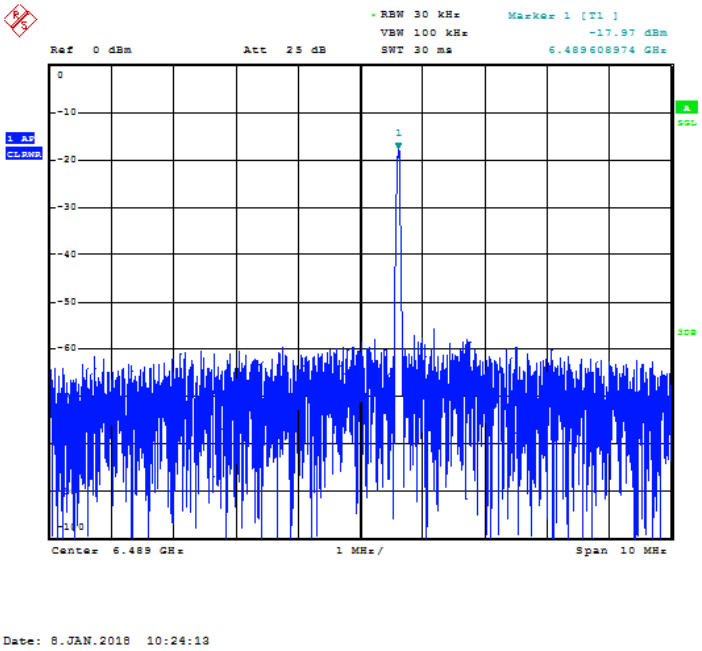
The calibration of the crystal oscillator is essential to maintain accurate TOF measurements. For that reason, it is required to trim the oscillator's frequency by configuring register `0x2B:0E FS_XTAL` of the DW1000.

Crystal calibration. Frequency deviations are more pronounced and simpler to measure at high frequencies, because the deviations are then higher as well. Therefore, the crystal trim was calibrated at the highest configurable channel frequency of $6\,489.6\,MHz$ while monitoring the RF output. The remaining frequency offset was minimized at the crystal trim value `0x16` (Decawaves default configuration is `0x10`). On channel 5, the remaining offset was $0.008\,974\,MHz$ and on channel 4 the spectrum analyzer showed a remaining offset of $0.0MHz$. Captures of the measurements for channel 5 are shown in Figure 4.6a and for channel 4 in Figure 4.6b. In every subsequent test this crystal trim value was used.

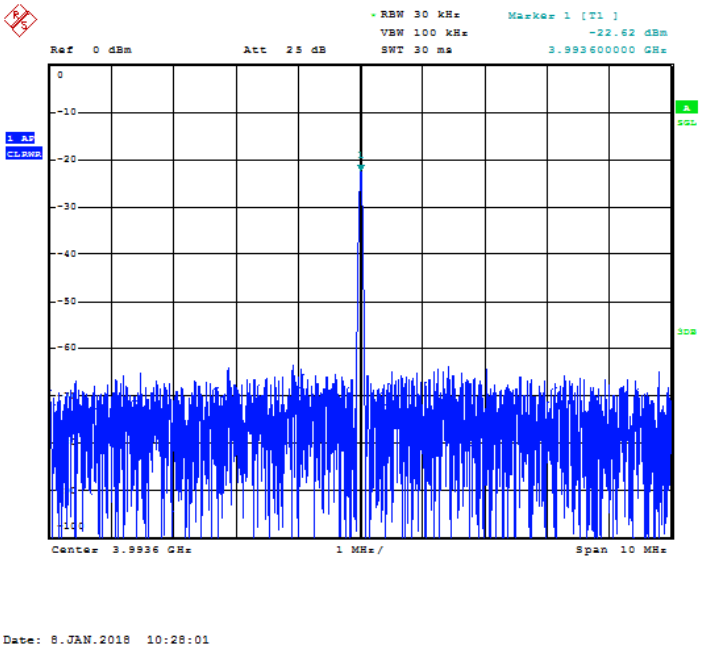
Transmit power calibration. The attenuation of the SMA cable was about $1dB$ across all tested frequencies. We configured the spectrum analyzer offset to take that attenuation into account, so we did not need to subtract it later, as described in Appendix C.1.10. Along with the transmit power, we adjusted the bandwidth as well. The bandwidth can be adjusted by changing the UWB pulse width, using the pulse generator delay (`PGDELAY`) register `0x2A:0B`. By increasing the pulse generator delay, the pulse width (and the transmit

Maximum current consumption	Voltage
[mA]	[mV]
220	3268

Table 4.8: Maximum current measurement.



(a) Result of crystal calibration on channel 5. The marker 1 shows that the peak is at a frequency of 6.489608974 GHz.



(b) Result of crystal calibration on channel 4.

Figure 4.6: Two measurements after calibrating the crystal oscillator. When configuring the DW1000 to transmit on channel 5, the remaining frequency offset is 0.008 974 MHz. On channel 4, the remaining frequency offset is zero.

Channel	f_c	B	f_{PRF}	Code	G_C	G_F	PGDELAY
	[MHz]	[MHz]	[MHz]		dB	dB	
1	3494.4	499.2	16	1	9 (+0)	10.5 (+0)	`0xC9` (+0)
1	3494.4	499.2	64	9	9 (+0)	3.5 (+0)	`0xD0` (+7)
2	3993.6	499.2	16	3	9 (+0)	9.5 (-1)	`0xD4` (+15)
2	3993.6	499.2	64	9	9 (+0)	3.5 (+0)	`0xD4` (+15)
3	4492.8	499.2	16	5	9 (+0)	9.0 (+1.5)	`0xD1` (+12)
3	4492.8	499.2	64	9	6 (+0)	6.0 (-0.5)	`0xD1` (+12)
4	3993.6	1331.2	16	7	**15 (+3)**	**12.0 (+3.5)**	`0x95` (+0)
4	3993.6	1331.2	64	9	6 (+0)	11.5 (-2.5)	`0x95` (+0)
5	6489.6	499.2	16	3	12 (+0)	3.0 (+1)	`0xD2` **(+18)**
5	6489.6	499.2	64	9	6 (+0)	1.5 (-1)	`0xD2` **(+18)**
7	6489.6	1081.6	16	7	6 (+0)	13.0 (+4)	`0x93` (+0)
7	6489.6	1081.6	64	17	0 (+0)	12.0 (+3.5)	`0x93` (+0)

Table 4.9: Results for the transmit power and bandwidth calibration. For each possible carrier frequency f_C, bandwidth B and pulse repetition frequency f_{PRF}, two registers were configured. The register `TXPOWER` configures a coarse gain G_C and a fine gain G_F, and the register `PGDELAY` configures the pulse generator delay. The value in brackets describes the deviation from Decawave's reference setting, e.g. +3 means reference value had to be increased by 3, −1 means the reference value had to be decreased 1. +0 indicates that the reference value was left unchanged.

power) increases and the bandwidth decreases. If this value is decreased, the bandwidth increases (and the transmit power decreases). The measurement results are listed in Table 4.9. The numbers in brackets indicate the difference to the reference values from the DW1000 user manual [26]. Figure **??** and Figure B.4 in the Appendix show the spectrum for all channels and pulse repetition frequencies (PRF). Outside the band, the transmit power has to be $10dB$ below the maximum limit of $-41.3\,dBm/MHz$.

Antenna delay calibration. After the crystal oscillator and the transmit power/bandwidth, also the antenna delay has to be calibrated. The DW1000 supports a special transmit mode to support TOF measurements. In this mode, the timestamps of received and transmitted packets are saved in a register. The timestamp is internally corrected by taking the antenna delay into account, when reading the timestamp from the register. The antenna delay is the time duration between a frame being processed digitally and the corresponding RF signal passing the antenna. For reception and transmission the antenna delay is different. The ranging accuracy can be increased from 30cm to 4.5cm by calibrating the antenna delay [20]. A description of the test procedure can be found in Section C.1.11. Figure 4.7 shows the histogram of 10000 ranging measurements in a cable-based setup. The test boards were connected using a $1\,m$ coaxial cable[8]. The propagation speed of electromagnetic signals in the coaxial cable is, according to its datasheet [43], 69.4% of the speed of light in vacuum. Taking the average measured distance $\mu = 155.29\,m$, this gives

[8]Harbour Industries M17/152-00001 MIL-DTL-17, tested for a maximum frequency of $12.4\,GHz$. This cable's dielectric material is RG-316 solid Teflon (ST).

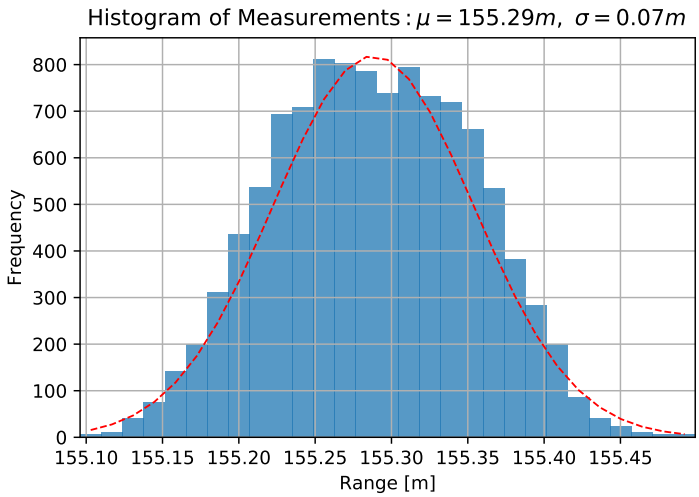


Figure 4.7: Histogram of TWR measurements.

a combined antenna delay of $741.58\,ns$ according to Eq. 4.2.

$$Delay_{TX,RX} = \frac{155.29\,m - 1.0\,m}{0.694 \cdot 299\,792\,458 \frac{m}{s}} = 741.58\,ns \tag{4.2}$$

The antenna delay for transmitted frames is then given by Eq. 4.3 and by 4.4 for received frames (refer to Appendix C.1.11 for details). Processing received frames takes slightly longer than transmitting frames. According to [20] the combined delay is apportioned 56% and 44% for the receiver delay and the transmitter delay respectively.

$$Delay_{RX} = 741.58\,ns \cdot 0.56 = 415.28\,ns \tag{4.3}$$

$$Delay_{TX} = 741.58\,ns \cdot 0.44 = 326.29\,ns \tag{4.4}$$

After calibrating the antenna delay, the transmit power and bandwidth, and the crystal oscillator of each device, centimeter-accurate distance estimates can be derived using the self-designed UWB shield.

Chapter 5

Software Design

This chapter explains the software design of NetLoc. First, the extension of the routing protocol and the Contiki application running on all nodes in the network (built using the platform described in Chapter 4) are explained in Section 5.1. We further explain in this section how distance information of the network is collected. Next, the localization application is described in Section 5.2, whereas Section 5.3 gives details on the NetLoc system performance validation. Limitations of the software design are discussed in Section 5.4.

5.1 Contiki Application

This section describes the Contiki application running on the UWB platform. The Contiki application's building blocks as well as its communication flow are shown in Figure 5.1. The position of these blocks within the operating system's architecture is displayed in Figure 2.2. Note that every node in the network is programmed with the same application. The user application that this sections refers to, is named `rpl-uwb` in Figure 2.2.

5.1.1 User Application

The Contiki user application implements the following three processes:

`role_assignment_process.` This is the only user process that is started by Contiki automatically. Its task is to check the node's hardware address and based on that address either start the `udp_server_process` or the `udp_client_process`. Only the sink node starts the server process, all other nodes start the client process. After either the `udp_server_process` or the `udp_client_process` is started, the `role_assignment_process` exits.

`udp_server_process.` This process is supposed to run only on the network's sink node and consists of the following two execution phases. First, during a setup phase, the process configures the sink node's IP address, then initializes a new RPL instance and declares itself as the RPL root. The next step is to open a UDP connection and to listen for incoming packets. This UDP connection is used for application layer communication as well. After this setup phase, the process enters the data collection phase. During this data collection

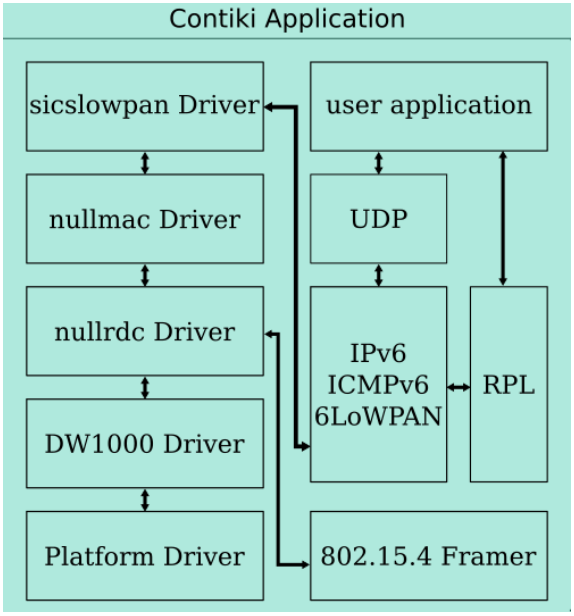


Figure 5.1: Building blocks of NetLoc's Contiki application

phase the sink node does not contact the clients and waits in an endless loop for incoming UDP packets.

`udp_client_process.` This process is executed on all nodes in the network except the sink node. The process has a setup phase and a continuous execution phase. In the setup phase, the IP address is configured and a UDP connection is opened. The UDP connection listens for incoming packets and can be used to send data to other nodes. Then the process enters the second phase of continuous execution. In this example application the client nodes remain idle.

5.1.2 Embedding Distance Information into RPL Efficiently

The following modifications to Contiki's implementation of the RPL protocol allow to measure the TOF between nodes using DIO packet transmissions and to collect the network's distance information at the RPL root.

`dio_output()` This function builds a DIO packet to be sent as a unicast packet to one of the node's neighbors. This `ICMPv6` packet is, unlike other IPv6 packets, not added to the `TCPIP` packet buffer and scheduled for transmission, but sent instantly. There is no task switch and no other transmission before the radio driver at the bottom of the network stack is invoked. Because of that, it is safe to change the state of the DW1000 driver just before calling the `ICMPv6's` send function. To change the radio driver's state, the value of the radio parameter `DW1000_UCRNG` is changed, informing the driver to set the **ranging bit** in the physical header of the next transmitted frame (refer to Section 2.2.2 for details about the frame structure). Implementation details of this radio parameter are given in Section 5.1.3. The receiving node of this DIO packet will see an active ranging bit,

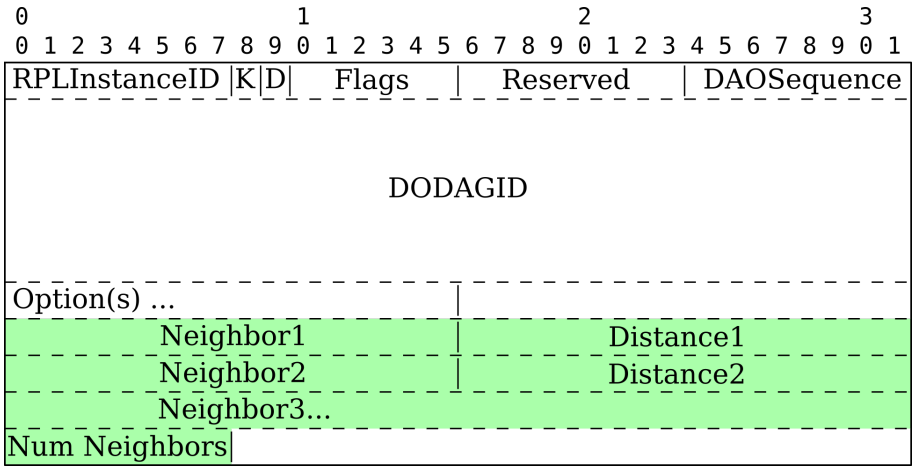


Figure 5.2: Format of the DAO Base Object [86, Section 6.4.1.]. Appended new fields are colored green.

remember the reception timestamp, and report that the received frame is a ranging frame. The transceiver remembers the destination address of the DIO to identify the expected ACK frame. If the ACK is lost or a different packet is received instead, the link statistics are updated accordingly and the distance measurement failed.

`dao_output()` The modifications to this function comprise changes to the DAO packet. After this function assembled a unicast DAO packet, distance information from the node's neighbor table is appended to it. The maximum additional payload is therefore limited by the maximum number of neighboring nodes, which is defined in Contiki during compile time. New fields of the DAO packet are colored green in Figure 5.2. The field `num neighbors` holds the number of appended $(neighbor, distance)$ tuples. A loop iterates through all entries in the neighbor table and checks if distance information is present. If so, four bytes are appended to the DAO packet containing the 16 bit neighbor address and the distance in centimeter as a 16 bit decimal number. For every $(neighbor, distance)$ pair added to the buffer, a counter is incremented. After the loop, this one-byte counter holding the number of neighbors is appended to the packet buffer as well. Then this `ICMPv6` packet is sent without further modifications.

Since the DAO format defines optional fields (DODAGID and Option(s)), the length of a DAO packet can vary. Hence, the bit alignment of the new fields as displayed in Figure 5.2 is just one possible layout.

`dao_input_storing()` DAO packets are sent by all nodes in the network and received by the sink node. In this method, the appended $(neighbor, distance)$ pairs are printed by the sink node to the serial interface, in a way that allows further processing. The distance information of each received DAO packet is represented by a single line in the output. To differ distance related information from other serial output, the line starts with the identifying character string `RPLUWB`. All following fields are separated from each other by commas. The next field is the 16 bit node address of the sender printed as a hexadecimal string. The third field is the DAO sequence number, printed in decimal format. The

rest of the line consists of $(neighbor, distance)$ pairs, where the neighbors address is the hexadecimal representation of its node address and the distance is a decimal number. The number of neighbors is contained in the last byte of the buffer, as displayed in Figure 5.2.

The size of the additional payload in the DAO packet does not grow with the network's size, but only with the number of neighbors, therefore it is proportional to the network density. Contiki's default value for the maximum size of the neighbor table is 8. This number can be changed depending on the platform's resources and the application requirements, but must be chosen during compile time. Typically, the maximum number of neighbors is less than 10. A node with eight neighbors would append 33 bytes additional payload to a DAO packet. To efficiently transmit the neighbor table, the DAO packet and the additional payload should fit into a 127 byte IEEE 802.15.4 frame. Without using IPv6 header compression, one feature of 6LoWPAN, the packet size sums up to 73 byte (9 byte IEEE 802.15.4 header + 40 byte IPv6 header + 4 byte ICMPv6 header + 20 byte DAO header), assuming a 4 byte addressing mode, no IEEE 802.15.4 security header and no optional fields in the DAO header. This would leave enough space to send 13 $(neighbor, distnace)$ tuples in the same packet. Compressing the 40 byte IPv6 header to 3 byte would increase the maximum number of tuples in the same packet to 22. DAO options can take up to 42 byte additional space[1], reducing the number of tuples to 12 if header compression is used, and to 2 if no header compression is used. Therefore, using header compression and setting the maximum number of neighbors to 12 or less is highly recommended. In NetLoc, the maximum number of neighbors is set to 8.

5.1.3 DW1000 Driver

The ported DW1000 Contiki driver was originally implemented for the EVB1000 platform [15, 59]. The driver's emplacement within Contiki's software architecture is displayed in Figure 2.2 in the bottom-right corner (dw1000/dw1000-ranging). Because of the driver implementation's well-structured design, only few changes were necessary to port the DW1000 driver to the NetLoc platform. Except for some hardware dependent timing constants and EVB1000 board specific configurations, platform dependent code was separated from the radio driver. The DW1000 driver consists of three parts.

1. Decawave's own DW1000 application programming interface (API) [25]. A collection of constants as well as functions to configure and control the DW1000 UWB radio transceiver.

2. A DW1000 ranging process, a state machine, and helper functions of all ranging related code are encapsulated in two files. TOF measurements for SS-TWR and DS-TWR (described in Section 2.2.4) and distance calculations are implemented in the ranging process.

3. The DW1000 radio process handles all non-ranging related packet reception and transmission. Contiki's `NETSTACK_RADIO` interface functions are implemented as well as callback functions for the DW1000 API.

[1] 20 byte for the RPL Target option and 22 byte for the Transit Information option.

DW1000 Interrupt Service Routine. The DW1000 is configured to trigger a hardware interrupt for specific events like the reception of a frame, receiver timeouts, or receiver errors. Table A.2 shows the `IRQ` pin and its connection to the MCU. The same interrupt service routine (ISR) is called for all interrupt events. After such an event occured, the DW1000's status register is read to check, which callback function of the driver must be called.

DW1000 Ranging Implementation. A Contiki user process can utilize the DW1000's ranging capabilities by calling the `dw1000_range_with()` function. This function will initiate either a single-sided or double-sided TWR with a target node, identified by its 16-bit address. In contrast to data communication packets, the frames sent by the ranging methods will set the ranging bit active in the PHR (see PHR description in Section 2.2.2). The aforementioned ISR will notice if a received frame has this bit set and calls a different function for ranging frames than for data frames. After successfully completing the TWR process, the distance is calculated from the TOF and the initiating user process is notified. The user process gets the distance in meter to the target node.

DW1000 Radio Process. The Contiki `NETSTACK_RADIO` interface is implemented to handle data exchanges. While packet transmissions happen instantly, packet reception is handled event driven by the radio process. When a good frame is received by the radio, it is not immediately copied to the Contiki packet buffer. The ISR executes a callback function that will set a `frame_pending` flag in the DW1000 driver and then this callback function polls the DW1000 radio process. The next time the radio process is scheduled, it will then download the data from the radio's RX buffer to the Contiki packet buffer and hand over control to the upper layer (`NETSTACK_RDC`) which will process the received data.

DW1000 Driver Modifications

The radio driver was extended (i) to support SS-TWR through unicast DIO transmissions, (ii) to collect link metrics so RPL can optimize the MRHOF (explained in Section 2.1.4), (iii) to load calibrated radio configurations and transmission delays, as well as (iv) to support the driver's `set_value()` and `get_value()` interface functions.

Unicast DIO Ranging. The following driver modifications were necessary to support SS-TWR for IEEE 802.15.4 data packets.

- The custom radio parameter `DW1000_UCRNG` was added to enable and disable the driver's state through the generic `NETSTACK_RADIO` interface. Setting this parameter by calling `NETSTACK_RADIO.set_value(DW1000_UCRNG, 1)` causes the driver to enable the ranging bit of the PHR for the next packet transmission. This state is kept until the expected response to the ranging-DIO (the ranging-ACK) is received, or a receiver error was detected.

- If a ranging-DIO packet was received, the function `dw1000_receive_ranging_uc()` is called to process it and respond accordingly. First, the DW1000 reception timestamp is read. The delayed transmission of the IEEE 802.15.4 ACK frame is scheduled and

both timestamps are appended to the packet. After responding with a ranging-ACK, the radio process is polled to continue processing the received DIO.

- If a ranging-ACK packet was received from the same node to whom a ranging-DIO packet was sent, its timestamps are read from the receive buffer. Then, the node's own transmit timestamp and receive timestamp are read from the DW1000. Knowing all four timestamps, the TOF can be calculated using Eq. 2.3. To deal with potential measurement errors due to clock drift (discussed in Section 2.2.4) the TOF result is corrected by the offset derived from a DW1000 register called *carrier recovery integrator register* [26, Section 7.2.40.11]. The node's neighbor table entry is updated and the radio process informed to continue processing the IEEE 802.15.4 ACK.

Link Metrics. To collect link statistics, certain attributes of every received packet are saved by the driver to be processed by higher layers. RPL needs to know each link's expected number of transmissions (ETX) to optimize the MRHOF. The ETX is derived form the received signal strenght indicator (RSSI) and the timestamps of successful transmissions. Normally, the ETX is a function of the packet reception ratios, but in Contiki the RSSI is used to estimate this ratio. The timestamp of the received packet is updated by reading the current `rtimer`[2] value is set as the `PACKETBUF_ATTR_TIMESTAMP`. The DW1000 user manual suggests Eq. 5.1 to estimate the RSS [26, Chapter 4.7.2]. In Eq. 5.1, A depends on the pulse repetition frequency (PRF) of the radio, C and N are both register values that are related to the received signal strength. This formula is used in the driver to calculate the approximated RSSI, which is then set as the `PACKETBUF_ATTR_RSSI`.

$$RSS[dBm] = 10 \cdot log_{10}\left(\frac{C \cdot 2^{17}}{N^2}\right) - A \tag{5.1}$$

Load Radio Calibration. During initialization of the DW1000 and after changing its configuration, the calibrated values for the crystal-trim, the transmit power and the pulse generator delay are loaded (described in Section 4.4.2 in detail). The crystal-trim is a constant and does not depend on other radio configurations. However, the transmit power and the pulse generator delay depend on the carrier frequency, the channel bandwidth and the PRF. Calling the function `dw1000_load_transmitter_calibration()` will load the calibrated values for a specific channel and PRF.

New Transmission Delays. TWR does not work with the driver's original configuration, but results in timeout errors. The reasons are the slower SPI bus speed[3] and the longer processing times of the MCU[4] used in the NetLoc platform. In the double-sided TWR method, three packets are sent to measure two round-trip times (see Figure 2.9). The second and third packet must be sent at a future timestamp using the DW1000's feature of delayed transmission. The problem was that the original delay of about $460\,\mu s$ was too short. To adapt the DW1000 driver to the NetLoc platform, the transmission delay had to be changed. By the time the MCU gave the command for starting the delayed

[2] Contiki's own real-time clock timer, running at a frequency of 32.768kHz.
[3] The SPI clock is $8\,MHz$, the EVB1000's SPI clock is $25\,MHz$.
[4] The STM32L152RE is clocked with $32\,MHz$, the EVB1000's MCU is clocked with $50\,MHz$.

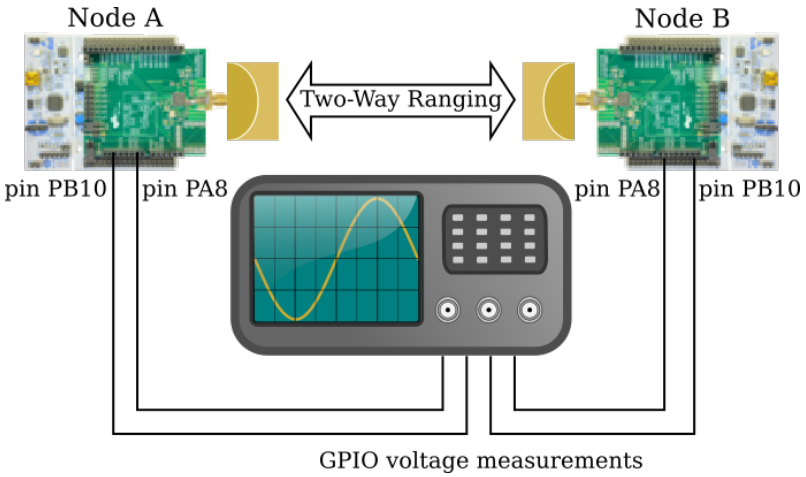


Figure 5.3: Test setup to measure processing times and communication times during DS-TWR using an oscilloscope.

transmission, the DW1000's clock had already passed the specified timestamp. Choosing a timestamp that is too far in the future would increase the duration of the TWR and therefore the systematic error introduced by clock frequency offsets (see Section 2.2.4). To find a reasonable trade-off for the transmission delay and to estimate the duration of the double-sided TWR process, the processing times and communication times were measured using an oscilloscope. The same transmission delays are used for the first and second message for single-sided TWR as well. Figure 5.3 shows a sketch of the measurement setup. The test application initiates DS-TWR measurements in an endless loop. The general purpose input/output (GPIO) pins `PA8` and `PB10` of the MCU were configured to change their value to HIGH while they were processing specific data. At the same time, the test program tried to find the minimum delay between two consecutive ranging measurements. Figure 5.4 shows one example measurement using a bit rate of 6.8Mbps and a transmission delay set to $2\,000\,uus$[5]. It shows a processing time on node A of $812\,\mu s$ between calling the `dw1000_range_with()` function and the start of transmission, followed by a $2.472\,ms$ long delay until node A receives the response from node B. Then it takes node A $944\,\mu s$ to process the response, create the third and final packet, and send it to node B. Node B starts idle, waiting for an initiator packet. When node B receives this packet, it responds to it after a processing time of $796\,\mu s$ and then receives the final packet after a delay of $3.316\,ms$. The whole DS-TWR process, from calling the `dw1000_range_with()` function to getting the resulting distance in meter, takes $6.472\,ms$. To shorten this duration, one could switch to a faster MCU, such as the $50\,MHz$ clocked STM32F105 on the EVB1000 board. The length of the transmitted frames and the bit rate have an impact as well. In this case, the preamble length was 128 symbols and the payload length 12 bytes.

A long-term test of this DS-TWR implementation using the new transmission delays showed that out of 562 293 ranging attempts 98.98% were successful and only 9 resulted in a timeout of a response packet. 5 703 (1.01%) ranging attempts failed because the

[5] uus stands for UWB-microsecond, the DW1000's equivalent to a microsecond. Its definition is $1uus = 512/499.2\,MHz = 1.026\,\mu s$

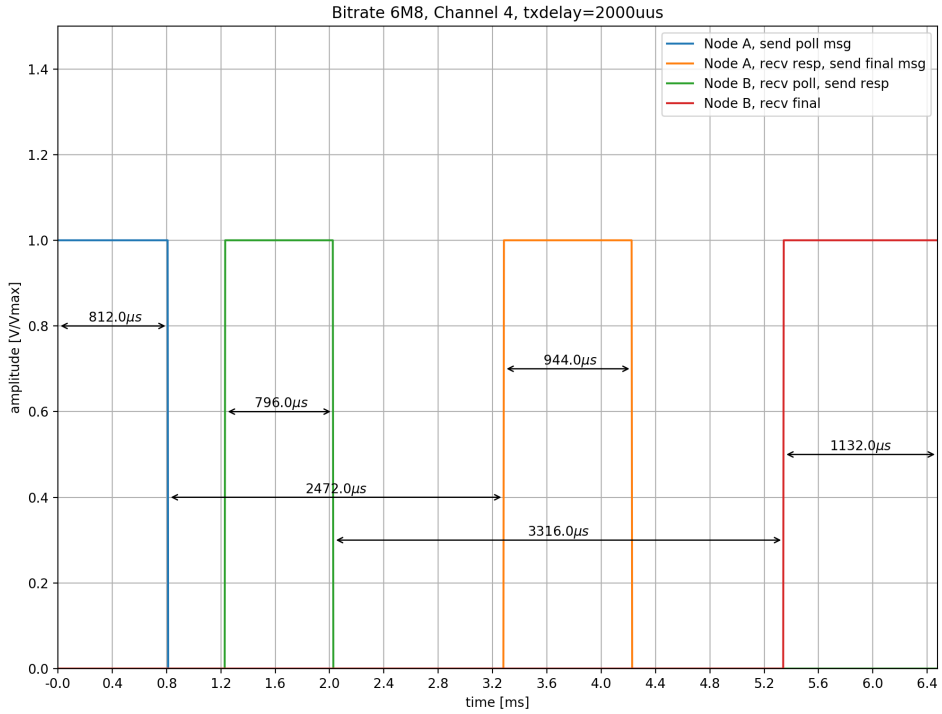


Figure 5.4: Measured time intervals during a DS-TWR measurement between two nodes.

initiating, first packet was lost. 98.76% out of all ranging attempts were successful and started after a delay of 15.625 ms after the previous attempt.

Set/Get Radio Values. Contiki's `radio.h` defines a list of generic radio parameter to control its configuration and check its state. User applications and higher layer protocols can call the `NETSTACK_RADIO.set_value()` and `NETSTACK_RADIO.get_value()` functions to get information about the radio's state, change its channel or control hardware features like automatic ACK.

5.1.4 Platform Driver

The platform driver is the assembly of all hardware components that are controlled or configured by software. A similar platform driver, based on the same MCU board but combined with a different radio, was implemented for Contiki by STMicroelectronics [77]. Porting this implementation of Contiki's `stm32nucleo-spirit1` driver was done in two steps: (i) remove all dependencies to the `spirit1` radio and (ii) add support for the DW1000 radio.

Removing `spirit1` radio dependencies involved changing the sources as well as the Makefiles. Dependencies and necessary modules for compiling the new `stm32nucleo-dw1000` platform are defined in `Makefile.stm32nucleo-dw1000`, which includes the MCU Makefile `Makefile.stm32l152` and all HAL and CMSIS source files.

DW1000 SPI. The SPI pins that connect the DW1000 radio and the MCU are defined and initialized in `dw1000-arch.c`. Decawaves API for the DW1000 leaves the hardware dependent `deca_sleep()` function and the SPI function defined, but not implemented. The functions `writetospi()`, `readfromspi()`, `decamutexon()`, `decamutexoff()` and `deca_sleep()` are called by API functions, but must be implemented in the platform driver.

DW1000 Interrupt. Three interrupt-related functions must be implemented to control the DW1000. The `decamutexon()` and `decamutexoff()` functions enable and disable the external interrupt on pin `PC7` on the MCU, which is connected to the `IRQ` pin of the DW1000. If the interrupt is enabled, the MCU will automatically call the interrupt handler `EXTI9_5_IRQHandler`, which will clear the interrupt and call the DW1000 API's interrupt service routine.

Parameter	Value
Port	declared on the command line
Baudrate	921600
Hardware flow control	off
Software flow control	off
RTS/CTS	off
Byte size	8
Parity	no
Timeout	$1s$
Stop bits	1

Table 5.1: Serial device settings.

5.2 Localization Application

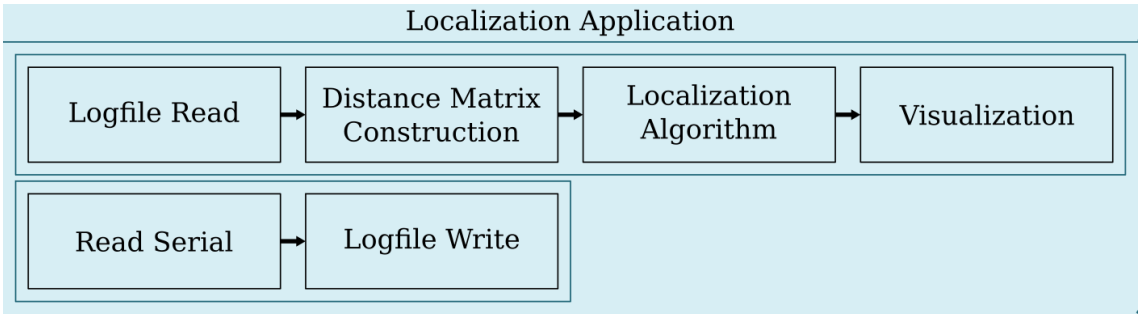


Figure 5.5: Building blocks of NetLoc's localization application.

Figure 5.5 shows the blocks of NetLoc's architecture that belong to the localization application. The localization application was implemented in the Python programming language and runs on the host PC. Two separate programs, one for each process, must be started to execute the application. The first process opens a figure to continuously show updates of the sensor network on a map. The second process reads from a serial connection and writes the data to a log file.

5.2.1 Read Serial

This block opens a serial device at a specific port, filters and passes the output on to the block *Write Logfile*. The filter mechanism ensures that only Unicode characters will be written to the log file to prevent programs from crashing by reading corrupted data. Every non-Unicode character is replaced by a '.' (period) character.

Table 5.1 lists the settings of the serial device from which this block reads its input. The corresponding UART settings configured by the *Platform Driver* must match the ones from Table 5.1.

5.2.2 Logfile Write

The task of this block is to create a log file once the connected sink node boots Contiki and keep it updated until the node is disconnected or reset. The block gets its input from the *Read Serial* block in the form of lines. Until the Contiki boot message is identified, all lines are skipped. The following text sample shows the first lines of a boot log:

```
Contiki on node a5 23 00 10 e7 5f 20 10
node address: a5.23.00.10.e7.5f.20.10
ROLE server
Success: created a new RPL dag
```

Once the boot message is identified, a new log file is created with the node's address as its filename. The example text would be written to a file with the name `a5230010e75f2010.log`. Optionally, a timestamp can be added to the beginning of each line.

5.2.3 Logfile Read

This block parses the log file for specific lines that contain distance estimations. The log file contains a character string, which is read line by line. A line starting with the identifying string `RPLUWB` will be parsed and processed, all other lines are ignored.

The syntax of these distance updates is as follows:

1. `RPLUWB`: the identifying string
2. A short form of the senders address, which is a 16 bit number in hexadecimal format.
3. A sequence number in decimal format.
4. One or more pairs consisting of a neighbors address and the distance to that neighbor node. The address is a 16 bit number in hexadecimal format, the distance is a decimal number and the unit is centimeter. Both numbers are separated by a comma.

All listed entries are separated by commas. The sequence number is not used by the localization application, it is important to the validation application described in Section 5.3. The sender's address N_i is combined with every neighbor's address N_j and this neighbor's distance estimation d_{ij} in a tuple (N_i, N_j, d_{ij}). A list of all tuples is then passed on to the block *Distance Matrix Construction*.

Example

An example log file contains the following lines:

```
RPL: Received a DAO with sequence number 42 from fe80::000a:a116:151:3204
RPL: DAO lifetime: 30, prefix length: 128 prefix: fd00::000a:a116:151:3204
RPL: Adding DAO route
RPLUWB, 000a, 42, 000b, 224, 000c, 300, 000d, 400, 000e, 283
RPL: Received a DIO from fe80::8c03:a156:151:3204
```

Except for the second to last line, all lines are ignored. The line that starts with `RPLUWB` will be processed. The node that sent these distance estimations has the address `000a` and the sequence number is 42. The following fields are pairs of neighbor-address and distances. Neighbor `000b` is at a distance of 2.24 meter, neighbor `000c` at 3 meter, neighbor `000d` at 4 meter and neighbor `000e` at 2.83 meter. These addresses and distance estimations are each packed into one tuple together with the sender's address. The list of these tuples $\{(A, B, 2.24), (A, C, 3.00), (A, D, 4.00), (A, E, 2.83)\}$ is then given to the next block which updates the distance matrix.

5.2.4 Distance Matrix Construction

The task of this block is to create and maintain the network's distance matrix $\mathbf{D}$. Its input from the previous block (*Logfile Read*) are lists of N edges:

$$\mathbf{E} = \{e_0, e_1, ..., e_N\}.$$

Each list item e_n is a tuple of two node addresses N_i and N_j, and a distance estimation between those two nodes d_{ij}:

$$e_n = (N_i, N_j, d_{ij}).$$

For every tuple in this list, two elements in the distance matrix are updated[6]. The distance d_{ij} between the nodes N_i and N_j is updated in column i and row j, and row i and column j of matrix $\mathbf{D}$. If an entry in the matrix is zero, this does not necessarily mean that the distance between both nodes is zero, but that there is no edge between these two nodes. Every time an address has not yet been added to the distance matrix, the matrix is extended by one row and one column and its elements initialized to zero.

Example

The block gets a first list of edges $\{(A, B, 2.24), (A, C, 3.00), (A, D, 4.00), (A, E, 2.83)\}$. The resulting matrix is $\mathbf{D}_1$ is:

$\mathbf{D}_1 =$

	A	B	C	D	E
A	0.00	2.24	3.16	4.00	2.83
B	2.24	0.00	0.00	0.00	0.00
C	3.16	0.00	0.00	0.00	0.00
D	4.00	0.00	0.00	0.00	0.00
E	2.83	0.00	0.00	0.00	0.00

A second input contains the list $\{(D, A, 4.00), (D, C, 4.24), (D, E, 6.32), (D, B, 2.24)\}$, which is used to update $\mathbf{D}_1$. The matrix $\mathbf{D}_2$ results to

$\mathbf{D}_2 =$

	A	B	C	D	E
A	0.00	2.24	3.16	4.00	2.83
B	2.24	0.00	0.00	2.24	0.00
C	3.16	0.00	0.00	4.24	0.00
D	4.00	2.24	4.24	0.00	6.32
E	2.83	0.00	0.00	6.32	0.00

[6] Because the matrix is quadratic and symmetric.

A third input $\{(E, B, 4.12), (E, C, 3.00)\}$ misses some edges:

$\mathbf{D}_3 =$

	A	B	C	D	E
A	0.00	2.24	3.16	4.00	2.83
B	2.24	0.00	0.00	2.24	4.12
C	3.16	0.00	0.00	4.24	3.00
D	4.00	2.24	4.24	0.00	6.32
E	2.83	4.12	3.00	6.32	0.00

And a fourth input $\{(C, B, 2.24), (\mathbf{C}, \mathbf{E}, \mathbf{3.16})\}$ overwrites existing elements:

$\mathbf{D}_4 =$

	A	B	C	D	E
A	0.00	2.24	3.16	4.00	2.83
B	2.24	0.00	2.24	2.24	4.12
C	3.16	2.24	0.00	4.24	**3.16**
D	4.00	2.24	4.24	0.00	6.32
E	2.83	4.12	**3.16**	6.32	0.00

The reception of these lists lead to the distance matrix also shown in Table 5.2. After every update of the distance matrix, the latter is passed on to the next block (*Localization Algorithm*).

5.2.5 Localization Algorithm

The blocks *Localization Algorithm*, *Distance Matrix Construction* and *Logfile Read*, are executed by the same thread. The latter communicates with the *Visualization* thread via a thread-safe `Queue` object. The localization algorithm is responsible for estimating the node positions based on the distance matrix of the network graph. An optimization problem is defined by the error function in Eq. 5.2, where $err_n(\mathbf{x})$ is the error for node n at a point $\mathbf{x}$ on the map. The squared differences of measured distances $d_{n,k}$ and distances calculated from the node position estimates $d(\mathbf{x}, \mathbf{x_k})$ are summed up for all known nodes $\{\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_K}\}$. The error function is minimized by the **conjugate gradient** [65] method, implemented in `scipy.optimize.minimize`.

$$err_n(\mathbf{x}) = \sum_{k=0}^{K} [d(\mathbf{x}, \mathbf{x_k}) - d_{n,k}]^2 \tag{5.2}$$

The distance matrix $\mathbf{D}$ and the sink node's address are enough information to calculate coordinates of the nodes relative to the sink. To get coordinates with respect to a predefined map, the following inputs are required:

1. The distance matrix of the network.
2. The sink node's address and its (x, y) coordinates on the map[7].
3. The address of a second node and its angle in respect to the sink node. The distance between these two nodes can vary, but the angle must stay constant. To simplify the network setup, this angle was chosen to be zero.

[7] The scope of this thesis covers two-dimensional localization. Three-dimensional localization is discussed in Section 7.1.2.

4. The address of a third node whose movement is restricted to the half-plane that is defined by the line between the sink and the second node.

Next, a general description of the localization algorithm is given and its function is demonstrated using the example distance matrix shown in Table 5.2.

Algorithm

1. Calculate the position of the second node depending on the position of the sink node and its distance to the sink node.

2. Calculate the position of the third node by minimizing its error function. Check if the position is on the correct half-plane. If not, mirror it at the line formed by the sink node and the second node.

3. Create two lists: `known` and `unknown`. Add the first three nodes to list `known` and add all other nodes to list `unknown`.

4. Search the list `unknown` for the node that has the most neighbors in the list `known`.

5. Calculate its coordinates by minimizing its error function. Take the previous position of the current node as initial guess for the `minimize` function. If no previous position is available, chose random coordinates (x_R, y_R) as initial guess. Both, x_R and y_R are uniformly distributed random numbers, where $x_{min} \leq x_R \leq x_{max}$ and $y_{min} \leq y_R \leq y_{max}$. Of all nodes in the list `known`, x_{min} is the minimum x-coordinate, x_{max} is the maximum x-coordinate, y_{min} is the minimum y-coordinate, and y_{max} is the maximum y-coordinate.

6. Remove the located node from the list `unknown` and add it to the list `known`.

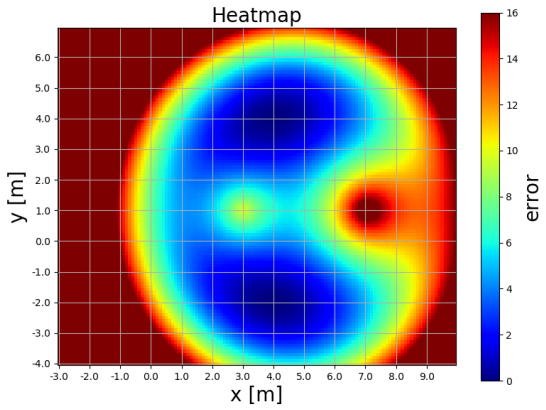
7. If list `unknown` is not empty, continue at step 4.

The algorithm's output is the list of the network addresses and their positions on the map (list `known`).

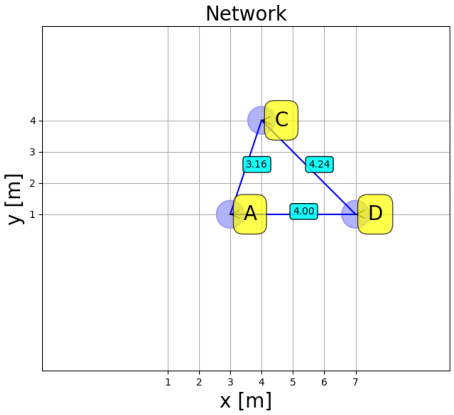
Example

To demonstrate the functionality of the algorithm step-by-step, the example network in Figure 5.7 will be located. The distance matrix of this example is shown in Table 5.2. Node A shall be the sink and fixed at map coordinates $\mathbf{x}_A = (3, 1)$. The second node shall be node D and the angle $\angle\vec{AD} = 0$. The third node shall be node C and it shall is the left of $\vec{AD}$, in other words: $\angle ADC < \pi$.

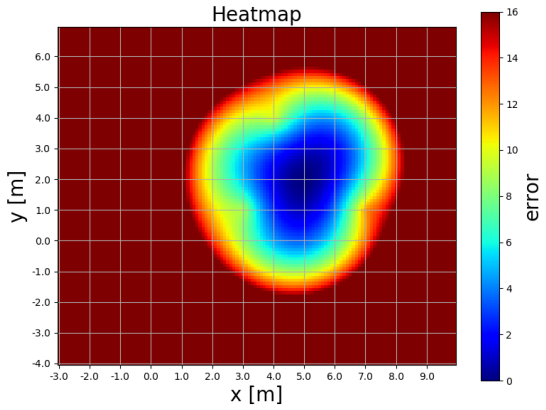
Step 1: Calculate the coordinates of the second node: $\mathbf{x}_D = (x_D, y_D)$. Because of the condition $\angle\vec{AD} = 0$, D has the same y-coordinate as A. Table 5.2 shows that node D lies at a distance of 4 meter from A, hence $\mathbf{x}_D = (x_A + 4, y_A) = (7, 1)$.



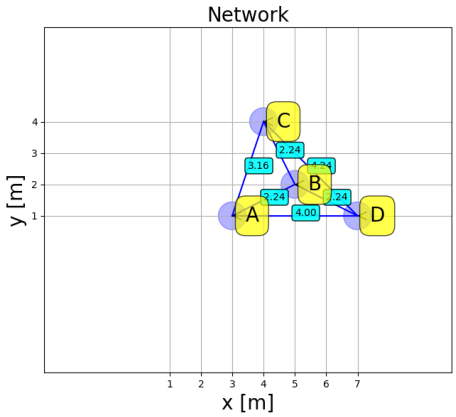
(a) Heat map plot of Eq. 5.3, showing two possible positions for node C at $(4, -2)$ and $(4, 4)$.



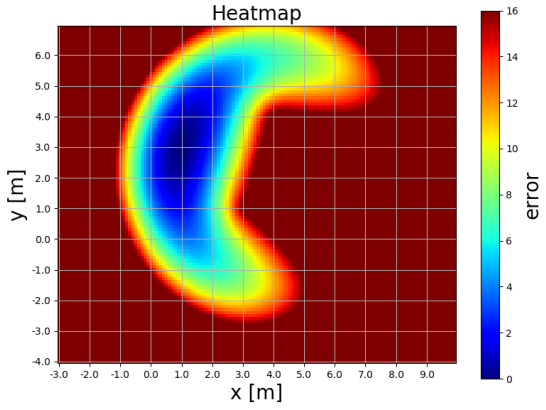
(b) Node C placed at the upper minimum of err_C.



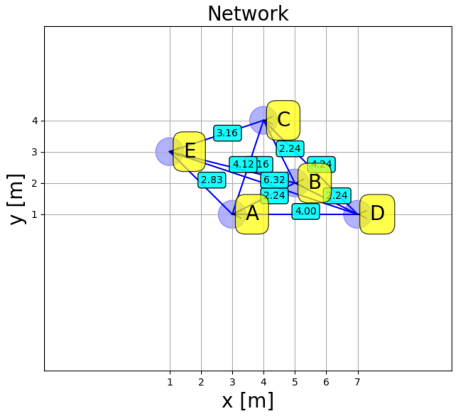
(c) err_B has only one global minimum.



(d) Node B placed on the global minimum of err_B.



(e) err_E has only one global minimum.



(f) Node E placed at the global minimum of err_E.

Figure 5.6: Stepwise localization of nodes in a network.

$d\ [m]$	A	B	C	D	E
A	0.00	2.24	3.16	4.00	2.83
B	2.24	0.00	2.24	2.24	4.12
C	3.16	2.24	0.00	4.24	3.16
D	4.00	2.24	4.24	0.00	6.32
E	2.83	4.12	3.16	6.32	0.00

Table 5.2: Distance matrix **D** of the network graph shown in Figure 5.7

Step 2: Calculate the coordinates $\mathbf{x}_C = (x_C, y_C)$ of the third node C by minimizing Eq. 5.3. Figure 5.6a displays a heat map plot of Eq. 5.3 showing that this function has two extrema.

$$err_C(\mathbf{x}) = [d(\mathbf{x}, \mathbf{x}_A) - 3.16]^2 + [d(\mathbf{x}, \mathbf{x}_D) - 4.24]^2 \tag{5.3}$$

Because of the constraint $\angle ADC < \pi$, there is only one possible solution. If the wrong position is found, the solution can simply be mirrored on the line $\vec{AD}$. Choosing the initial point of the `minimize` method on the correct half-plane avoids this extra step, because it will converge to the minimum on the same half-plane. The `minimize` method calculated the coordinates $\mathbf{x}_C = (4, 4)$. In Figure 5.6b, node C is added at this position.

Step 3: Create two lists, one consisting of nodes with known coordinates: $known = \{A, D, C\}$; and one consisting of unknown nodes: $unknown = \{B, E\}$.

Step 4: Both unknown nodes have an equal number of neighbors (3), as can be seen in Table 5.2. Node B is chosen next.

Step 5: The error function err_B of node B is shown in Figure 5.6c. The coordinates of its minimum are $\mathbf{x}_B = (5, 2)$. Figure 5.6d displays the network visualization after including node B.

Step 6: The lists are updated: $known = \{A, D, C, B\}$ and $unknown = \{E\}$.

Step 7: Because list $unknown = \{E\}$ is not empty, **Step 4** is next.

Step 4: Only node E is left, so it is chosen next.

Step 5: The error function err_E in Figure 5.6e shows a minimum at $\mathbf{x}_E = (1, 3)$. Figure 5.6f displays the network visualization after adding node E.

Step 6: The lists are updated: $known = \{A, D, C, B, E\}$ and $unknown = \{\}$

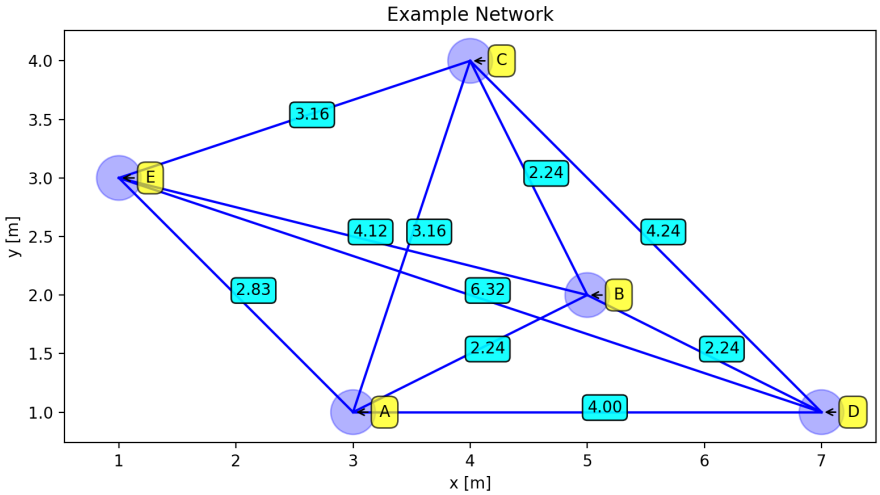


Figure 5.7: Example visualization of a network without any obstacles on the map.

Step 7: The list $unknown = \{\}$ is empty. Stop execution. The algorithm yields:

$$\mathbf{x}_A = \begin{pmatrix} 3 \\ 1 \end{pmatrix};\ \mathbf{x}_B = \begin{pmatrix} 5 \\ 2 \end{pmatrix};\ \mathbf{x}_C = \begin{pmatrix} 4 \\ 4 \end{pmatrix};\ \mathbf{x}_D = \begin{pmatrix} 7 \\ 1 \end{pmatrix};\ \mathbf{x}_E = \begin{pmatrix} 1 \\ 3 \end{pmatrix} \tag{5.4}$$

The results 5.4 are then passed on to the *Visualization* block. Next, Section 5.2.4 describes how the distance matrix is constructed from messages received by the sink node.

5.2.6 Visualization

A separate thread takes care of visualizing the map and the devices, so the localization algorithm can run independently. The `FuncAnimation` object is part of the `matplotlib` package and updates a plot periodically with an update rate of $10\,Hz$. Every time the previous block hands over a new, updated list of coordinates, the following steps are executed:

1. The figure is cleared.
2. A scatter plot of all nodes and their addresses is drawn.
3. Between all nodes that have an active communication link, an edge and the corresponding distance in meter is drawn.
4. All map items are plotted.

An exemplary visualization is shown in Figure 5.7. Section 5.2.5 describes how the positions are calculated.

5.3 Validation Application

The localization application described in Section 5.2 cannot be used to evaluate the accuracy of the algorithm. During normal operation, the localization application would locate all nodes in the network every time an update from one of the nodes is received. In other words, only a single row and column of the network's distance matrix would be updated before calculating the positions. All other matrix elements would remain the same. To evaluate the accuracy of the NetLoc system, it was necessary to write a special Contiki application. Position estimations must be calculated from independent distance matrices. The sink node must receive updates from all nodes in the network, before the localization algorithm calculates new results.

5.3.1 Contiki Validation Application

The application described in Section 5.1.1 was modified to control the collection of distance estimations by the sink. A fourth process called `ranging_process` was implemented to start ranging measurements on demand. The `ranging_process` waits until it is polled by either the server or the client process. After being polled, it iterates over the node's neighbor table and estimates the distance to each neighbor. The neighbor's addresses and the corresponding distances are written to an array, then the `ranging_process` signals back that measurements are finished and blocks until it is polled again.

Both `server_process` and `client_process` wait one minute for the RPL algorithm to converge. After every nodes neighbor table is more or less stable, the collection of data starts as follows:

1. `server_process`: The sink polls the `ranging_process` to estimate distances to all neighbors of the sink. This information is printed in the same way like the function `dao_input_storing()` does, which is described in Section 5.1.2. Instead of the DAO sequence number, a measurement counter variable is used. This counter is later used by the evaluation application to detect the start of a new measurement. Next, the sink creates a ranging request in the form of a UDP packet and sends it to all nodes in its routing table. After every request, the sink waits for the node's response or a timeout event if an error occurred. Once every node in the network responded to the request with their measurements, the sink increments the value of the measurement counter and starts again by ranging to its own neighbors.

2. `client_process`: The client nodes wait for an incoming UDP packet that contains a valid ranging request and the current measurement counter value. The process then polls the `ranging_process` to collect distance estimations, builds a UDP packet containing the node's measurement data and send the packet back to the sink. Then the client node waits for the next request.

By merging all measurements with the same measurement counter value into one distance matrix respectively, the position estimations can be independent from each other.

5.3.2 Logfile Analyzer

This application uses the same localization algorithm as the one described in Section 5.2.5 to calculate the node's positions. Its inputs are:

1. The sink's log file from the test measurement. It should contain a sufficiently large amount of measurements to make a meaningful statistical analysis.
2. A map of the environment.
3. The addresses of three reference nodes as described in Section 5.2.5.
4. A list of all nodes that are part of the measurement together with their real coordinates on the map. These coordinates are taken as references to calculate the positioning errors.

If all inputs are provided, the application starts by reading and parsing the log file. In contrast to the actual localization application, which maintains and updates a single distance matrix instance, this evaluation application creates a new distance matrix for every set of measurements that share the same measurement counter value (see Section 5.3.1).

At this point, the application has a list of distance matrices where every matrix element represents a unique distance measurement between two nodes. These distance measurements are now compared to the actual distances between the reference positions to get the error of the distance measurements. This way, the average distance error per link is calculated. Knowing the average error for every link of a node, the average error for all distance measurements can be calculated. This mean error of all distance measurements on all links gives an estimation how well a node's radio is calibrated.

The next step is to run the localization algorithm on every distance matrix. For every distance matrix, the result is a set of node positions. The Euclidean distance between a reference position (x_{ref}, y_{ref}) and a calculated position (x_{cal}, y_{cal}) is the position error e_P shown in Eq. 5.5.

$$e_P = \sqrt{(x_{ref} - x_{cal})^2 + (y_{ref} - y_{cal})^2} \tag{5.5}$$

For each node, a list of position errors is then sorted by error value to get the Cumulative Distribution Function (CDF).

5.4 Limitations

The error function can have more than one extremum. When all neighbors of a node are arranged in one straight line, then there is no distinct solution. However, this situation is highly unlikely if there are more than two neighboring nodes. If a node has only two neighboring nodes, there are two equally likely solutions. If a node has only one single neighbor, there are an infinite number of possible solutions. The result depends on the initial guess of the `minimize` function.

This limitation applies to all state-of-the-art systems. One way to cope with such a limitation is to add additional constraints to the optimization problem definition. In the NetLoc localization algorithm an additional constraint is considered by adding a penalty term to the error function.

5.4.1 Penalty Term

Because the signal strength decreases as the distance to the signal source increases, it is more likely for a node to have links to nodes that are close than to nodes that are farther

$d\ [m]$	A	B	C	D	E
A	0.00	2.24	3.16	4.00	2.83
B	2.24	0.00	2.24	2.24	0.00
C	3.16	2.24	0.00	4.24	3.16
D	4.00	2.24	4.24	0.00	0.00
E	2.83	0.00	3.16	0.00	0.00

Table 5.3: Distance matrix of a network graph. Node E has only two links to neighbors.

away. This constraint can be considered during optimization by adding a penalty term to the error function. Its purpose is to increase the error function at positions that are less likely, thus decreasing the probability that the *minimize* function converges in the wrong extremum. If a node has less than three links, it is assumed that other nodes are at least as far away as the farthest neighbor.

A node n measures K distances, where d_{max} is the maximum distance.

$$d_{max} = max(d_{n,0}, d_{n,1}, ..., d_{n,K})$$

The error function is extended by considering not only K neighbors at positions $P_N = \{x_0, x_1, ..., x_K\}$ but L remaining nodes at positions $P_R = \{x_0, x_1, ..., x_L\}$ as well. P_R is the set of nodes, who's positions are already known, but have no link to node n. Eq. 5.2 is extended to Eq. 5.6 if $K < 3$.

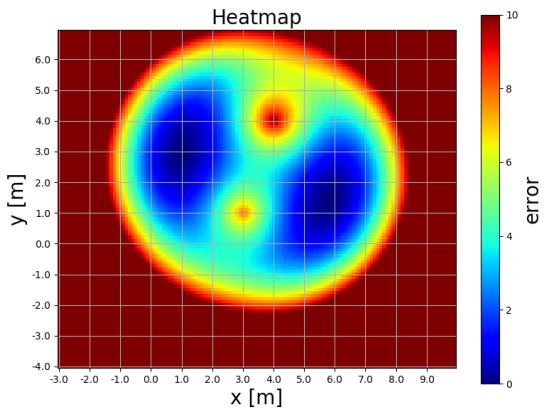
$$err_n(\mathbf{x}) = \begin{cases} \sum_{k=0}^{K} \left[d(\mathbf{x}, \mathbf{x_k}) - d_{n,k}\right]^2 + \sum_{l=0}^{L} [d(\mathbf{x}, \mathbf{x_l}) - d_{max}]^2 & \text{if } d(\mathbf{x}, \mathbf{x_l}) < d_{max} \\ \sum_{k=0}^{K} \left[d(\mathbf{x}, \mathbf{x_k}) - d_{n,k}\right]^2 & \text{else} \end{cases} \tag{5.6}$$

Example

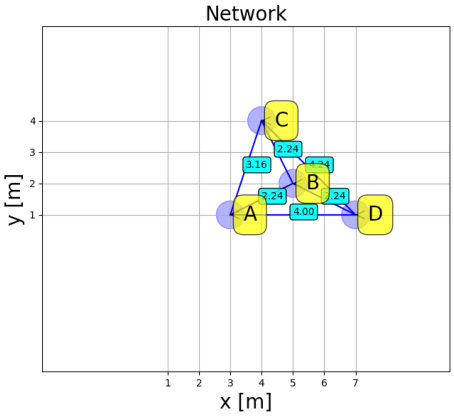
Node E must be located based on the distance matrix shown in Table 5.3. Nodes A, B, C and D have been located already. Figure 5.8b shows the network of all previously located nodes next to the error function according to Eq. 5.2 in Figure 5.8a. Adding the penalty term leads to the error function err_E as defined in Eq. 5.6.

$$err_E(\mathbf{x}) = \left[d(\mathbf{x}, \mathbf{x}_A) - 2.83\right]^2 + \left[d(\mathbf{x}, \mathbf{x}_C) - 3.16\right]^2 + \left[d(\mathbf{x}, \mathbf{x}_B) - 3.16\right]^2 + \left[d(\mathbf{x}, \mathbf{x}_D) - 3.16\right]^2 \tag{5.7}$$

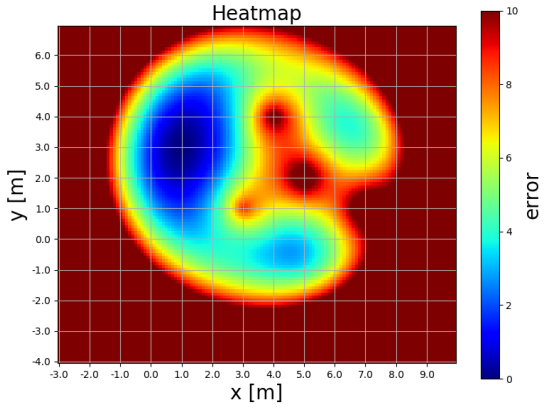
Figure 5.8c displays err_E according to Eq. 5.7. The error function has multiple extrema, but only one global minimum. Due to the penalty term, there is a distinct solution for the position of node E at $\mathbf{x}_E = (1, 3)$. Figure 5.8d shows the network after all nodes have been successfully located.



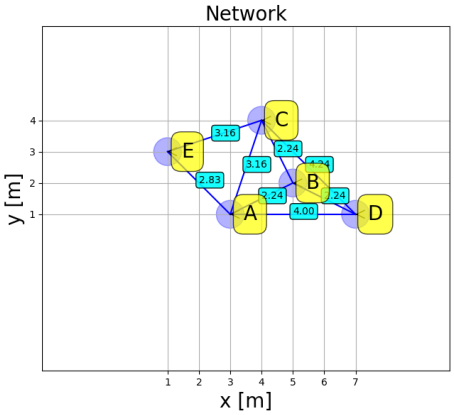
(a) Error function without penalty term, where node E has only links to node A and node C.



(b) Nodes A, B, C and D that have already been located.



(c) err_E **with** penalty term. The global minimum of the error function is at $(1, 3)$.



(d) Node E at the global minimum of err_E.

Figure 5.8: Comparison of err_E with and without penalty term.

Chapter 6

Evaluation

In this chapter, the NetLoc system performance is evaluated. A network of nodes was set up in a laboratory environment to capture enough measurement data for the statistical analysis of the position accuracy. Section 6.1 describes the laboratory setup of the network. In Section 6.2 the evaluation of the localization algorithm and the results are discussed.

6.1 Experimental Setup

A network consisting of nine nodes was set up in a room of $10\,m\,x\,6.3\,m$. All nodes in the network were programmed with the validation application, which is described in Section 5.3. Each node's radio was configured according to Table 6.1. The node's radios were not calibrated individually, but all loaded with the same calibration data. This was taken into account during data analysis as described in Section 6.2. Figure 6.1 shows the position of all nodes in the network. During test execution, the node's positions were static. The positions were chosen such that line-of-sight communication was possible between any pair of nodes. There were two types of antennas used in the network. The nodes with the addresses `a523`, `9d23`, `9f23`, `34c6`, `11c6`, and `8e03` were built as described in Chapter 4 and had their antenna connected via SMA. The remaining three nodes with the addresses `4010`, `4210`, and `4443` had a DWM1000 [27] breakout board (instead of the DW1000 extension shield) with a smaller, integrated ceramic antenna. A host PC was connected to node `a523`, which acts as the network's sink node, to save the log file and to power the sink node. To avoid node failures because of drained batteries, all nodes had a wired power source.

Parameter	Value
Channel	4
Center Frequency	3993.6MHz
Bandwidth	1331.2MHz
PRF	16MHz
Bit rate	6.8Mb/s
Preamble length	128
Preamble code	7

Table 6.1: Radio configuration during evaluation.

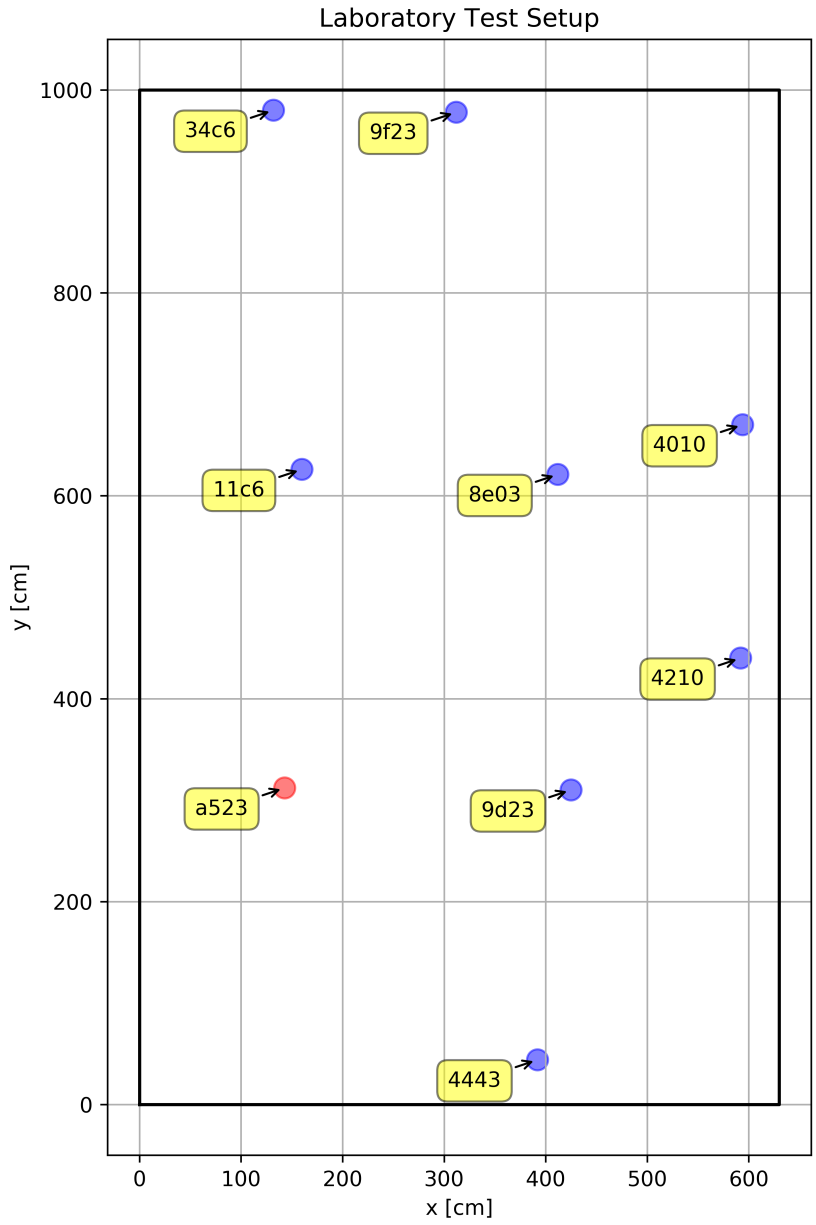


Figure 6.1: A map of the laboratory test setup for the NetLoc evaluation. The room borders are represented by black lines. Colored dots mark the node's positions. The node addresses are shown in boxes next to their position. Node `a523` is the network's sink node and therefore the only fixed position. The sink node is colored red, nodes that must be located are colored blue.

address =	$(x[m]$	$y[m])$
a523 =	(1.43,	3.12)
9d23 =	(4.25,	3.10)
9f23 =	(3.12,	9.78)
34c6 =	(1.32,	9.80)
11c6 =	(1.60,	6.26)
8e03 =	(4.12,	6.21)
4010 =	(5.94,	6.70)
4210 =	(5.92,	4.40)
4443 =	(3.92,	0.44)

Table 6.2: Reference coordinates of all nodes in the network.

6.2 Performance of the Localization Algorithm

The log file containing all measurements was saved on the host PC, which was connected to the sink node. After gathering measurement data, the *Logfile Analyzer* application described in Section 5.3 was started. The inputs to the *Logfile Analyzer* application were:

1. The log file, saved on the host PC, which contains the sink node's serial output.
2. A simple map, containing the walls of the $10\,m\,x\,6.3\,m$ room where the network was located.
3. The three addresses `a523`, `9d23` and `34c6`, which are the reference nodes as described in Section 5.2.5.
4. Table 6.2, listing the reference coordinates of all nodes on the map.

6.2.1 Evaluating Distance Measurements

The validation application collected 17 858 distinct neighbor tables each consisting of up to eight distance measurements. If one of the neighbor tables was lost due to a packet collision, the whole distance matrix was ignored during the evaluation. This step is necessary to have comparable measurements. From these 17 858 distinct neighbor tables the *Logfile Analyzer* application then generated 2 229 distance matrices. This set of distance matrices will be used for further analysis. As a first step, distance measurements are statistically analyzed.

Distance Measurement Analysis. If a node's radio was not calibrated properly, the distance measurements of this node can be corrected. This is achieved by calculating the average error for every link of a node and then derive the average of all these error values. This is done only once during an initial phase. Table 6.3 shows the exemplary distance measurement analysis of node `9f23`. Note that nodes `4010`, `4210`, and `4443` have a comparably higher error, which can be explained by the fact that they use a different antenna. The mean error of node `9f23` according to Table 6.3 is $-0.420\,m$.

Neighbor address	$d_{meas}[m]$	N_{meas}	$d_{ref}[m]$	$error[m]$
a523	7.167	1985	6.871	-0.296
9d23	7.102	1973	6.775	-0.327
8e03	4.058	1981	3.707	-0.351
34c6	2.262	1977	1.800	-0.462
11c6	4.160	1981	3.834	-0.326
4010	4.628	1968	4.180	-0.448
4443	10.024	1954	9.377	-0.648
4210	6.568	1954	6.068	-0.501

Table 6.3: A list of all neighbors of node 9f23, the average measured distance d_{meas} to each neighbor, the number of measurements N_{meas}, the reference distance d_{ref}, and the *error* between the average distance and the real distance. It is noticeable that on average all distance measurements were too high.

Node address	$d_{correction}$ $[m]$
a523	-0.433
9d23	-0.425
8e03	-0.406
9f23	-0.420
34c6	-0.432
11c6	-0.376
4010	**-0.522**
4443	**-0.530**
4210	**-0.548**

Table 6.4: Distance correction $d_{correction}$ for every node after analyzing all distance measurements. The highlighted nodes use a different antenna and therefore have a higher error.

The same analysis is done for every node in the network, yielding the correction values that are listed in Table 6.4. If all nodes were calibrated correctly, the mean error and the distance correction value should be close to zero. Knowing the correction values in Table 6.4, the log file is parsed again taking these correction values into account. Alternatively, the whole measurement could have been repeated after re-programming each node with a modified application that uses the individual correction values from Table 6.4. Figure 6.2 shows all distance measurements by node 8e03 to its neighbors after correcting the measurements according to Table 6.4. The plots show that distance measurements to neighbors 4210 and 4443 are close to the actual distance, whereas distance measurements to neighbor 11c6 have a high error. Note that the measurement precision varies as well depending on the neighbor. Measurements between nodes 8e03 and 11c6 are very precise, whereas measurements between 8e03 and 9f23 have a higher variance.

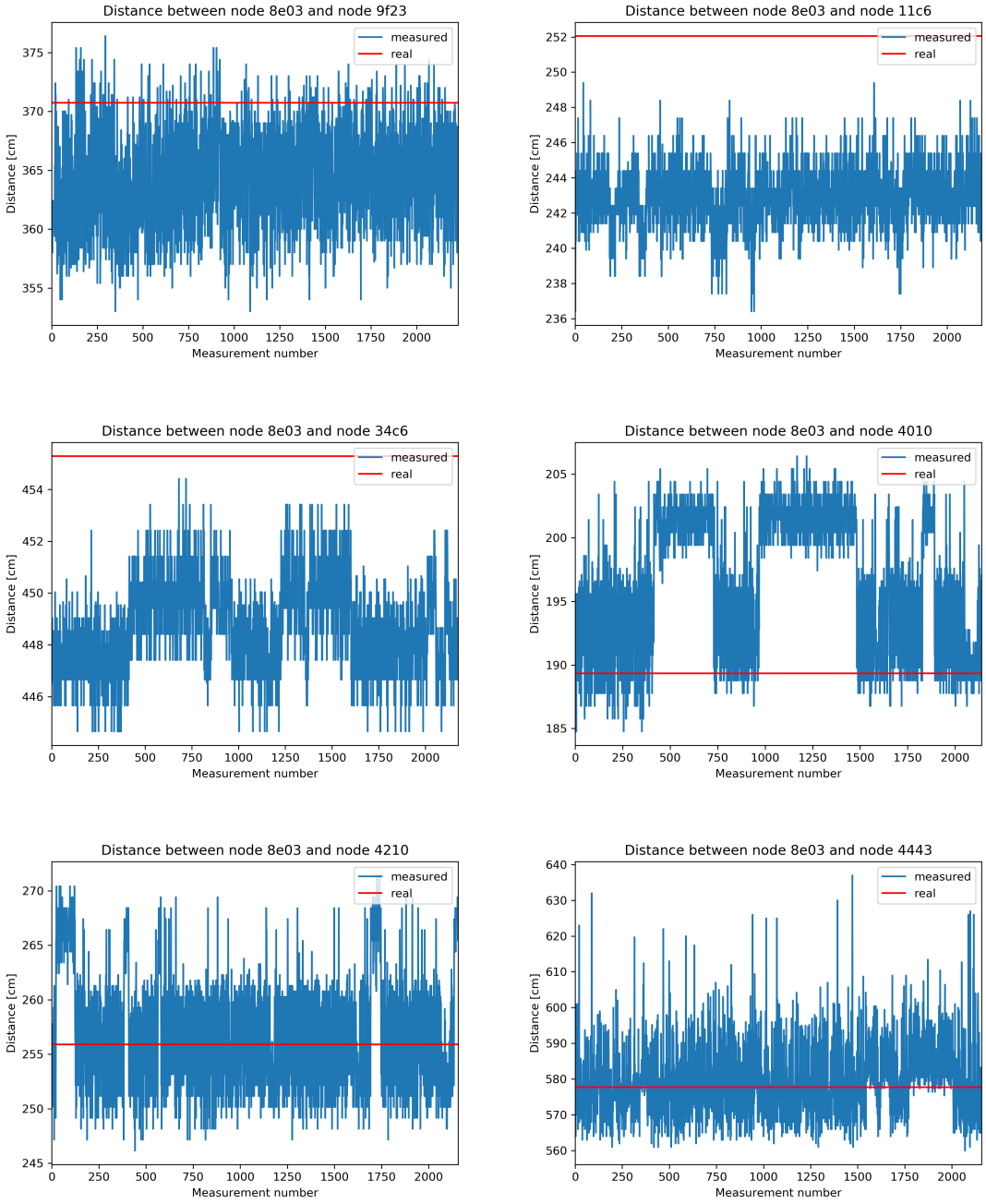


Figure 6.2: Distance measurements of node `8e03` to its neighbors `9f23`, `11c6`, `34c6`, `4010`, `4210`, and `4443`.

6.2.2 Evaluating Position Estimations

After the evaluation of measured distances as described in Section 6.2.1, a list of 2 229 distance matrices were available. For every distance matrix in that list, the node's positions are estimated, resulting in a list of 2 229 sets of coordinates. Figure 6.3 shows all position estimates in a single plot. The reference positions are indicated by crosses, position estimates are drawn as dots. Node `a523` is the sink node, hence there is no position variation. The very small variance of node `9d23` is the consequence of having only one degree of freedom. All nodes except `a523` and `9d23` have two degrees of freedom. Despite the low variance of the position estimates, there are some outliers that are far off the actual position. For an unknown reason, some distance measurements were far off from the actual distance. If a node's position estimate has a high error because of such a distance measurement, all subsequent position estimates have an high error too.

Further analysis is based on the list of coordinate sets. Each set of coordinates contains one position estimate of every node on the map. In the next step a second list is generated, holding the position errors for every set of coordinates. Every position error is the Euclidean distance between the estimated position and the real node position, which is listed in Table 6.2.

These position errors are now visualized to evaluate the position accuracy. The cumulative distribution function (CDF) is calculated and displayed for two exemplary nodes as well as combined for all nodes that have two degrees of freedom. Figure 6.4 shows the CDF of all position errors except for the position errors of node `9d23` and the sink node `a523`, which have less than two degrees of freedom. The CDF in Figure 6.4 proves that 96 percent of the position estimates have an error of less than 30 cm.

Figure 6.5 shows the two CDFs of the most accurate and the least accurate nodes. The CDF of node `8e03` as well as its absolute position errors are shown in Figure 6.5a. Out of all nodes in the network, node `8e03` had the most accurate results. Because node `8e03` is surrounded by other nodes and not like node `4443` located at the network's verge, the error function of `8e03` has a more distinct shape than the error function of node `4443`. The same effect can be observed when comparing Figure 5.6c to Figure 5.6a. Therefore, the optimization algorithm is more likely to converge close to the error functions global minimum. More than 95 percent of all position errors were smaller than 20 cm. The worst position accuracy was calculated for node `4443`. Figure 6.5b shows that almost no position error was smaller than 10 cm. This is clearly due to a bad calibration of node `4443`. However, over 90 percent of node `4443`'s position errors were still smaller than 30 cm. By comparing both plots of absolute errors in Figure 6.5a and Figure 6.5b one can see outliers at the same measurement numbers. From measurement number 1500 to 1750 only node `4443` has outliers. It can be assumed that one neighbor of node `4443` was located wrong before, but the neighbors of node `8e03` were located correctly.

6.2.3 Qualitative Evaluation of Multi-Hop Localization

In a qualitative experiment the multi-hop localization capability was tested. For this purpose the localization application described in Section 5.2 was used instead of the validation application. Figure 6.6 shows a network of eight nodes. The four nodes `a523`, `9d23`, `9f23`, and `34c6` were located in the same room. The room's walls are shown as black lines in Figure 6.6. The four nodes `11c6`, `4010`, `4210`, and `8e03` were located in the next room with

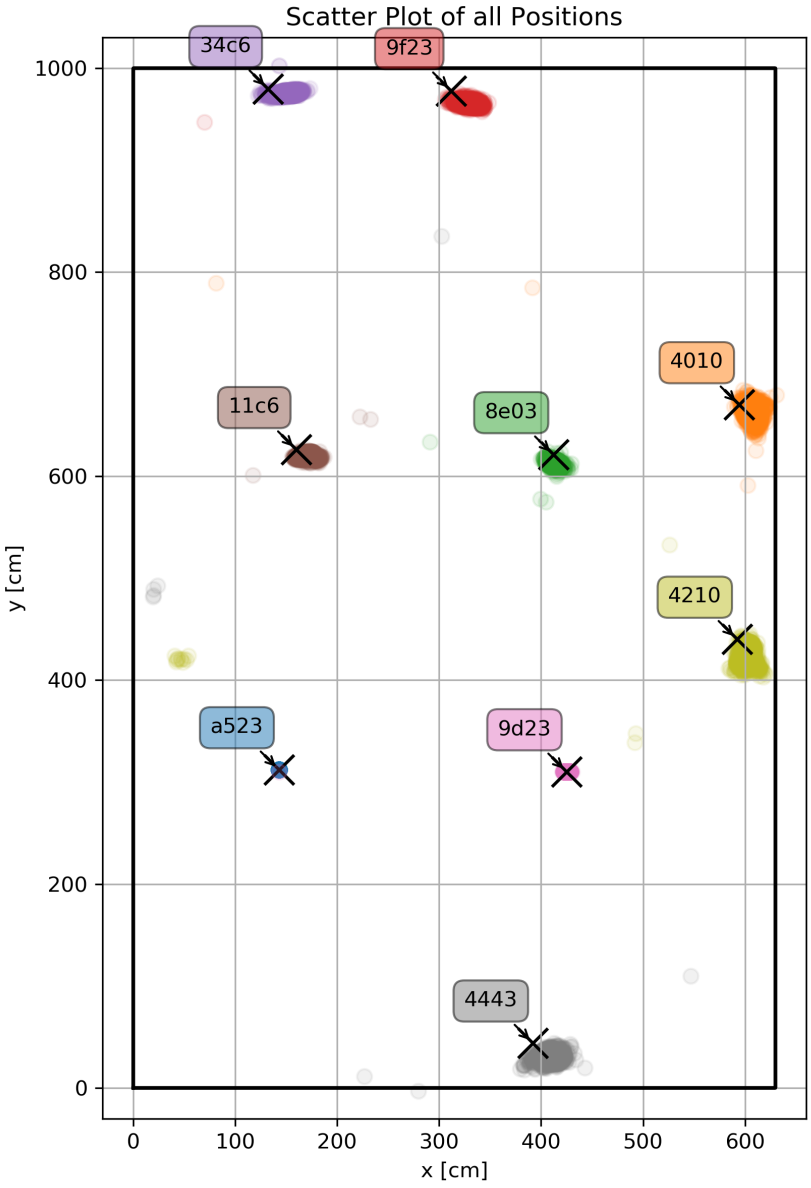


Figure 6.3: Scatter plot of all calculated positions.

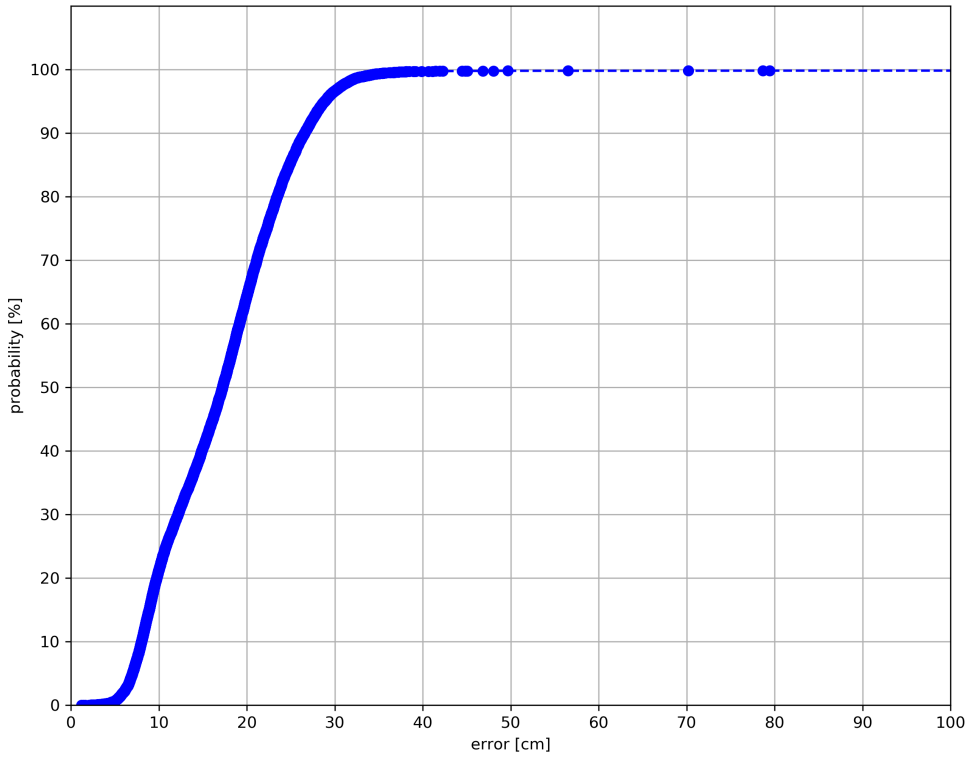
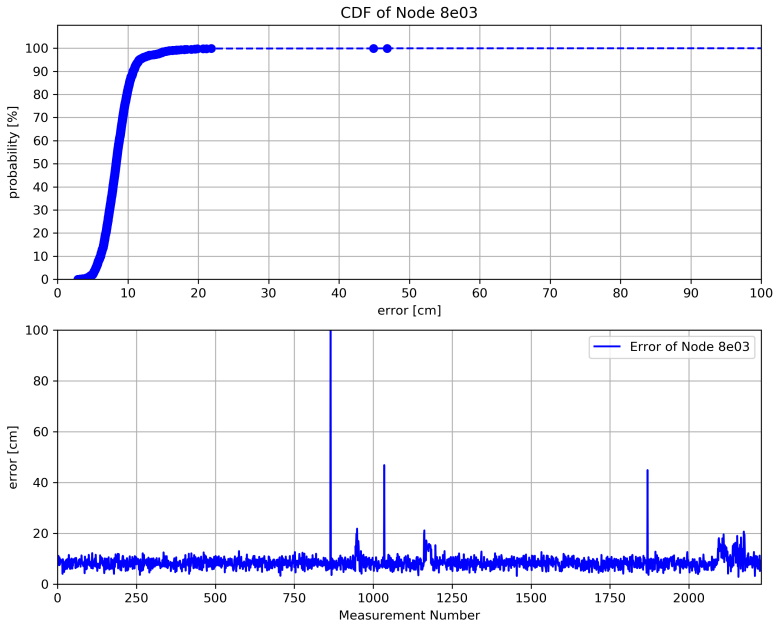
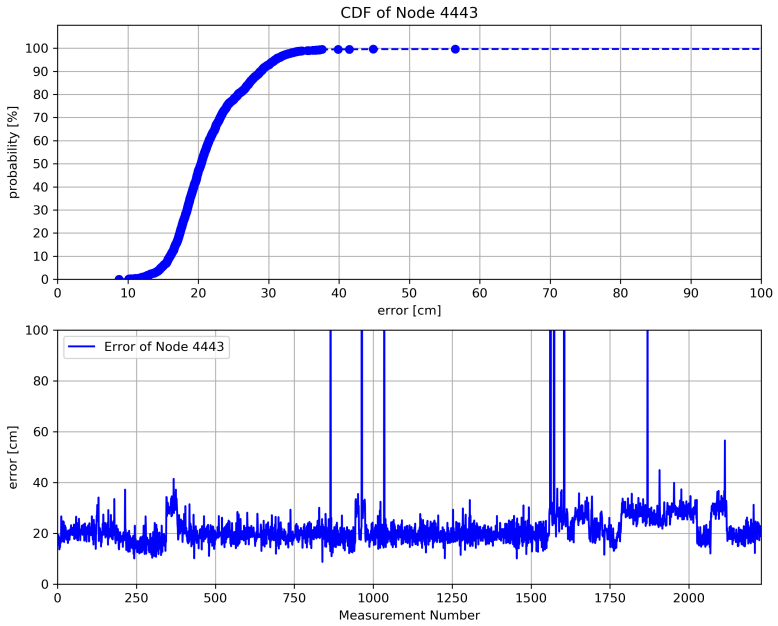


Figure 6.4: CDF of all nodes that have two degrees of freedom.



(a) Node `8e03` had the highest accuracy of all nodes in the network. 90% of all position estimates are well below 20 cm.



(b) Node `4443` had the worst accuracy of all nodes in the network. Hardly any position error was smaller than 10 cm.

Figure 6.5: CDFs and absolute position errors of nodes `8e03` and `4443`.

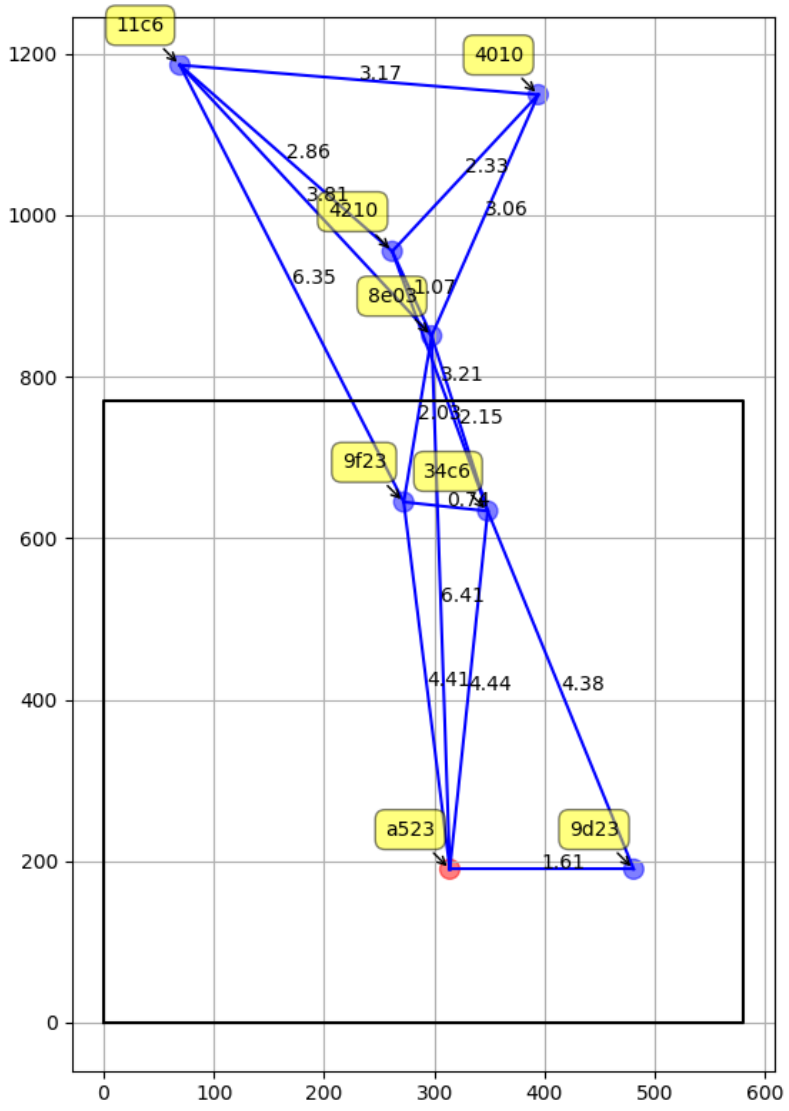


Figure 6.6: Multi-hop localization of nodes in a network. The unit on both axis is centimeter. The sink node's position is marked with a red dot, mobile nodes are shown as blue dots. A line between two nodes indicates that these nodes have a communication link. The distances between two nodes in meter is shown next to the links. The shown figure is a single shot of a live localization test, which shows that NetLoc works also with broken links.

a concrete wall in between. Note that node `4010` has only communication links to nodes that are in the same room, but the localization application was still able to calculate the node's position.

The ground-truth positions of the nodes are not available making it impossible to evaluate the position error. The visualization indicates an error of less than one meter. Further validation measurements of multi-hop networks were out of scope of this thesis. Figure 6.6 shows just one trial of locating nodes while moving them to different rooms.

6.2.4 Comparison to Spring Layout Algorithm

A computational efficient algorithm for drawing graphs was validated as well. The Fruchterman-Reingold algorithm [38], also known as spring layout algorithm, calculates coordinates for nodes in a network by solving an optimization problem. The input for the algorithm is the networks distance matrix. Node positions are initialized randomly at first. The network nodes are modeled as physical masses that are connected to each other by links modeled as springs. Each spring applies a force on both masses, if its length differs from the distance in the distance matrix. Node positions are updated according to the forces that push and pull them. Solving the optimization problem minimizes the overall stored energy in the network. The energy is zero, if every spring has the exact same length as given by the distance matrix.

Three different implementations of this algorithm from the Python package *networkx* and the *DOT* package were tested. Unfortunately, all performed very poorly in terms of position accuracy. Figure 6.7 shows a scatter plot of all positions, when replacing NetLoc's optimization algorithm with the spring layout algorithm. Compared to Figure 6.3 the accuracy and the precision are much worse. The accuracy gets better if the initial node positions are close to the actual reference positions, but the precision does not. One possible reason is that the tested implementations are designed for plotting and visualizing large network graphs, not for localization.

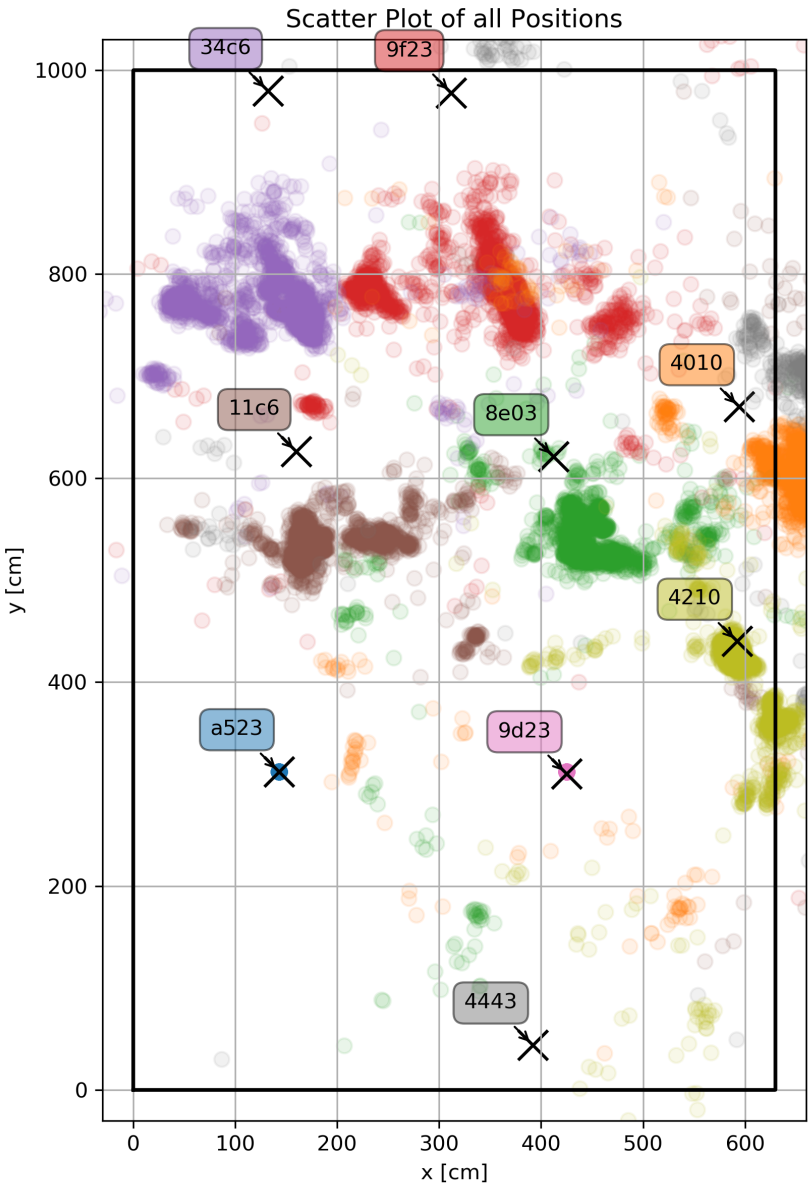


Figure 6.7: Scatter plot of all calculated positions when using the spring layout algorithm.

Chapter 7

Conclusion and Outlook

In this thesis I presented NetLoc, a localization system for IoT devices and a new, low-cost UWB based platform design, which is ideal to develop IoT and localization applications.

UWB IoT Platform. The new platform design, introduced in Chapter 4, proves that it is possible to build low-cost IoT devices based on UWB technology. The transmit power and bandwidth calibration results show that the platforms performance is comparable to the manufacturer's reference design.

Furthermore, the design allows upgrades of all hardware components, should a future application require specific features that are not supported yet.

Localization Algorithm. The advantage of low-power, high accuracy distance measurements that comes with IR-UWB technology allows precise position estimates. In this thesis I showed that high position accuracy does not necessarily require a large network infrastructure. The evaluation of the localization algorithm shown in Chapter 6 proves decimeter accuracy even for independent[1] measurements in networks using a single anchor. Another advantage of the localization algorithm is that it does not require map information, but benefits from it. Nodes can be located relative to the sink node even without map information. The localization application is able to locate nodes that do not have a communication link with the sink node, but are multiple hops away. Although distance measurements are very accurate, it should be considered that the errors of position estimates accumulate if a node is multiple hops away from the sink node. A possible solution to this problem is to add multiple reference nodes at known positions, so the algorithm can correct all coordinates in a post-processing step after the localization of the network.

Scalability. Unlike anchor based localization systems, NetLoc performs even better in dense networks where nodes have many neighbors. The communication load to measure distances between nodes and then forwarding this information for further processing is distributed across all nodes in the network. The computational complexity of calculating coordinates from large distance matrices can be done by more powerful computers.

[1] Independent, meaning the localization algorithm is not initialized with the previous coordinates, but with random coordinates.

Embedding Distance Information in the Routing Protocol. IoT devices are required to be low-energy and are often battery powered. Embedding the process of distance measurements and data collection into the routing protocol is a very efficient solution to the problem of constrained energy. No additional packets need to be sent, neither to measure distances, nor to collect the distance information at the sink node. Updating distance measurements based on the Trickle timer makes a lot of sense, because the update interval is constantly adjusted to the dynamics of the network. This way, distances are measured in increasing intervals, if communication links are static, and in decreasing intervals if communication links are dynamic.

Standard-Compliant Protocols. The implementation in this thesis is built on top of standardized communication protocols. Hence, making it portable and compatible with other standard-compliant systems. Furthermore, standardized communication protocols are rarely changed making the NetLoc system compatible with future IoT systems as well.

Position Update Rate. Depending on the configuration of RPL's Trickle timers, the update rate varies. DIO messages, and therefore distance measurements, are triggered by a Trickle timer that adapts to link dynamics. If communication links are steady the timer adapts and increases its period, but if links are dynamic and volatile, the timer decreases its period. Thus positions are updated at a higher rate. In my experiments, the maximum Trickle timer period was configured to be 15 seconds. If required, the update rate can be increased to be multiple times per second by implementing additional application layer distance measurements.

Evaluation. The evaluation showed that the accuracy of the localization algorithm is comparable to anchor based state-of-the-art solutions. It is expected that the results can be further improved by properly calibrating all radios, which was not in the scope of this thesis.

7.1 Future Work

Future work includes the improvement of NetLoc's localization algorithm, which is described in Section 7.1.1. Section 7.1.2 discusses the pros and cons of 2D vs. 3D localization. Furthermore, a proper MAC protocol would optimize the power consumption of the NetLoc hardware platform. Section 7.1.3 describes the challenges of UWB MAC protocols and how they can be addressed.

7.1.1 Improving Localization Algorithm

The current localization algorithm can be improved and optimized in the following ways:

Sensor Fusion. Utilizing multiple sensors to measure the magnetic field, inclination, acceleration, air pressure and altitude to improve the position accuracy is referred to as sensor fusion. Not relying solely on the radio to determine the position of a device makes the system more robust and more accurate. As mentioned in Chapter 4, the platform

was designed in a way that it can easily be extended with a board that combines multiple sensors.

Furthermore, distance measurements can be triggered by accelerometers instead of Trickle timers, so positions are only updated if nodes are actually in motion.

Client-Based Localization. The current implementation estimates positions on a host PC. This computational load can be distributed to all nodes in the network. Nodes would broadcast their position to neighboring nodes, which combine distance measurements and their neighbor's positions to estimate their own position.

Divide and Conquer. The localization algorithm could be optimized by dividing the network into small clusters of fewer nodes, estimate the positions of all nodes within the cluster, and then combine them again until all nodes in the network have been located. Furthermore, it would be easier to distribute the computational load onto multiple processors.

LOS detection and SLAM. The DW1000 datasheet suggests a way to determine on the receiver whether the transmitting node was in line-of-sight (LOS) or not [26, Chapter 4.7]. In a dense network with many communication links, this LOS indicator can be used to detect obstacles on a map. By detecting a lot of obstacles and marking their positions on a map, the map information gets updated and extended during the application's runtime.

Map Constraints. In this thesis, the map information was not considered by the localization algorithm. By adding equality and inequality constraints to the optimization problem that is solved by the algorithm, the position accuracy and the algorithm robustness can be improved. For example, the space of possible positions can be reduced to the feasible positions on a map.

Filtering. Neither position estimates nor distance measurements are currently filtered. Low-pass filtering distance measurements or applying a Kalman filter [44] on position estimates could improve the algorithm's performance.

7.1.2 2D vs. 3D Localization

The decision to support two-dimensional localization was made for several reasons. First, because most maps as well as most use cases (e.g. indoor navigation systems) are two-dimensional. Although, multi-level maps and multi-story buildings are the common case, the required accuracy in terms of altitude is very low. An indoor navigation system, for example, does not need to know a device's altitude at decimeter precision. Knowing the level on which the device is located is sufficient to determine its position. NetLoc's localization algorithm can be extended with little effort to calculate three-dimensional coordinates, if an application requires this. Note, that one more neighbor (four neighbors in three dimension, three neighbors in two dimensions) is needed in three dimensions to have a unique solution for a position estimate. If devices move in three dimensions (e.g. flying drones) or must be located on a three dimensional map, it is necessary to support 3D localization.

7.1.3 UWB MAC protocol

The MAC protocol `nullmac` mentioned in Section 3.2.2, which is used in NetLoc wastes energy because the radio is not duty-cycled and packet collisions are ignored. Contiki does not offer a proper MAC protocol for UWB. Furthermore, the advantages of UWB signals of being hard to detect by scanners and being highly immune to jamming, make it challenging to find an efficient MAC protocol. Narrow-band radio technologies can sense, if the channel is clear, which is an ability that UWB radios do not have. MAC protocols that are based on carrier sense multiple access (CSMA) and clear channel assessment (CCA) techniques, hence are not suitable to be used with UWB radios. The task of a MAC protocol is to avoid energy waste by dealing with the following four problems:

1. Collision avoidance. Two devices sending at the same time causes a packet collision and the loss of both packets.
2. Useless listening. Turning on the receiver, when there is no one sending is a waste of energy.
3. Overhearing. Receiving packets that were meant for someone else.
4. Protocol overhead. Additional data like MAC headers or beacon packets that must be exchanged to coordinate the medium access.

We compared publications of various different MAC protocols for UWB. The most promising solutions for a MAC protocol were PMAC [12], Widemac [69], and FrameCommDM [2], but implementations for Contiki were not available. Because we were unable to find a suitable MAC protocol implementation for NetLoc, we decided to use the `nullmac` protocol, which does not rely on carrier sensing techniques. Implementing an energy efficient MAC protocol for NetLoc is yet to be done.

Appendix A

Hardware Design

A.1 Pin Usage and Jumpers

Table A.2 lists the names of the Arduino header connector pins. The communication buses SPI, I2C and UART are bold. The UART and I2C communication ports were left unused on the UWB extension shield so that the WiFi and MEMS boards can be used simultaneously. Three boards are listed in the table:

1. NUCLEO-L152RE/STM32L152RE
   The labels of this board's connectors are CN6, CN8, CN5 and CN9. In Figure 4.2a, CN6 is connected to the UWB shield with the upper left, CN8 with the lower left connector, CN5 with the upper right and CN9 with the lower right. The pin names of the STM32L152RE are taken from the datasheet [62].

2. X-NUCLEO-IDW04A1
   The WiFi shield has multiple GPIO pins, but GPIO9, GPIO13, GPIO14 and GPIO15 cannot be used because they are reserved for the UWB shield's SPI pins. The X-NUCLEO-IDW04A1 is controlled using a UART bus interface.

3. UWB Shield
   The DW1000 can only be controlled using the SPI bus. Therefore, using the SPI pins cannot be avoided on the UWB shield. The GPIOx pins of the UWB shield are optional. They are only connected to the CN9 header pins if the jumpers GPIO0 - GPIO6 are closed (see Table A.1)

The X-NUCLEO-IKS01A1 pins are not listed in Table A.2. It is controlled using the I2C bus of CN5 and powered through CN6. Table A.1 lists all the jumpers of the UWB shield.

P1. This jumper was added to the design to simplify current measurements. The supply current of the UWB shield can be measured by connecting an ampere-meter to the pins P1-1 and P1-2. P1 connects the voltage supply pin CN6-4 (see Table A.2) to the power plane (see Table 4.3) and must be closed for normal operation.

Jumper	Pin	Net Name	Description
P1	P1-1	+3V3 plane	positive voltage supply of UWB shield
	P1-2	+3V3 pin	positive voltage output of NUCLEO-L152RE header
P2	P2-1	SPI-CS2	alternative (2nd) SPI chip select
	P2-2	SPI-CS	SPI chip select of DW1000
	P2-3	SPI-CS1	default SPI chip select
P4	P4-1	LEDs cathodes	connected to the cathodes of all 4 LEDs
	P4-2	GND	connected to the ground plane of the UWB shield
GPIO0-GPIO6	GPIO0-1	DW1000 pin 38	connected to GPIO0 of DW1000 (LED_RXOK)
	GPIO0-2	Pin 8 of CN9	connected to pin number 8 of header CN9
	GPIO1-1	DW1000 pin 37	connected to GPIO1 of DW1000 (SFDLED)
	GPIO1-2	Pin 7 of CN9	connected to pin number 7 of header CN9
	GPIO2-1	DW1000 pin 36	connected to GPIO2 of DW1000 (RXLED)
	GPIO2-2	Pin 6 of CN9	connected to pin number 6 of header CN9
	GPIO3-1	DW1000 pin 35	connected to GPIO3 of DW1000 (TXLED)
	GPIO3-2	Pin 5 of CN9	connected to pin number 5 of header CN9
	GPIO4-1	DW1000 pin 34	connected to GPIO4 of DW1000 (EXTPA)
	GPIO4-2	Pin 4 of CN9	connected to pin number 4 of header CN9
	GPIO5-1	DW1000 pin 33	connected to GPIO5 of DW1000 (SPIPHA)
	GPIO5-2	Pin 3 of CN9	connected to pin number 3 of header CN9
	GPIO6-1	DW1000 pin 30	connected to GPIO6 of DW1000 (SPIPOL)
	GPIO6-2	Pin 4 of CN9	connected to pin number 3 of header CN8

Table A.1: List of Jumpers and their purpose.

Pin	STM32L152RE	WiFi Shield	UWB Shield
	CN6 (Power)		
1	NC		
2	3V3		NC
3	Reset Button		NC
4	3V3		
5	5V		NC
6	GND		
7	GND		
8	VIN		NC
	CN8 (Analog)		
1	PA0/ADC_IN0	NC	nRST
2	PA1/ADC_IN1	NC	**CS2**
3	PA4/ADC_IN4	NC	*GPIO6*
4	PB0/ADC_IN8	GPIO6	SYNC/GPIO7
5	PC1/**SDA**	GPIO1	NC
6	PC0/**SCL**	NC	
	CN5 (Digital)		
10	PB8/**SCL**	GPIO5	NC
9	PB9/**SDA**	GPIO4	NC
8	AVDD		NC
7	GND		
6	PA5/**SCK**	GPIO15	**SCK**
5	PA6/**MISO**	GPIO13	**MISO**
4	PA7/**MOSI**	GPIO14	**MOSI**
3	PB6/**CS**	NC	**CS1**
2	PC7	GPIO9	**IRQ**
1	PA9	GPIO2	WAKEUP
	CN9 (Digital)		
8	PA8	WiFi RST	*GPIO0*
7	PB10	PB10	*GPIO1*
6	PB4	PB4	*GPIO2*
5	PB5	PB5	*GPIO3*
4	PB3	NC	*GPIO4*
3	PA10	GPIO11	*GPIO5*
2	PA2/**UART_TX**	GPIO11/**UART_RX**	NC
1	PA3/**UART_RX**	**UART_TX**	NC

Table A.2: Pin usage of header connectors.

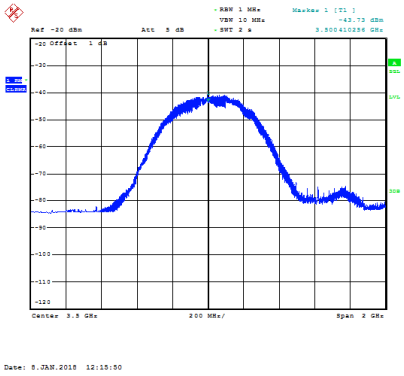
P2. The center pin P2-2 is connected to the DW1000's SPI chip select pin. By closing P2-2 and P2-3 (on the right), the CN5-3 pin (see Table A.2) is used as the SPI chip select. This is the default use-case. If P2-1 and P2-2 is closed instead, a second UWB shield (or any other shield that uses pin CN5-3 as SPI chip select) can be stacked onto the same NUCLEO-L152RE board. Then the UWB shield can still communicate by using CN8-2 as SPI chip select.

P4. There are 4 LEDs that can be soldered on the UWB shield to indicate the communication state. If GPIO0 - GPIO3 have to be used, the LEDs must be disconnected by opening P4. This also helps saving energy.

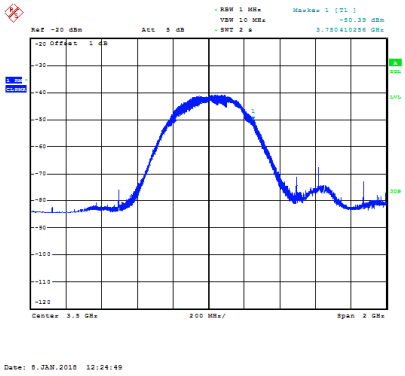
GPIO0 - GPIO6. There are 8 GPIOs on the DW1000 that can be controlled by the firmware. GPIO7 is connected to CN8-4. GPIO0 to GPIO6 are accessible to the user in two ways. One way is to attach measurement probes to the test points on the edge of the board. The other way would be to solder a pin header onto these test points or shorten them using solder bridges (in Figure 4.1b the pin header is soldered onto the board). Then the GPIOs are accessible to the NUCLEO-L152RE board via its CN9 header pins as well. Closing these jumpers is optional. By closing the GPIOx jumpers, the DW1000's GPIO0 - GPIO5 pins are connected to the CN9-8 - CN9-3 pins and GPIO6 is connected to CN8-3 (see Table A.2).

Appendix B

TX Power and Bandwidth Calibration Results

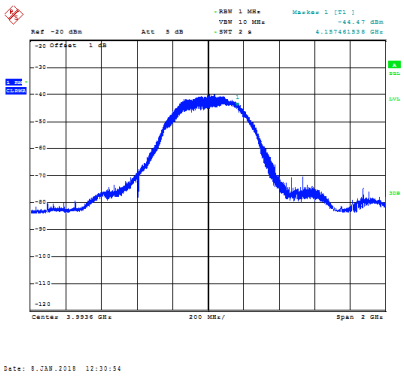


(a) Channel 1 at 16MHz PRF

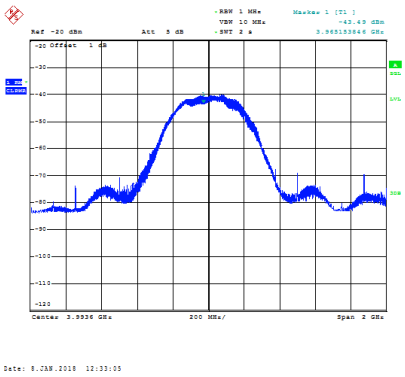


(b) Channel 1 at 64MHz PRF

Figure B.1: Transmit Power and Bandwidth on Channel 1 for 16 and 64MHz PRF, respectively.

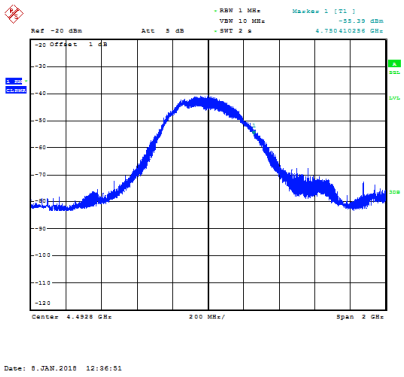


(a) Channel 2 at 16MHz PRF

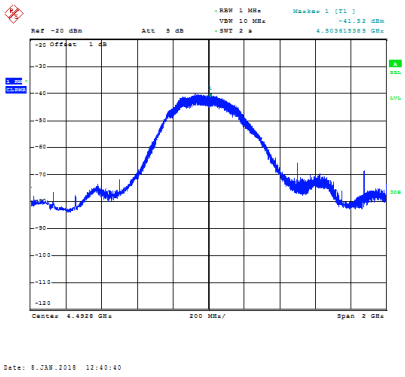


(b) Channel 2 at 64MHz PRF

Figure B.2: Transmit Power and Bandwidth on Channel 2 for 16 and 64MHz PRF, respectively.

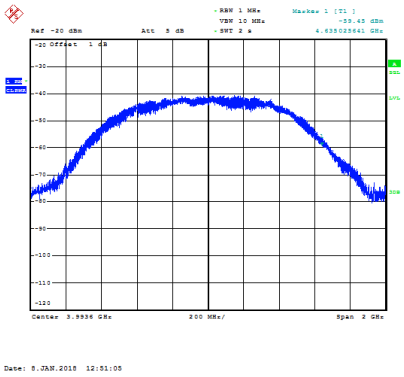


(a) Channel 3 at 16MHz PRF

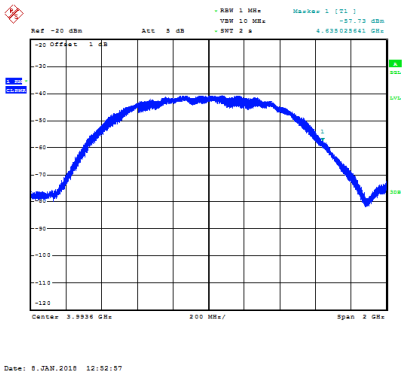


(b) Channel 3 at 64MHz PRF

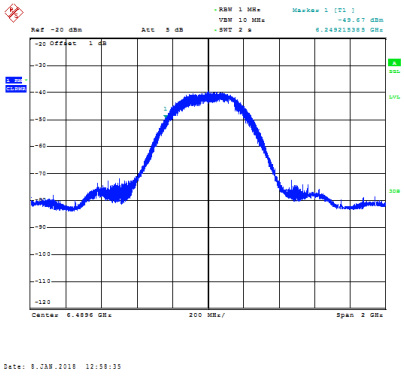
Figure B.3: Transmit Power and Bandwidth on Channel 3 for 16 and 64MHz PRF, respectively.



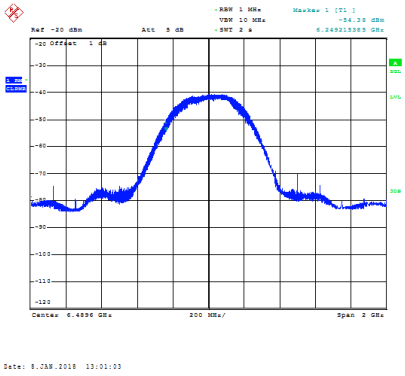
(a) Channel 4 at 16MHz PRF



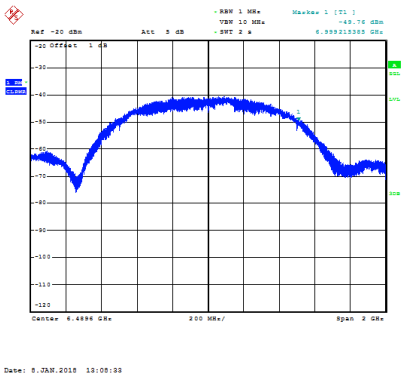
(b) Channel 4 at 64MHz PRF



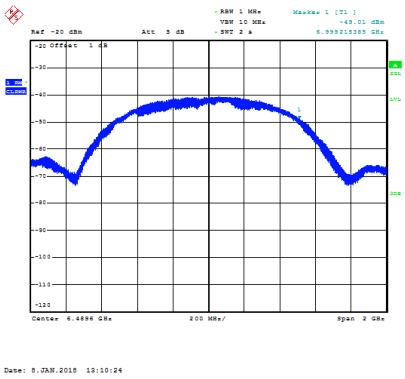
(c) Channel 5 at 16MHz PRF



(d) Channel 5 at 64MHz PRF



(e) Channel 7 at 16MHz PRF



(f) Channel 7 at 64MHz PRF

Figure B.4: Transmit Power and Bandwidth on Channels 4, 5, and 7 for 16 and 64MHz PRF, respectively.

Appendix C

Validation and Calibration Instructions

The source code and the firmware binaries for each test can be downloaded from [6]. To monitor the output of the firmware during each test, a serial terminal is needed. The communication setting is: *baudrate 115200 8N1*, no flow control. To ensure that the extension shield works correctly and within legal boundaries, the following tests must be performed.

C.1 Operational Tests

Operational tests do not use the RF section of the device under test (DUT). They test the DW1000s GPIO lines and SPI bus communication. During all operational tests the supply voltage and the supply current should be monitored. A voltmeter and an ampere meter are required for these tests.

C.1.1 Measure Idle Mode Current

This test measures the current consumption of the extension shield during the DW1000 IDLE state. After being connected to a power source, the DW1000 is in the WAKEUP state. The transceiver then initializes itself with the Decawave factory default settings (see Table 4.6) and waits until the crystal is stable and the RSTn is HIGH [26, Section 2.3]. After successful initialization, the DW1000 remains in IDLE state[1]. The jumper P1 on the extension shield was added in the design to measure its current consumption. It connects the 3V3 pin and the VCC plane.

Instructions

1. Flash the test program named `01_idlemode.bin` to the NUCLEO-L152RE board.
2. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.

[1] If current consumption is measured over time, it is possible to trigger on the rising edge of the RSTn as this firmware is performing a hardware-reset after power-up.

3. Remove all jumpers from the DUT (P1, P2, P4, P5 must be open).
4. Disconnect the NUCLEO-L152RE board from its USB power source.
5. Connect the ampere meter to the pins of P1.
6. Connect the voltmeter to the 3V3 and the GND pins of the DUT.
7. While monitoring voltage and current, power the board again by plugging in the USB cable.
8. Measure the current and the voltage a few seconds after the power source was connected.

Remarks. The current consumption depends on the supply voltage and all other components within the test circuit. The limits according to [21, Section 3.2] are min. 10mA and max. 25mA.

C.1.2 SPI test

This test can either be pass or fail. The device ID of every transceiver is programmed to address `0x000`. Reading this address must return the 4 byte device ID.

Instructions

1. Flash the test program named `02_readdevid.bin` to the NUCLEO-L152RE board.
2. Jumper P1 must be closed, jumper P4 can be closed or open.
3. Jumper P2 must be in position **CS1**.
4. Reset the board by pressing the button **B2** on the NUCLEO-L152RE board.
5. If the green LED **LD2** on the NUCLEO-L152RE board is on, reading the device ID was successful. If the green LED is flashing, the test failed.

After this test, disconnect the NUCLEO-L152RE board from its power source for a few seconds. This is necessary to reset the configuration of the STM32's GPIO6 pin on port B, which is the same GPIO used for SPI-CS.

C.1.3 Maximum Current Consumption

In this test, the maximum current consumption in the listening mode and with Decawave default settings (see Table 4.6) is measured. The current consumption should be in the range 10mA to 25 mA.

Instructions

1. Flash the test program named `03_max_current.bin` to the NUCLEO-L152RE board.
2. Jumper P4 must be open and jumper P2 must be in position CS1.
3. Optional: If a DWM1000 module is soldered on the board, then jumper P5 must be closed too.
4. Jumper P1 must be used to measure the current consumption. Connect the amperemeter to P1.
5. Connect the voltmeter to the 3V3 pin and GND pin.
6. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.
7. Measure the current consumption and the voltage.

C.1.4 GPIO strobe test

The GPIO lines that are connected in the schematic must be tested. Therefore, the firmware sends the SPI commands to set (or reset) each of the GPIO pins. The firmware then reads the state of the GPIO pin automatically and verifies if it works properly (the jumpers GPIOx must be closed). Additionally, the GPIO level can be measured with a voltage probe on the extension shield.

The state LOW is defined in [22] as max. 0.3*VDDIO and the state HIGH as min. 0.7*VDDIO.

Instructions

1. Flash the test program named `04_gpio.bin` to the NUCLEO-L152RE board.
2. Jumper P1 must be in position CS1, jumper P1 must be closed. If the LED_RXOK, LED_SFD, LED_RX, LED_TX are soldered on the board, jumper P4 must be closed. If a DWM1000 module is soldered on the board, jumper P5 must be closed too.
3. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.
4. Measure the voltage on every GPIO pin. It must be higher than 0.7*VDDIO.

C.1.5 RSTn test

After the board is powered, the line RSTn must go high. This happens when the DW1000 changes from its WAKEUP to INIT state. During the power-on phase, the DW1000 drives the RSTn line LOW. After 5 ms, the RSTn line must be HIGH.

Instructions

1. Flash the test program named `05_reset.bin` to the NUCLEO-L152RE board.
2. Jumper P1 must be closed and jumper P2 must be in position CS1.
3. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.

C.1.6 WAKEUP test

To save energy, the DW1000 supports a sleep mode. After a command was sent to the DW1000 via SPI, the device goes to sleep and must wake up again after driving the WAKEUP line HIGH.

1. Flash the test program named `06_wakeup.bin` to the NUCLEO-L152RE board.
2. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.
3. Jumper P1 must be closed and jumper P2 must be in position CS1.
4. Connect the ampere meter to the pins of P1.
5. Connect the voltmeter to the 3V3 and the GND pins of the DUT.
6. Push the button **B1** to trigger a WakeUp.
7. Measure the current and the voltage after pushing the button. If the DW1000 enters IDLE state successfully the current consumption should be the same as in C.1.1 (idle mode current).

Remark: The DW1000 enters the IDLE state after waking up again. Therefore, the current consumption must be the same as in C.1.1.

C.1.7 EXTON test

This test is equivalent to C.1.5 (RSTn test). Instead of the RSTn line, it is checked if the EXTON line goes HIGH after power is applied.

Instructions

1. Flash the test program named `07_exton.bin` to the NUCLEO-L152RE board.
2. Jumper P1 must be closed and jumper P2 must be in position CS1. If a DWM1000 module is soldered on the board, JP5 must be closed too.
3. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.

C.1.8 Transmitter Calibration

The second group of tests measure and then calibrate the DUTs transmitter. The extension shield has a SMA RF connector to use various antennas. During calibration it is recommended in [21] to use a coax cable, if possible. Before measuring the transmit power of the DUT, the path loss P_{LOSS} of the SMA cable must be determined for every tested frequency in Table C.2. This systematic measurement error must then be added to the transmit power P_{SA} measured by the spectrum analyzer. The real transmit power of the DUT P_{DUT} is defined by Eq.C.1. These measurements require a spectrum analyzer.

$$P_{DUT} = P_{SA} + P_{LOSS} \tag{C.1}$$

C.1.9 Crystal Trim

The DW1000 supports tuning the crystal oscillator by changing the value in register FS_XTAL `0x2B:0E` [26, Section 7.2.44.5]. The initial value of the trim register is `0x00` and the maximum value is `0x1F`. By pushing a button on the NUCLEO-L152RE board, the value of this register can be increased by 1. After reaching the maximum value the register will become `0x00` again. This needs to be done for one frequency only, because all carrier center frequencies are derived from the same oscillator frequency.

Instructions

1. Flash the test program named `08_xtaltrim.bin` to the NUCLEO-L152RE board.
2. Jumper P1 must be closed and jumper P2 must be in position CS1.
3. Connect the DUT to the spectrum analyzer using a coax cable.
4. Connect the Voltmeter to the pins 3V3 and GND of the extension shield.
5. Configure the spectrum analyzer center frequency at 6489.6 MHz, the channel 5 center frequency.
6. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board. The board will send a continuous wave (CW) signal on the channel 5 center frequency.
7. If the output signal is not equal to the channel 5 center frequency, increase the trim value by pushing button **B1** on the NUCLEO-L152RE board.
8. Count how often the button has been pushed and write down the count when the output signal was closest to the channel 5 center frequency. This value must be used to trim the crystal.

Resolution Bandwidth	1 MHz
Video Bandwidth	1 MHz
Span	2 GHz
Sweep time	2 seconds
Detector	rms
Average time per point	1 ms

Table C.1: Spectrum analyzer settings for measuring transmit power

C.1.10 Transmit Power and Bandwidth Calibration

It is important to calibrate the transmit power and bandwidth for each channel that will be used during operation later-on. Otherwise it is possible that the extension shield violates EU or FCC spectrum regulations. If the transmit power is too low the performance of the extension shield is not optimal. To find the highest transmit power gain that does not violate regulations [21, Appendix A], the user can set a coarse gain in 3dB steps

Channel	f_c	B	f_{PRF}	Code	G_C	G_F	PGDELAY
	[MHz]	[MHz]	[MHz]		dB	dB	
1	3494.4	499.2	16	1			
1	3494.4	499.2	64	9			
2	3993.6	499.2	16	3			
2	3993.6	499.2	64	9			
3	4492.8	499.2	16	5			
3	4492.8	499.2	64	9			
4	3993.6	1331.2	16	7			
4	3993.6	1331.2	64	9			
5	6489.6	499.2	16	3			
5	6489.6	499.2	64	9			
7	6489.6	1081.6	16	7			
7	6489.6	1081.6	64	17			

Table C.2: Measurements of transmit power and bandwidth.

and a fine gain in 0.5dB steps. Furthermore, the bandwidth limits (-51.3dB outside the channel boundaries) must not be exceeded while the bandwidth should be as high as possible. To change the channel bandwidth, the PGDELAY is adjusted. Increasing the PGDELAY value results in wider transmit pulses, hence decreasing the channel bandwidth. Calibrating the transmission gain and the PGDELAY is done via a serial terminal interface. The DW1000 supports two modes for choosing the transmit power. A manual transmit power mode, where the gain is set to a fixed value for all transmission bit rates and frame lengths and a smart transmit power mode. The SSmart Tx Powermode applies only for the highest bit rate of 6.8 Mbps and allows to boost the transmit power if the transmitted frame is shorter than one millisecond. This test procedure calibrates the manual transmit power mode only.

Instructions

1. Flash the test program named `09_txbandwidth.bin` to the NUCLEO-L152RE board.
2. Jumper P1 must be closed and jumper P2 must be in position CS1.
3. Connect the DUT to the spectrum analyzer using a coax cable.
4. Connect the DUT to a computer using a USB cable. Open a serial terminal on this computer to display the NUCLEO-L152RE boards serial output. The settings for the serial connection are `115200 8N1, no flow control`.
5. Set the Spectrum Analyzer settings according to Table C.1 recommended in [26, Section 8.2].
6. Reset the device by pushing the button **B2** on the NUCLEO-L152RE board.
7. The DUT is now transmitting continuously at the highest power setting. The power spectrum must be visible on the spectrum analyzer.

8. Follow the instructions in the serial terminal to configure the TXPOWER and PGDELAY registers.

9. Once the optimal combination of PGDELAY and TXPOWER (coarse gain G_C and fine gain G_F) is found, write down the values in Table C.2.

10. Repeat steps 1 to 8 for every line in Table C.2.

C.1.11 Antenna Delay Calibration

For this measurement a second DW1000 device is necessary. The antenna delay describes the time that the signal needs to travel from the TX/RX ports of the DW1000 until it reaches the antenna. Because the DW1000 is an impulse radio (IR), it can measure distances very accurately through time-of-flight (ToF) measurement. This is done by the two-way ranging (TWR) application which measures the distance between two DW1000 transceivers. The antenna delay adds a constant error to this measurement that must be compensated for accurate distance measurements. The propagation velocity of the transmitted signals depends on the medium. This must be taken into account when calculating the distance from the measured time-of-flight. This test requires a computer with a serial terminal installed to monitor the DUTs output a reference test board and a coaxial cable.

Instructions

1. Flash the test program named `11_twr.bin` to **two** NUCLEO-L152RE boards.

2. On both boards jumper P1 must be closed and jumper P2 must be in position CS1.

3. Connect both extension shields RF connectors with an coax cable.

4. At least one test board must be connected to a computer. Open a serial terminal on this computer to display the NUCLEO-L152RE boards serial output. The settings for the serial connection are `115200 8N1, no flow control`.

5. Reset both devices by pushing the button **B2** on the NUCLEO-L152RE boards. The serial terminal must show the output of the TWR application.

6. In the terminal the distance in meter is printed continuously. Take 30 measurements and calculate the mean D_{mean} and the standard deviation.

7. Measure the length of the coax cable. The antenna delay in meter can be calculated according to Eq.C.2.

Formulas: Formula for calculating T_{delay}

$$Delay_{RX,TX} = \frac{(D_{mean,measured} - D_{cable})}{propagation\ speed} \tag{C.2}$$

The antenna delay for transmitted frames is then given by Eq. C.3 and by Eq. C.4 for received frames. Receiving frames takes slightly longer than transmitting frames in the

DW1000. Therefore the combined delay is not apportioned equally, but 56% and 44% for the receiver delay and the transmitter delay respectively.

$$Delay_{RX} = Delay_{RX,TX} \cdot 0.56 \tag{C.3}$$

$$Delay_{TX} = Delay_{RX,TX} \cdot 0.44 \tag{C.4}$$

The programmed values of the delays must be a multiple of Decawave-time-units (dwtu). The are calculated by Eq. C.5.

$$Delay_{dwtu} = \frac{Delay_{seconds}}{1 \,/\, 499.2 \cdot 10^6 \,/\, 128} \tag{C.5}$$

Appendix D

Bill of Materials

Comment	Pattern	Quantity	Components
0.10uF	CAP 0402/1005	15	C11-C20, C23, C25-C27, C29
0732511150	MOLEX SD-73251-115	1	SMA
1.2pF	CAP 0402/1005	1	C6
100	RES 0402/1005	1	R7
10000pF	CAP 0402/1005	1	C22
100k	RES 0402/1005	1	R40
10pF	CAP 0402/1005	1	C30
11K 1%	RESC0603(1608)L	1	R3
12pF	CAP 0402/1005	2	C9, C10
16k	RES 0402/1005	1	R4
18pF	CAP 0402/1005	1	C8
270	RES 0402/1005	1	R5
27pF	CAPC0402(1005)60L	1	C5
330pF	CAP 0402/1005	2	C28, C31
38.4MHz	EPSON TSX-3225	1	Y1
4.7uF	CAPC0603(1608)100M	2	C1, C24
47uF	CAP 0805/2012	1	C21
8.5pF	CAPC0402(1005)60L	2	C3, C4
81-LXDC2HL18A-052	PCBComponent1	1	U2
820pF	CAP 0402/1005	1	C7
analog	HDR1X6	1	CN8
CurrentProbe	HDR1X2H	1	P1
digital	HDR1X10	1	CN5
digital	HDR1X8	1	CN9
DW1000 IC	QFN-48	1	U1
HHM1595A1	HHM1595A1	1	T1
LMK107BJ106MALTD	CAPC1608X100X35ML20T25	1	C2
POWER	HDR1X8	1	CN6

Table D.1: Bill of Materials of the UWB extension shield.

Bibliography

[1] Thingsquare AB. Contiki: The Open Source OS for the Internet of Things. https://www.contiki-os.org, 2018.

[2] Paul Alcock et al. Implementation and Evaluation of Combined Positioning and Communication. In *Proceedings of the 4th International Conference on Real-world Wireless Sensor Networks*, REALWSN'10, pages 126–137, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Arduino. Arduino Website. https://www.arduino.cc/, 2017.

[4] Autonomous Networks Research Group, University of California. Network Stack - Contiki. http://anrg.usc.edu/contiki/index.php/, 2018.

[5] E. Baccelli et al. RIOT OS: Towards an OS for the Internet of Things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, April 2013.

[6] Bernd Baumann. UWB extension shield production tests. https://github.com/BerndBaumann/, 2017.

[7] Azby Brown et al. Safecast: Successful citizen-science for radiation measurement and communication after Fukushima. *Journal of Radiological Protection*, 36(2):S82–S101, 2016.

[8] Michael Buettner et al. X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 307–320, New York, NY, USA, 2006. ACM.

[9] Nicolas Burri et al. Dozer: ultra-low power data gathering in sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 450–459. ACM, 2007.

[10] Pozyx BVBA. Pozyx Accurate Positioning. https://www.poyzx.io, 2015.

[11] M. Ceriotti et al. Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 187–198, April 2011.

[12] Paul Cheong and Oppermann Ian. An Energy-Efficient Positioning-Enabled MAC Protocol (PMAC) for UWB Sensor Networks. *IST Mobile and Wireless Communications Summit*, pages 95–107, 2005.

[13] Jaewoo Chung et al. Indoor location sensing using geo-magnetism. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 141–154. ACM, 2011.

[14] CIHOLAS. DWUSB. https://www.ciholas.com/.

[15] Pablo Corbalán et al. Poster: Enabling Contiki on Ultra-wideband Radios. In *Proc. of EWSN*, 2018.

[16] MAPSTED Corp. Mapsted. https://mapsted.com, 2018.

[17] Seiko Epson Corporation. *TSX-3225 Datasheet*. Seiko Epson Corporation, 2017.

[18] D. Lymberopoulos and J. Liu. The Microsoft Indoor Localization Competition: Experiences and Lessons Learned. *IEEE Signal Processing Magazine*, 34(5):125–140, Sept 2017.

[19] Decawave Ltd. *Application Note, Sources of Error in DW1000 Based Two-Way Ranging (TWR) Schemes, Version 1.0*. Decawave, 2014.

[20] Decawave Ltd. *APS014: DW1000 Antenna Delay Calibration Version 1.01*. Decawave, 2014.

[21] Decawave Ltd. *APS012: Production Tests for DW1000-Based Products, Version 1.3*. Decawave, 2015.

[22] Decawave Ltd. *DW1000 Datasheet Version 2.09*. Decawave, 2015.

[23] Decawave Ltd. *APS023: DW1000 Transmit Power Calibration and Management, Version 1.0*. Decawave, 2016.

[24] Decawave Ltd. *APS023 Part 2: DW1000 TX Bandwidth and Channel Power Compensation, Version 1.1*. Decawave, 2016.

[25] Decawave Ltd. *DW1000 Device Driver Application Programming Interface (API) Guide, Version 2.4*. Decawave, 2016.

[26] Decawave Ltd. *DW1000 User Manual Version 2.10*. Decawave, 2016.

[27] Decawave Ltd. *DWM1000 Datasheet Version 1.7*. Decawave, 2016.

[28] D. Dujovne et al. 6TiSCH: deterministic IP-enabled industrial internet (of things). *IEEE Communications Magazine*, 52(12):36–41, December 2014.

[29] Adam Dunkels. The contikimac radio duty cycling protocol. 2011.

[30] Adam Dunkels et al. Making TCP/IP viable for wireless sensor networks, 2003.

[31] Adam Dunkels et al. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32. ACM, 2007.

[32] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads. *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, page 29, 2006.

[33] Mathilde Durvy et al. Poster Abstract : Making Sensor Networks IPv6 Ready. pages 1–2, 2003.

[34] ENEVO. ENEVO - Better Waste Management. https://www.enevo.com/, 2018.

[35] Dave Evans. The Internet of Things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[36] Fitbit, Inc. Fitbit, Advanced Health and Fitness Tracker. https://www.fitbit.com/, 2018.

[37] Linux Foundation. Zephyr Project. https://www.zephyrproject.org/, 2018.

[38] Fruchterman, Thomas M. J. and Reingold, Edward M. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164.

[39] David Gay et al. The nesC Language: A Holistic Approach to Networked Embedded Systems. *SIGPLAN Not.*, 49(4):41–51, jul 2014.

[40] Omprakash Gnawali and Philip Levis. The minimum rank with hysteresis objective function. RFC 6550, RFC Editor, September 2012.

[41] Bernhard Großwindhager et al. Poster: Switchable Directional Antenna System for UWB-based Internet of Things Applications. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, pages 210–211. Junction Publishing, 2017.

[42] Isola Group. ISOLA-Group DE104 Datasheet. http://www.isola-group.com/products/all-printed-circuit-materials/de104/, 2017.

[43] Harbour Industries. MIL-DTL-17 Coaxial Cables. http://www.harbourind.com, 2018.

[44] Andrew C Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990.

[45] W. Hu et al. The design and evaluation of a hybrid sensor network for cane-toad monitoring. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 503–508, April 2005.

[46] IANA. IANA IPv4 Address Space Registry. https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml, 2018.

[47] IEEE. IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016.

[48] Inc. Saturn PCB Design. Saturn PCB Design Toolkit Version 7.04. http://www.saturnpcb.com/pcb_toolkit.htm, 2017.

[49] infsoft GmbH. Infsoft - Smart Connected Solutions. https://www.infsoft.com, 2018.

[50] National Instruments. TX-LINE: Transmission Line Calculator. http://www.awrcorp.com/products/additional-products/tx-line-transmission-line-calculator, 2017.

[51] Y. Jiang and V. C. M. Leung. An Asymmetric Double Sided Two-Way Ranging for Crystal Offset. In *2007 International Symposium on Signals, Systems and Electronics*, pages 525–528, July 2007.

[52] Heuel & Löher GmbH & Co. KG. Localino: Open Source Indoor Localization System. https://www.localino.net/, 2019.

[53] KiCAD. KiCAD PCB Calculator. http://kicad-pcb.org/, 2017.

[54] Lab11. Lab11 PolyPoint. https://lab11.eecs.umich.edu/projects/polypoint/, 2017.

[55] Philip Levis et al. Rfc6206: The trickle algorithm. RFC 6206, RFC Editor, March 2011.

[56] T. Lin et al. A Survey of Smart Parking Solutions. *IEEE Transactions on Intelligent Transportation Systems*, 18(12):3229–3253, Dec 2017.

[57] Altium LLC. Altium Website. http://www.altium.com, 2017.

[58] Decawave Ltd. *APH001: DW1000 Hardware Design Guide, Version 1.0*. Decawave, 2016.

[59] Decawave Ltd. *EVK1000 User Manual Version 1.12*. Decawave, 2016.

[60] Jan Lutz. luftdaten.info - Feinstaub selber messen - Open Data und Citizen Science aus Stuttgart, 2018.

[61] Rainer Mautz. Indoor positioning technologies. 2012.

[62] ST Microelectronics. *STM32L152 Datasheet DocId17659 Rev 12*. ST Microelectronics, 2016.

[63] Multi-CB. Multi-Circuit-Board. https://www.multi-circuit-boards.eu, 2017.

[64] Nest Labs. Nest Learning Thermostat, . https://nest.com/thermostats/nest-learning-thermostat/overview/, 2018.

[65] Jorge Nocedal et al. *Numerical Optimization 2nd*. Springer, 2006.

[66] Göran Nordahl. LPS Mini. https://hackaday.io/project/7183/instructions.

[67] OpenRTLS. UWB, WiFi case study for a large hospital. https://openrtls.com/page/cases.

[68] Matteo Ridolfi et al. Analysis of the Scalability of UWB Indoor Localization Solutions for High User Densities. *Sensors (Basel, Switzerland)*, 18(6), 2018.

[69] J. Rousselot et al. WideMac: A low power and routing friendly MAC protocol for ultra wideband sensor networks. *Proceeedings of The 2008 IEEE International Conference on Ultra-Wideband, ICUWB 2008*, 3:105–108, 2008.

[70] A. R. Jiménez Ruiz and F. Seco Granja. Comparing Ubisense, BeSpoon, and DecaWave UWB Location Systems: Indoor Performance Analysis. *IEEE Transactions on Instrumentation and Measurement*, 66(8):2106–2117, Aug 2017.

[71] Thomas Schmid. *Time in wireless embedded systems.* PhD thesis, University of California, Los Angeles, 2009.

[72] Zach. Shelby and Carsten. Bormann. *6LoWPAN : the wireless embedded internet.* 2009.

[73] s.r.o. Sewio Networks. Sewio. http://www.sewio.net, 2017.

[74] ST. *IDW04A1 Wi-Fi expansion board based on SPWF04SA module for STM32 Nucleo.* ST Microelectronics.

[75] ST. *IKS01A1 Motion MEMS and environmental sensor expansion board for STM32 Nucleo.* ST Microelectronics.

[76] ST. *UM1724 User manual. STM32 Nucleo-64 board. Rev 11.* ST Microelectronics, 9 2016.

[77] STMicroelectronics. *User Manual UM2000, Getting started with the Contiki OS/6LoWPAN on STM32 Nucleo with Spirit1 and sensors expansion boards.* STMicroelectronics, 2016.

[78] Texane. ST-Link Repository. https://github.com/texane/stlink, 2017.

[79] Texas Instruments. *CC256x Dual-Mode Bluetooth Controller, Datasheet.* Texas Instruments, 2016.

[80] TinyOS. TinyOS. http://webs.cs.berkeley.edu/tos/, 2018.

[81] Nicolas Tsiftes et al. Enabling large-scale storage in sensor networks with the coffee file system. In *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*, pages 349–360. IEEE, 2009.

[82] Nicolas Tsiftes et al. Low-power wireless IPv6 routing with ContikiRPL. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 406–407. ACM, 2010.

[83] UNISET. Sequitur. http://www.unisetcompany.com/, 2017.

[84] G. Werner-Allen et al. Monitoring volcanic eruptions with a wireless sensor network. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pages 108–120, Jan 2005.

[85] H. Will et al. The Geo-n localization algorithm. In *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10, Nov 2012.

[86] T. Winter et al. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, RFC Editor, March 2012.

[87] Wirola, Lauri and Syrjarinne, Jari. Crowd sourced vision and sensor-surveyed mapping, May 2 2017. US Patent 9,641,814.

[88] H. Wymeersch et al. Cooperative Localization in Wireless Networks. *Proceedings of the IEEE*, 97(2):427–450, Feb 2009.

[89] Z. Yang and Y. Liu. Quality of Trilateration: Confidence-Based Iterative Localization. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):631–640, May 2010.