Technische Universität Berlin
Faculty IV – Electrical Engineering and Computer Science

Bachelor's thesis

Batch localization algorithm for floating wireless sensor networks

Martin E. Haug

October 17, 2019
First Supervisor: Odej Kao
Second Supervisor: Felix-Valentin Lorenz

Martin Haug
Matr.-Nr. 380861
Major: B.Sc. Computer Science
m.haug@campus.tu-berlin.de

Abstract

Wireless Sensor Networks (WSNs) with mobile nodes require a mechanism to assign a position to each reading to fully leverage the gathered data. As implementing dedicated localization hardware in each node is often too expensive, WSNs demand other localization solutions that use the specific properties of the network. This thesis examines existing solutions and presents the novel Graph-based Localization (GRAL) algorithm. GRAL is a centralized, range-free algorithm that is designed to assign positions to readings of nodes in a mobile WSN deployed in an environment like a water grid that can be modeled by a graph and has some nodes with a known position. It operates by partitioning each nodes' reading packages in distinctive sets for which it can determine the start- and endpoints. The position for the in-between packages for each set is then linearly interpolated. In this thesis, I implement and evaluate GRAL with respect to several relevant scenarios.

Zusammenfassung

Wireless Sensor Networks (WSNs), die mobile Sensor-Knoten nutzen, benötigen einen Mechanismus, um jeder Messung eine Position zuzuweisen, damit die gesammelten Daten optimal genutzt werden können. Da die Nutzung von spezialisierter Lokalisierungshardware auf jedem der Sensor-Knoten oft zu teuer ist, benötigen WSNs andere Lokalisierungsmethoden, die die speziellen Eigenschaften des Netzwerks nutzen. Diese Abschlussarbeit untersucht bestehende Lösungen und präsentiert den neuen Graph-basierten Lokalisierungsalgorithmus (GRAL). GRAL ist ein zentralisierter, range-free Algorithmus. Er ist für mobile WSNs, die sich in Topologien wie einem Abwasser-Netzwerk, die mit einem Graph modelliert werden können, befinden und in denen die Positionen von manchen Sensoren vorher bekannt sind, entworfen. Er funktioniert, indem er die Messwertpakete jedes Sensor-Knotens in Reihen einsortiert, deren Start- und Endpunkt festgestellt werden können. Die Position der Pakete, die sich temporal zwischen diesen Punkten befinden, wird dann linear interpoliert. In dieser Abschlussarbeit implementiere ich GRAL und evaluiere den Algorithmus anschließend anhand einiger relevanter Szenarien.

Keywords: localization, WSN, range-free, positioning, mobile nodes

Contents

1 Introduction **1**

1.1 Water network monitoring 1

1.2 Problem setup and contributions 2

2 Background **3**

2.1 Fundamentals of WSNs 3

2.2 Localization algorithms in WSNs 4

2.2.1 Range-based algorithms 4

2.2.2 Range-free algorithms 6

3 Method **9**

3.1 Algorithm 9

3.2 System architecture 18

4 Results **22**

5 Conclusions **27**

Bibliography **28**

List of Figures

2.1 The principle of *Time Difference of Arrival* 5

3.1 Exemplary environment graph with junctions and relays as vertices 9

3.2 Epochs created for a single node moving from a relay to another 10

3.3 Checkpoint creation on a single edge 15

4.1 Distribution of iRSME for a dataset with one sensor node and no junctions 22

4.2 Instances with low or high iRSME from the first dataset 23

4.3 Distribution of iRSME for a dataset with two sensor nodes and no junctions 23

4.4 Instances with low or high iRSME from the second dataset 24

4.5 Distribution of iRSME for a dataset with two sensor nodes and a junction without a relay in the middle 24

4.6 Comparison between enabled and disabled path rectification 25

4.7 Distribution of iRSME for a dataset with two sensor nodes and a junction with a relay 26

1 Introduction

Increased availability of cheap micro-controllers, matching development boards, and small single-board computers drive the adoption of Wireless Sensor Networks (WSNs). Typical usage of WSNs is to continually survey characteristics across a spread out area which makes them useful for a variety of fields like disaster prevention, environmental monitoring, and ambient computing (Reinhardt, Zöller, and Christin 2014), to name a few.

Readings have to be assigned a location within the deployment area to get the maximum insight from the measurements of the network's nodes. Using satellite localization systems like GPS or Galileo is in many applications not feasible for all sensors of the network since the units are designed to be cheap to produce for massive deployment and to use little power. Additionally, the strength of satellite signals can be too weak in some environments. In such cases, one can instead use proximity information to other nodes and properties of the environment to assign a position to each reading.

In some situations, possible locations of a sensor node are constrained by its environment, i.e., for WSNs situated in grid-like structures. An algorithm can make use of this by reasoning about the environment and the possible paths that the mobile nodes may take throughout to decrease the number of nodes with a known position required to achieve sufficient accuracy.

In this thesis, I investigate a way to effectively determine the sensors' location within constrained environments whose topologies can be mapped onto a tree graph. Some reference nodes are present throughout the environment. The algorithm is best suited for deployments with frequently changing node positions as the nodes traverse the deployment area. The presented system performs the localization on a centralized backend server once readings are transmitted. As more data is received, it emits batches of localized measurement packages. Targeted environments feature links such as roads or pipes that have a certain length and junctions. The position the system outputs for each measurement consists of the two junctions connected by the link the node is on and the distance it is removed from the start junction on the link.

I first present the project in which this research has been embedded, then review related work and give a quick overview of the functionality and characteristics of WSNs. After that, I introduce and discuss a novel algorithm for sensor localization. A section containing results obtained in a simulated environment follows. Finally, the thesis offers a conclusion and possible avenues for future research into this localization approach.

1.1 Water network monitoring

A possible use case for localization system presented in this thesis is the WaterGridSense 4.0 project. WaterGridSense 4.0 (WGS) is a joint research & development project of the Complex IT Infrastructure group as well as multiple german research institutions and small businesses. The project aims to provide monitoring and analytics for urban sewage networks by using a WSN with two classes of nodes. The first class of nodes consists of devices installed in the drains connected to the sewage system. These nodes are mounted in the drain pipes and thus remain static. They have a steady power supply and continuously transmit their data to a centralized backend that runs the analytics platform.

The second type of sensor is a floating, mobile sensor node that can be inserted into the sewage

system at any point to obtain measurements from pipes between two vertical tunnels. These devices run on battery and roam the network. They cannot directly contact the data center because of their limited wireless range. Instead, they use deliberately placed gateways that are deployed during the measurement campaign as a relay to transmit their data every time they make contact with one. The floating sensor nodes consist of a small micro-controller equipped with a short-range wireless transceiver and several measurement devices.

The structure of the targeted sewage systems in Berlin and Hamburg equals a tree, i.e., there are only pipes that branch in with more wastewater. The network eventually merges into a big trunk, in which there are no loops.

1.2 Problem setup and contributions

The problem adressed in this thesis is the localization of readings mobile nodes created at a specific time. The WSN has to be deployed in an environment consisting of junctions that optionally feature stationary nodes with known locations and links of a known length connecting the junctions. The graph representing the structure of said deployment, as well as the approximate stationary nodes' wireless radius, has to be known beforehand. There has to be a sink node to which all readings are transmitted.

The central contribution of this thesis is the Graph-based Localization algorithm that solves this problem on the sink node for the incoming packages. The algorithm outputs localized reading packages in batches as it retains the input data until a route that a node took in the environment can be determined. The algorithm is customizable; this thesis develops and evaluates two optional features for it. Furthermore, I propose an infrastructure for a WSN that enables it to transmit its data to a sink in a timely, efficient, and scalable manner.

2 Background

In this section, I introduce the terminology used throughout this thesis, give an overview of the basics of Wireless Sensor Networks and then present prior research into localization techniques.

I use the term position to refer to a point in the physical plane where the sensor node is located whereas location is a more general term. A reference node denotes a node of the sensor network for which the position is known; the term relay is used interchangeably since a sensor is a reference node iff it is a relay in our application. An unknown node is a sensor node with an unknown location. Note that in the context of the algorithm presented in chapter 3.1 a position consists of a start and destination vertex representing junctions or relays of the deployment area in a graph and a distance that describes how far from the start vertex the node is currently removed. In the context of this thesis, a position defined in this way is also represented as a 4-tuple.

2.1 Fundamentals of WSNs

Wireless Sensor Networks have garnered a lot of research interest in the last decade. To start this section, I will first define the term:

Definition 1 *A Wireless Sensor Network is "a network of nodes that cooperatively sense and may control the environment enabling interaction between persons or computers and the surrounding environment." (Buratti et al. 2009, 6870)*

Tubaishat and Madria 2003 name another landmark property of WSNs: The energy budget of a node is typically constrained to minimize the cost of a node and maximize its lifespan. The WSN transmits its data to a central 'sink node' (Tubaishat and Madria 2003). Hierarchical networks like the one described in section 1.1 often route the communication between the lowest-level sensing nodes and the sink through a set of higher-level nodes. This behavior introduces another characteristic of WSNs: they are frequently heterogeneous, which means that the sensors may have different capabilities (Krishnamachari 2005). The overview Tubaishat and Madria 2003 names some more characteristics of WSNs: The nodes are usually inexpensive ($< 1\$$) and widely deployed. These properties and the influence of its not necessarily friendly environment makes the devices error-prone. Krishnamachari 2005 further notes that the memory size of each node is modest and computation may be slow and expensive.

Buratti et al. 2009 explain that research into WSNs started to evolve from work on ad-hoc networking in 2001. The critical difference between the two fields is that ad-hoc networking occurs between full-featured more sparsely set computer nodes, whereas WSNs are usually densely populated networks of battery-powered nodes with constrained computing capabilities. Just like in ad-hoc networks, routing is a significant problem in WSNs (Tubaishat and Madria 2003) since the network structure is unknown at the start and is subject to change in WSNs with mobile nodes (mobile WSNs). The choice of routing algorithm may also be informed by the required computational complexity to save the nodes' energy.

The nodes of a wireless network may not have unique identifiers at the start of the algorithm so that an algorithm like 'Self-organized ID Assignment' described in Lin, Liu, and Ni 2007 may be needed. The assigned IDs enable query of individual sensors based on their previous measurements,

position assignment, and can support routing. Such systems may be useful in conjunction with the algorithm presented in this thesis because it requires each node to have a unique identifier.

Researchers are also interested in how to detect and recover faults in a WSN. Souza, Vogt, and Beigl 2007 provide a survey of the self-monitoring and recovery techniques that nodes can use to increase the availability of the network as well as replication approaches to provide backup for failed nodes. Another form of error is a measurement error that can create outliers in the collected dataset. The paper Zhang, Meratnia, and Havinga 2010 gives an overview of the techniques to detect and eliminate measurements that are inconsistent with the dataset. The discussed approaches are often known from the data mining field of machine learning but require adaptation to the distributed data collection in a WSN.

To maintain the order of incoming measurements, but also to execute some algorithms, including Time of Arrival-reliant localization, described in section 2.2.2, the clocks of the sensor nodes have to be synchronized. Sadler and Swami 2006 summarize the methods a WSN can use to achieve this. They determine that classical synchronization algorithms like NTP are not appropriate for WSNs due to the power requirements of passive listening. The discussed approaches include synchronization with the sink node, pairwise synchronization, and synchronization of subsets of nodes.

Rault, Bouabdallah, and Challal 2014 review the research in making a WSN more energy-efficient by optimizing its radio transmissions and software. The techniques include aggregating data to save energy by transmitting less to the sink, adaptive sampling rates which vary depending on the rate of change in the observed environment and duty cycling schemes which enable the sensors to sleep more often.

2.2 Localization algorithms in WSNs

Location algorithms are relevant in WSNs to assign positions to the recorded measurements. The position has to be algorithmically determined because deploying a large number of devices rapidly often means that the individual nodes' final positions cannot be pre-planned. Because of different deployment topographies, project requirements, and hardware constraints, different WSNs have different requirements for a localization algorithm. Therefore, various approaches already exist.

There are generally two ways to determine sensor location: *range-based algorithms* which use data from the radio communication device of the sensor about the connectivity to its proxies (Karl and Willig 2007) and *range-free algorithms* which do not use any other information from the radio than binary connectivity (Mao, Fidan, and Anderson 2007). Most algorithms presented here, including the new approach in section 3.1 use anchors and are thus considered to be anchor-based algorithms. The last two algorithms in section 2.2.2, however, represent the class of anchor-free algorithms that do not require any nodes with known positions. Additionally, there is a difference between methods that can be used on every node independently (i.e., distributed) and those that require to be run centralized, with all sensor data present.

2.2.1 Range-based algorithms

In primitive range-based algorithms, several reference nodes establish contact to a sensor. They can then perform triangulation to locate the sensor using either distance or angular information. Distance-based localization (usually called *lateration*) means that at least three reference nodes have to take a distance measurement to the sensor node. The distance reading of each reference node now becomes the radius of the circle of possible locations of the sensor. The point at which the respective

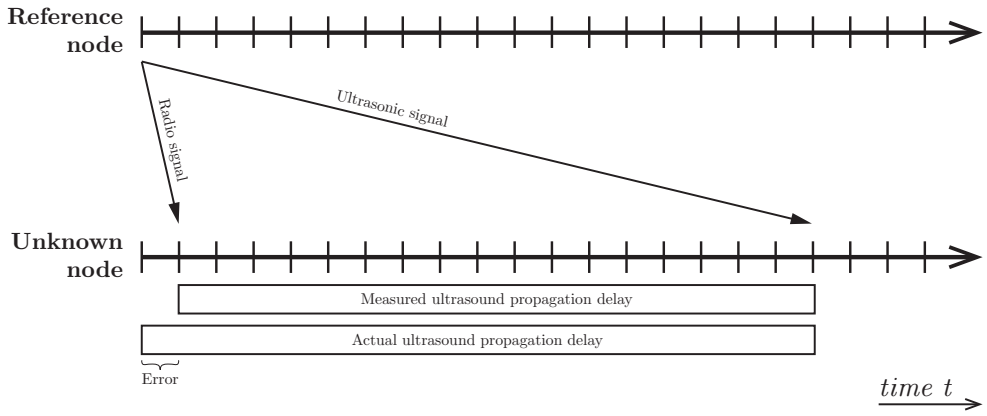


Figure 2.1: The principle of *Time Difference of Arrival* based on a graphic in Krishnamachari 2005, 40

circles of all three reference nodes intersect is the position of the sensor (Hightower and Borriello 2001, 2-3).

The implementation challenges of this algorithm are the required reference node density and implementing a method to measure the distance between the reference nodes and the unknown node. Just using the *radio signal strength indicator (RSSI)* is possible, since there exists a model for radio wave propagation (Rappaport 2002):

$$P_{r,dB}(d) = P_{r,dB}(d_0) - \eta 10 \log(\frac{d}{d_0}) \tag{2.1}$$

P denotes the signal strength of a sender at distance d, with a known value for d_0. η represents the path loss exponent that varies for different environments. In the description of the model by Krishnamachari 2005 a variable $X_{\sigma,dB}$ denoting a "log-normal random variable with variance σ^2 that accounts for fading effects" is added to the function since a random process can effectively model this effect.

The findings of Patwari, O'Dea, and Yanwei Wang 2001, however, demonstrate that RSSI distance measurements do only provide around 1m-accurate location. Karl and Willig 2007 highlight several other issues with RSSI distance measurements such as non-sufficient calibration of cheap radio devices and the presence of obstacles.

Another possibility to get the distance between two nodes is *time of fight*: Here, the relay emits a signal to the unknown node at a time known to both nodes. The unknown node can then calculate the distance between it and the reference node and return it if necessary by assuming constant signal propagation speed. Alternatively, it could return the package to the relay to eliminate the need for clock synchronization. The relay then assumes symmetric propagation times. Lacking clock resolution is the main reason why this method is not commonly applied for radio signals. Their propagation speed is so high that minor differences between the clocks of sender and receiver already cause significant errors (Hightower and Borriello 2001).

The third common way to solve this problem is using the *Time difference of arrival* discussed in Krishnamachari 2005, 39-40: The nodes now have an ultrasonic transmitter in addition to their radio device. The relay emits a pair of radio signal and ultrasonic pulse to an unknown node first. The node then measures the time between the arrival of the radio and the ultrasonic signal and calculates its distance by applying that time interval as the time of flight for the ultrasonic pulse. It dismisses the time that the radio signal transmission took because it is orders of magnitude smaller than the sound transmission speeds. Figure 2.1 illustrates this method. Ward, Jones, and Hopper 1997 use

this technique to enhance the Active Bade system outlined in section 2.2.2 with ultrasonic-enabled portable transceivers.

Usually, environments that use angulation feature a high density of reference nodes. With these redundant measurements, the problem can be expressed as a least-square optimization to reduce inaccuracies derived from measurement errors as described by Dag and Arsan 2018.

In fact, not only the unknown nodes can use lateration to obtain their position from the reference nodes, the reference nodes themselves are using the technique when connecting to a Global Navigation Satellite System like GPS. Each of the satellites regularly broadcasts its current clock time. The time of flight is used to perform lateration, but due to the high clock accuracy needed, a fourth satellite is required to resolve for the unknown time (Dawoud 2012, 1).

Using *angulation* instead of *lateration* is helpful to reduce active participation of the unknown nodes. As described by Hightower and Borriello 2001, 5-6, this procedure uses angles from two reference nodes to the unknown node to localize it using trigonometry by considering the known distance of the reference nodes. The reference nodes may use a rotating, directional antenna searching for sensor node contacts to get these angles. The reference node can then assign an antenna angle to this contact. The main drawback is the necessity of the dedicated angulation hardware.

There exists a range-based statistical technique in Parker and Langendoen 2004 for localization in WSNs with mobile reference nodes. The approach works by using rectangular bounding boxes for the locations of the sensor. At the start, the bounding boxes of each sensor span the whole deployment area. If a sensor node has acquired contact to some reference nodes, it then proceeds to intersect the boxes in which their signal can be picked up with their bounding boxes. Once the positions of at least two reference nodes have been acquired, it assigns a probability to the positions within the new bounding box. This probability is a constant multiplied with a normal distribution with its mean set at the RSSI-estimated distance from the reference node. The algorithm returns the location with the highest probability. This probability now becomes the certainty of the node. If that certainty exceeds a threshold, the node starts acting like a reference node itself.

2.2.2 Range-free algorithms

Range-free algorithms eliminate the need for extra hardware. One of the most straightforward techniques particularly suited for dense networks is the *Centroid algorithm* as discussed in Singh and Sharma 2015, 12: The position of an unknown sensor is calculated to be the centroid of the position of all visible reference nodes. That means that if the number of visible reference nodes is one, the algorithm sets the unknown node's position equal to the sole reference location.

Another simple range-free approach is the Active Badge system pioneered by Want et al. 1992. They use several batches that contain a transmitter which periodically emits a beacon signal. Each room in the indoor deployment area features a ceiling-mounted receiver which assigns the nodes whose beacon it received the current room as a location.

A more advanced technique described by Krishnamachari 2005, 35-37 is employing *geometric constraints*. Here, the shape of the region from which a reference nodes' signal is visible is used to refine the location data. For each unknown sensor, the system intersects the shapes of the regions where the beacons of received nodes are visible. The resulting set of positions is the location of the unknown node.

Tthe *Approximate point in triangle (APIT)* algorithm of He et al. 2003 pursues a similar approach. They partition the regions of the deployment area by spanning triangles of three reference node positions each over it. The algorithm then works by intersecting all the triangles which contain

a sensor node. The difficulty lies in detecting in which of these triangles a sensor node is. The paper proposes an algorithm to determine this for densely populated networks: "If no neighbor of [the unknown node] M is further from/closer to all three anchors A, B and C simultaneously, M assumes that it is inside triangle $\triangle ABC$. Otherwise, M assumes it resides outside this triangle." (He et al. 2003, 84).

Ray et al. 2004 propose the *ID-CODE* algorithm that marries proximity data with graph theory techniques. To execute their algorithm, chose a set of arbitrary positions for the whole network. Then create a connectivity graph $G = (V, E)$ using these positions as vertices, such that there exists an edge between two vertices iff nodes placed at each of these vertices could receive each other's signal. Then, choose a subset C of vertices such that the relation 2.2 holds. $N(v)$ returns the vertices that v has an edge to and is also called that vertex' ball.

$$\forall v \in V. \forall g \in V. v \neq g \leftrightarrow N(v) \cap C \neq N(g) \cap C \tag{2.2}$$

C is then called an identifying set and $I(v) = N(v) \cap C$ is the identifying code of v. Since every node has a unique identifying code, a simple lookup table for wireless contacts mapped to the position regions can be used on each sensor node to determine its position swiftly. The paper proposes an algorithm to identify an irreducible identifying set. The approach can only be used if the graph is distinguishable. Ray et al. 2004, 1019 assert that "one must merely check that there are no two vertices with the same ball" to determine if that property holds.

Maritime navigators developed and used the *Dead reckoning* technique. It uses a reference location and information about velocity and heading to estimate the current position. Kuang, Niu, and Chen 2018 appropriate the method for Inertial measurement units found in mobile phones and other devices. Given an initial position, velocity, and orientation, they can locate the device with a maximum error of 4m on an 80m parcours. Rashid and Turuk 2015 apply dead reckoning to a WSN in which all nodes are mobile. Their solution employs checkpoints that occur periodically for the whole system, thus necessitating clock synchronization. During these checkpoints, every sensor that has determined its location broadcasts it. Uninitialized sensors perform trilateration with that information to obtain a starting position. They then use their velocity data to do lateration at the next checkpoint using only two reference positions. The unknown node picks between the two resulting possible locations by minimizing a correctness term that depends on its previous velocity. This use of trilateration introduces a range-based technique in an otherwise range-free dead reckoning localization method. Both described techniques share a drawback with range-based techniques: They need additional hardware to gather the required information about their movement.

Scene analysis is another type of range-free localization solutions, that "use features of a scene observed from a particular vantage point to draw conclusions about the location of the observer or of objects in the scene" (Hightower and Borriello 2001, 6). Some algorithms use a camera to record images, simplify them, and match them with a database of references. Usage of deep learning methods like in the paper of Arandjelovic et al. 2016 can turn the camera into a tool to perform symbolical location (e.g., which room) lookup for many environments. Another scene analysis approach can be to use the RSSI data to create a database of what characteristics a signal from a particular location has to the relay nodes, as Bahl and Padmanabhan 2000 did. They use an offline measurement phase during which someone brings a mobile node to several positions. The relays receive its signal-to-noise ratio, signal strength, and coordinate and directionality annotations. The unknown nodes in the online system are localized by fetching some of the best matches to the received signal characteristics from the database. These matches' positions are then averaged to obtain the node position. The method achieves 2-3m accuracy.

The structure of densely populated WSNs is leveraged by Niculescu and Nath 2001 in their *DV-*

Hop scheme. Every reference node first propagates its position to all its neighbors. The receivers retransmit this position to their neighbors with the hop-count of that package incremented by one, thus flooding the network with information about the reference nodes' positions. Once a reference node has learned about the position to another such node and the number of hops between them, it can issue a correction term that enables the unknown nodes to calculate an average hop distance and therefore estimate their distance from each reference node. They can perform trilateration after they obtained data for three reference nodes. If there is data about more than three reference nodes, the nodes discard the packages with a higher hop-count. The distance to the neighboring nodes can also be estimated using a range-based RSSI measurement. Another approach outlined in the paper is using the distance to and between two neighbors, distance to the reference node, and the two neighbors' estimates for that distance. With these five distances of the quadrilateral known, a node can calculate the sixth (between the unknown node and the reference node).

In Robotics, *Monte Carlo methods* (statistically driven algorithms for which a non-zero chance for a wrong result) for localization exist. Hu and Evans 2004 adapt this approach for WSNs with mobile nodes. The method starts with a random sample of locations for each node. Then, in each step, there is a prediction and a filtering stage. During the prediction stage, a new set of locations for that node is built. For each previous position, a new position is drawn at random from the disk centering on the previous location with the maximum speed as its radius. The filtering step discards the positions that are impossible given that steps observations. The algorithm accomplishes this by reasoning about the positions of the reference nodes. If they have started receiving a reference node, they can safely discard positions removed any farther from the reference node's position than its maximum wireless range r radius. Similarly, if they do not receive a reference node, but their neighbors do, they assume that their distance is in the interval $]r; 2r[$. The position estimate at each step is the averaged position from the set.

Baggio and Langendoen 2008 build on this paper with their *Monte Carlo location boxed (MCB)* scheme. They use rectangular boxes instead of circles to approximate the radio range of each node. Before the prediction phase of each step, they first build the anchor box, which is the box a sensor node has to be in to hear all the reference nodes that it has heard. They then for each old position constrain that box further by requiring that all positions within it are reachable from that previous position. Then the prediction is performed, and the subsequent filtering step eliminates all positions where an uncontacted reference node should be visible.

If there are no reference nodes at all in the WSN, the nodes can solve a problem of graph embedding to determine their position in a virtual coordinate system that approximates the relative locations to each other that the nodes all have using the technique of Rao et al. 2003. A node first sends an initial message that the receivers flood through the whole network. The other nodes try to determine if they are at the physical perimeter (if they are perimeter nodes) of the network by determining if they have the largest hop distance to the initial origin of the message amongst their two-hop neighbors. The perimeter nodes then announce themselves to the network and save the hop distance to each other perimeter nodes. They then use triangulation to determine all perimeter node positions. Finally, the other nodes set their position to the average of the positions of their neighbors repeatedly, until the system converges.

Another approach in this scenario is to use *Anchor Free Localization* proposed by Priyantha et al. 2003. They try to find a graph embedding of the connectivity graph of the WSN that most resembles the graph embedding that uses the sensor nodes actual positions. To do that, they try to find a fold-free embedding first and then do a mass-spring optimization (the edges are simulated to act like springs between their two attached vertices). The first step seeks to avoid local minima in the optimization that often occur when components of the graph are 'folded' over each other.

3 Method

This section describes the main contribution of my thesis. I propose a system that assigns position estimates to node reading packages in a WSN with mobile nodes. The system leverages the mobility of nodes, their proximity relations, and the structure of the environment they traverse. Processing of reading packages is centralized and occurs as the network transmitted them to the sink. The foundational algorithm for the system described in section 3.1 is called *Graph-based Localization* (GRAL).

GRAL requires that the WSN's environment has a known structure of links and junctions. With the junctions as vertices and the links between them as edges in a weighted graph (environment graph), said graph has to be a tree. Figure 3.1 shows an example of an environment graph. The weight of each edge has to correspond to the physical length of the corresponding link. The identifiers should especially allow discriminating between mobile nodes and relays based on it alone. All nodes of the network shall be uniquely identifiable and broadcast their identifiers to their neighbors. The unknown nodes shall traverse the deployment area. They directly or indirectly transmit their readings to a single sink node. These transmissions need not occur immediately after the sensing node saved its current measurements but when convenient. The readings, bundled with a timestamp and a list of other nodes contacted (featuring identifier of the node and its RSSI) shall be called a package. The timestamps of the transmitted packages have to be non-decreasing. Packages do not have to have the same temporal distance to one another. The system shall include reference nodes that remain stationary. The environment graph models location of a reference node by including a vertex that splits the link edge the node is on in two if it is not already on a junction. An estimation for the wireless coverage radius of each relay has to be known.

3.1 Algorithm

The basic idea behind GRAL is to put the packages that a mobile node emits into epochs that share distinct characteristics. For each epoch, the average speed is calculated based on the first and last packages' timestamps and the traveled distance derived from the environment graph's edge weights. GRAL returns a list of localized packages as soon as it has enough information to determine the start and destination vertices of a movement as well as a final position. The returned position

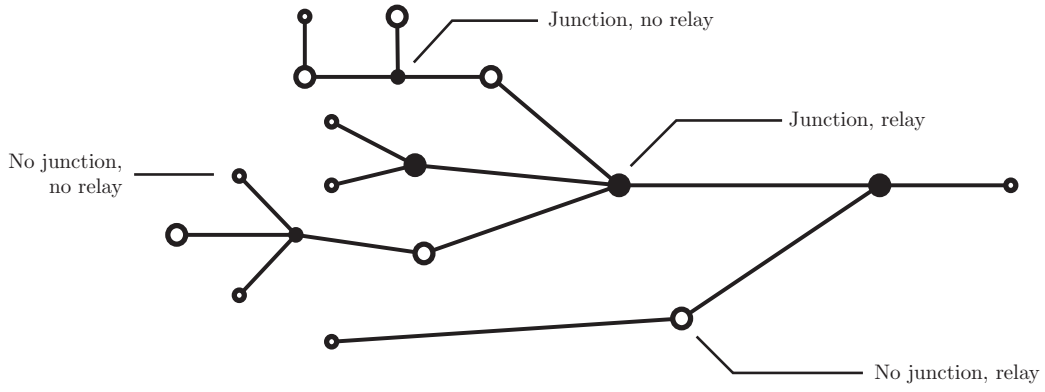


Figure 3.1: Exemplary environment graph with junctions and relays as vertices

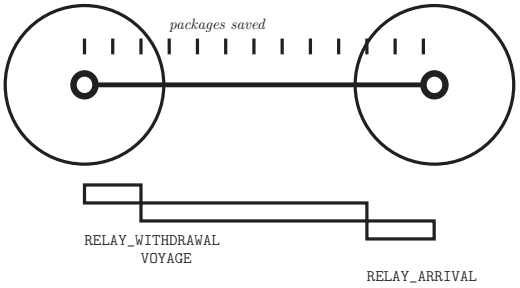


Figure 3.2: Epochs created for a single node moving from a relay to another

structures containing a start and end vertex and the distance the sensor is removed from the start can be mapped onto the real, geographical positions easily by an external system if necessary. The saved packages are evicted from memory once the system has determined their positions; only a final position gets saved.

The epochs are assigned one of these types: `RELAY_APPROACH` (the node approaches a relay; invariant 3.2), `RELAY_WITHDRAWAL` (the node withdraws from a relay; invariant 3.3) and `VOYAGE` (the node has no relay contacts; invariant 3.4). In the simplest scenario of a single mobile node traversing the environment, the invariants dictate the number and sequence of epochs, as illustrated in figure 3.2. In the following propositions, I denotes the set of package indices of the respective epoch, and p_i is the package at index i. R is the set of relays in the system, c is a function that returns a set of contacted nodes for each package, RSSI_p is a function that returns the RSSI for a node in the package p and identifier returns the identifier of a node.

$$r_i = \max_{\mathrm{RSSI}} \left(\mathrm{c}(p_i) \cap R \right) \tag{3.1}$$

The variable r_i helps formulating the propositions 3.2 and 3.3 by containing the relay with the strongest RSSI from epoch i.

$$\begin{aligned} \forall i \in I.\, i = 0 \vee \Big[& \mathrm{RSSI}_{p_i}(r_i) > \mathrm{RSSI}_{p_{i-1}}(r_{i-1}) \wedge \\ & \mathrm{identifier}(r_i) = \mathrm{identifier}(r_{i-1}) \Big] \end{aligned} \tag{3.2}$$

For all considered packages, the RSSI regarding the connection to the strongest relay node is greater than in the preceeding package (if that exists). The strongest relay node remains constant across all packages.

$$\begin{aligned} \forall i \in I.\, i = 0 \vee \Big[& \mathrm{RSSI}_{p_i}(r_i) \leq \mathrm{RSSI}_{p_{i-1}}(r_{i-1}) \wedge \\ & \mathrm{identifier}(r_i) = \mathrm{identifier}(r_{i-1}) \Big] \end{aligned} \tag{3.3}$$

For all considered packages, the RSSI regarding the connection to the strongest relay node is lower or equal than in the preceeding package (if that exists). The strongest relay node remains constant across all packages.

$$\forall i \in I.\, \mathrm{c}(p_i) \cap R = \emptyset \tag{3.4}$$

None of the considered packages contain records of wireless contacts to a relay.

Consequently, incoming packages are added to the latest epoch if that keeps the invariant valid or assigned to a new epoch if the constraints changed. There is a case distinction for assigning an epoch to a package that arrives if there are no previous packages: If there is a contact to a relay a new epoch of the type `RELAY_WITHDRAWAL` is created for the package. We choose this type because reaching the position of a relay triggers the processing and eviction of previous packages. Since the sensor has been at the relay's position, it can now only withdraw.
Enabling the checkpoint option enriches the data by using information about encounters with other unknown nodes. When processing an epoch's packages, a checkpoint is added to encountered nodes if they did not yet send packages of or after that moment themselves. There will only be one checkpoint added to another node per epoch – the algorithm selects the wireless contact with the highest RSSI. The checkpoint is a position with a timestamp and the identifier of the issuing node attached. When the encountered node reaches a relay and starts to transmit its packages, GRAL creates a new epoch for each package with a later timestamp than that of the checkpoint. The ending position of the previous epoch will is now the checkpoint. The checkpoint deletes itself from the node after usage. Figure 3.3 illustrates the process.
This mechanism principally aids with nodes that overtake each other. This maneuver may seem unlikely in scenarios like water monitoring described in section 1.1 since one could intuitively assume that flow speed across a single pipe is approximately constant at any moment in time. The Hagen-Poiseuille equation, however, actually suggests that in every single pipe, a liquid is fastest in its center while the velocity approaches zero at the walls of the pipe due to viscosity (Haug 2006).
Another important mechanism that can be triggered by encountering another node is *path rectification*. The basic idea is to place lower bounds on the positions where nodes encounter one another by using information of where they have been before and where the earliest shared vertex on their route may be. If a node a arrives at a relay r_f with information about encountering another node b, the algorithm checks at which relay r_b node b has been last. If the path from r_b to r_f did not contain all edges that the path from a's original relay r_a to r_f has (refer to proposition 3.5 where path returns a list of edges between two vertices), the earliest shared vertex v_c (*confluence vertex;* see proposition 3.7) is calculated. If the position estimation process yields a position for the encounter which is before v_c on a's path to r_f like described in proposition 3.8, the algorithm splits the respective epoch of a into two with the encounter as the last package of the first epoch with its ending position set to v_c. The function s returns the start vertex of an edge, and the function encounterEdge returns the environment graph edge on which two nodes have encountered each other last, according to naïve linear interpolation.

$$\neg(\text{path}(r_a, r_f) \subseteq \text{path}(r_b, r_f)) \tag{3.5}$$

The path from r_b to r_f does not contain the path from r_a to r_f.

$$\text{pathStartNodes}(v_a, v_b) = \{\text{s}(f) \mid f \in \text{path}(v_a, v_b)\} \tag{3.6}$$

This helper function for proposition 3.7 returns the nodes on the path from v_a (inclusive) to v_b (exclusive).

$$\exists v_c \in V. \neg\exists v_n \in V.\ v_c \neq v_n \wedge v_c \in \text{pathStartNodes}(r_a, r_f) \wedge v_c \in \text{pathStartNodes}(r_b, r_f) \wedge v_n \in \text{pathStartNodes}(r_a, r_f) \wedge v_n \in \text{pathStartNodes}(r_b, r_f) \wedge |\text{pathStartNodes}(v_c, r_f)| < |\text{pathStartNodes}(v_n, r_f)| \tag{3.7}$$

This proposition defines v_c as the earliest node that is contained both in the paths from r_a and r_b to r_f.

$$\exists i \in \mathbb{N}. \exists j \in \mathbb{N}.\ i < j \wedge \text{s}(\text{path}(r_a, r_f)_j) = v_c \wedge \text{encounterEdge}(a, b) = \text{path}(r_a, r_f)_i \tag{3.8}$$

The edge on which the calculated encounter position of a and b lies preceeds the edge which starts at v_c within the path from r_a to r_f

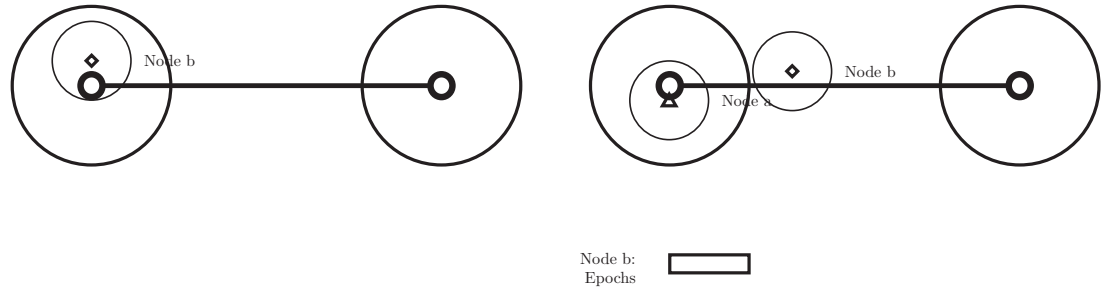
The following paragraphs explain the exact sequence of actions in GRAL in greater detail.
For each new package, a routine named `feed` is called (compare algorithm 1). It starts by registering previously unknown sensors in its persistent memory.
If the node has encountered a relay and its RSSI has fallen since the last package, set the contact's direction to `WITHDRAWAL`.
Look for the latest epoch that is not of the type `VOYAGE` before the possible multiple epochs of the type `RELAY_WITHDRAWAL` at the recent end of the epoch list. If that epoch is of the type `RELAY_APPROACH`, attempt to locate the packages and return early with a list of packages that got their position assigned.
If the relay's RSSI has instead risen since the last package, set the contact's direction to `APPROACH`, except if there were no last packages, then set it to `WITHDRAWAL`.
If the RSSI of the closest relay plus a tolerance constant matches or exceeds the highest RSSI ever measured for a relay, perform the steps outlined in this paragraph. If the last non-`VOYAGE` epoch was a `RELAY_APPROACH` one, add the package to a `RELAY_APPROACH` epoch and attempt to locate the packages and return early with a list of packages that got their position assigned. In the case that such epoch does not exist, try to add the package anyways. If the type of the epoch added by `addToEpochs` (see below) turns out to be `RELAY_APPROACH`, try to localize the packages and return that list.
Add the package to an epoch of matching type for that sensor. Also, try to localize and return if the actual type of the created epoch is `RELAY_APPROACH` and there was non-`VOYAGE` epoch previously or if the type is `RELAY_WITHDRAWAL`, and the most recent non-`VOYAGE` epoch had the type `RELAY_APPROACH`.
If finally there has no relay contact at all, return an empty list.
The `addToEpochs`(p, t) method from algorithm 1 adds p to the most recent epoch if it is of the specified type and creates a new one otherwise. It returns the type of the created epoch. If, however, the epoch is not of the type `VOYAGE` and the previous non-`VOYAGE` epoch has the same relay as the strongest contact, it merges all of the packages of the epochs after that into a new epoch of the type `RELAY_WITHDRAWAL`, with p as the most recent package. It will also create a new epoch, if there is a checkpoint for the with its timestamp $t \in]mostRecentEpoch.start; p.timestamp[$. The method moves the packages that have a timestamp that is greater than the checkpoints' timestamp to a new epoch of the same time. It sets the old epochs' end position to the checkpoint's position. The most recent package of the newly created epoch is p. Only the most recent checkpoint in the interval

Data: A package p with an identifier id, a list of encountered nodes and a timestamp
Result: A list of localized packages of that sensor

$sensor \leftarrow$ `nodeForId(`$p.id$`)`;
$relayContactList \leftarrow \{c \mid c \in p.contacts \wedge c$ is a relay$\}$;
$maxSignal \leftarrow \max(\max(\{c.RSSI \mid c \in p.contacts\}), maxSignal)$;
foreach $relayContact \in relayContactList$ **do**
    if *contact* **for** $relayContact.id \notin sensor.previousPackage$ **then**
        Set $relayContact.direction$ to APPROACH if $sensor.previousPackage$ is set, else use WITHDRAWAL;
    else
        Set $relayContact.direction$ to WITHDRAWAL or APPROACH w.r.t. $sensor.previousPackage$ and propositions 3.2, 3.3;
    end
    if $relayContact.direction =$ WITHDRAWAL **then**
        if *latest epoch not of type* VOYAGE *before the latest epoch that is not of type* RELAY_WITHDRAWAL *has type* RELAY_APPROACH **then**
            $sensor$`.addToEpochs(`p, RELAY_WITHDRAWAL`)`;
            return `tryLocatePackages(`$sensor$`)`;
        end
    end
end
if $|relayContactList| > 0$ **then**
    if $relayContactList.strongestContact + tolerance \geq maxSignal$ **then**
        if *there is an epoch* e *with* $e.type \neq$ VOYAGE **and** *the latest matching epoch is of the type* RELAY_APPROACH **then**
            $sensor$`.addToEpochs(`p, *epoch type for* $relayContactList.strongestContact.direction$`)`;
            return `tryLocatePackages(`$sensor$`)`;
        else if e *does not exist and* s`.addToEpochs(`p, RELAY_APPROACH`)` *yields* RELAY_APPROACH **then**
            return `tryLocatePackages(`$sensor$`)`;
        end
    end
    $latestRelevant \leftarrow$ latest epoch for which $type \neq$ VOYAGE;
    $type \leftarrow$ $sensor$`.addToEpochs(`p, *epoch type for* $relayContactList.strongestContact.direction$`)`;
    if $type =$ RELAY_WITHDRAWAL $\wedge$ $latestRelevant.type =$ RELAY_APPROACH **then**
        return `tryLocatePackages(`$sensor$`)`;
    else if $type =$ RELAY_APPROACH $\wedge$ $latestRelevant$ *is set* **then**
        return `tryLocatePackages(`$sensor, |epochs| - 1$`)`;
    end
else
    $sensor$`.addToEpochs(`p, VOYAGE`)`;
end
return *empty list*;

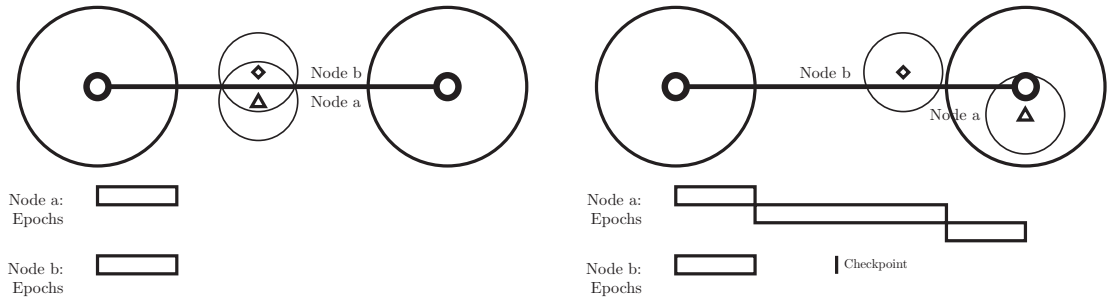
Algorithm 1: `feed` – Called for each new package

described above is used in this process. `nodeForId`(id) fetches the structure for the node with the identifier id out of the persistent memory.
The `tryLocatePackages` (algorithm 2) routine tries to assign positions to all of the epoch packages of a node where possible.
If the earliest non-`VOYAGE` epoch of a sensor is of the type `RELAY_APPROACH` and has only one package, then set the type to `RELAY_WITHDRAWAL` if the subsequent epoch is a `VOYAGE` epoch, otherwise, check if the subsequent epoch has contact to the relay with the highest RSSI in this epoch. If so, put its package into the subsequent epoch and delete it.
For every epoch, try to calculate the positions of the packages in the epoch. The subroutine may return a signal that prompts an early return that may include a list of packages with assigned positions. Then, find the package that contains the strongest RSSI for any other received nodes during that epoch. If the epoch type is `VOYAGE`, the contacted sensor has been at a relay before the contact occurred, and the starting relay for the positions in the epoch is set, then go and find the earliest graph vertex at which the routes of the two nodes could have joined each other. Calculate the distance between this epoch's starting relay and the shared vertex. If the previously calculated package position is smaller than that distance, split the epoch into two with the package where the contact with the highest RSSI occurred as the end of the first of the two new epochs. The operation is, however, only performed if the new epoch would have some packages. The end position of the first of the two newly separated epochs is set to be the calculated distance to the first shared vertex. Proceed to recalculate the positions of the packages in the first epoch.
Furthermore, for all contacted sensors, a rendezvous should be added to them if the package which contains the strongest RSSI for that sensor during the epoch has all fields of its position set and the path between the last relay the contact has been at and the current edge destination contains the edge of the epoch.
The `next`(e) method from algorithm 2 called on a set returns the element that comes after e from itself. The `prepend`(e) method of a collection inserts e at index 0, similarly `remove`(e) removes e from the collection and `getIndex`(e) returns the index of e. `getEarliestSharedNode`($node$, $position$) returns the *confuence node* v_c as defined in equation 3.7 for $node$ as r_b, $position.start$ as r_a and $position.destination$ as r_f. `getDistance`(n_a, n_b) returns the sum of the edge weights on the shortest path between the two argument nodes. `graphEdgePosition`($position$) returns a position that is equivalent to its argument but for which an edge with the same start and end vertices exists and with the distances adjusted accordingly. `pathContains`($start$, $dest$, $edgePos$) returns a boolean that describes whether the path form $start$ to $dest$ contains the edge described by $edgePos$. The `addRendezVous`($checkpoint$) method adds a checkpoint to its sensor if the checkpoint occurred after the most recent package positions have been determined. `mergeAndClearEpochs`($sensor$, i) returns the concatenated list of the sensor's epochs' packages up until the epoch with the index i. It sets the last known position of the sensor to the last returned package's position, the processed epochs are removed from the sensor.



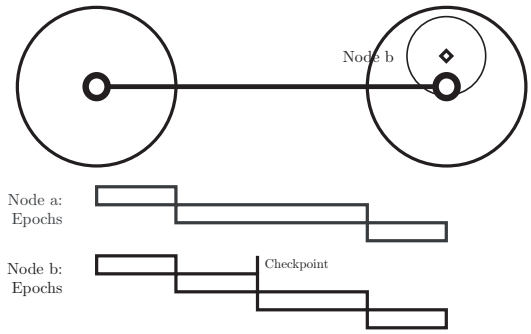
(a) Node b starts at a relay

(b) Node b left the relay, a second node a arrived



(c) Nodes a and b encounter one another

(d) Node a reaches the relay, a checkpoint for b is created



(e) Node b reaches the relay, its `VOYAGE` epoch is split at the checkpoint

Figure 3.3: Checkpoint creation on a single edge

Data: A sensor s and an optional index $maxIndex$ to determine which epochs are processed

Result: A list of localized packages of that sensor

```
1  epochs ← s.epochs;
2  if for the earliest epoch e: e.type ≠ VOYAGE then
3      if |e.packages| = 1 ∧ e.type = RELAY_APPROACH then
4          if epochs.next(e).type = VOYAGE then
5              e.type ← RELAY_WITHDRAWAL;
6          else if epochs.next(e).contacts contains contacts to the relay of this epoch then
7              epochs.next(e).packages.prepend(e.packages);
8              epochs.remove(e);
9          end
10     end
11 end
12 foreach epoch ∈ epochs where getIndex(epoch) < maxIndex do
13     result ← tryCalculateEpochPositions(s, epochs.getIndex(epoch));
14     if some result then return result;
15     foreach contactedSensor ∈ {nodeForId(d.id) | d ∈ epoch.strongestContacts} do
16         positionDuringContact ←
            epoch.strongestContacts[contactedSensor.id].position;
17         if epoch.type = VOYAGE and contactedSensor has been at a relay before the
            contact occurred and positionDuringContact.start is set and Path
            Rectification is enabled then
18             earliestConfluence ←
                getEarliestSharedNode(contactedSensor.lastRelay, positionDuringContact);
19             distanceToConfluence ←
                getDistance(positionDuringContact.start, earliestConfuence);
20             if positionDuringContact.distanceTraveled < distanceToConfluence then
21                 split epoch ∈ epochs into two with
                    epoch.strongestContacts[contactedSensor.id] as the last package of the
                    first one and the endPosition set to distanceToConfluence;
22                 if new epoch has |packages| > 0 then
23                     Insert new epoch into sensor epoch list after the original one;
24                     tryCalculateEpochPositions(s, epochs.getIndex(epoch));
25                     maxIndex ← maxIndex + 1;
26                 end
27             end
28         end
29         if positionDuringContact has start and end set and checkpoints are enabled
            then
30             edgePosition ← graphEdgePosition(positionDuringContact);
31             lastRelay ← last contacted relay of s;
32             if pathContains(lastRelay, edgePosition.end, edgePosition) then
33                 contactedSensor.addRendezVous(graphEdgePosition(positionDuringContact),
                    s, epoch.strongestContacts[contactedSensor.id].timestamp);
34             end
35         end
36     end
37 end
38 return mergeAndClearEpochs(s, min(maxIndex, epochs.size − 1));
```

Algorithm 2: tryLocatePackages – Prepares and refines localization

Finally, the `tryCalculateEpochPositions` tries to localize packages for a single epoch in a sensor (algorithm 3).
The routine determines the distance traveled during the epoch and the starting position depending on the epoch time. With this information, it calls the `setPackagePositions` routine of the epoch and returns.

- **If the epoch is of the type `VOYAGE`:**

  Determine the next relay contact by searching for the first subsequent epoch with such contact. If that did not yield any result and the end position of the previous epoch is defined, return `mergeEpochPackagesAndClear` for up to the current epoch exclusive.

  Try to determine the last relay contact as the relay that was the start of the previous epoch. Also, set the last known position to the end position of the last epoch. If the end position indicates that the node has arrived at its destination, set the last relay to the previous destination and the last known position to the start of the way from the last relay to the next relay.

  If the last relay was not found using the instructions from the previous paragraph, try to use the destination of the last known position of the sensor. If that too failed and the next relay contact is set, set the position of all packages of the epoch to have an unknown start, next relay as destination and zero of an infinite distance traveled and return early with no value. In the case that neither the last nor the next relay could be found, an empty list is returned.

  Set the total distance to the distance between the last and the next relay. Determine the location of the node on a link edge by setting it to the traveled distance of the last known position if the start and endpoints are identical to the previous and next relay, else set it to zero.

  If there is a path between the last and the next relay that contains start and end of the set end position of the epoch, calculate the distance from the start to that position as this epochs distance span, otherwise set it to the distance between the start and the border of the next relay's reception radius.

  The starting position is set to be a new position with previous and next relays as the start- and endpoints, the previously calculated starting position on the edge as the traveled distance and the calculated total distance.

- **If the epoch is not of the type `VOYAGE`:** Get the relay with the strongest RSSI from any package of the epoch as this epoch's relay. Try to determine the last relay by looking for the latest previous epoch with relay contacts. If that fails, set the last relay to the destination of the last known position of the sensor if that does not equal the current contact. In that case, use the start of the last known position as the previous relay.

  Try to determine the next relay by looking for the earliest of the subsequent epochs with relay contacts.

  - **If the epoch is of the type `RELAY_APPROACH`:**

    If the epoch's end position is set, look at whether the start relay of the end position is the relay with the highest RSSI in this epoch. If so, convert the epoch type to `RELAY_WITHDRAWAL` and return the result of another call of this method, unless this is already a recursive call, then return nothing. Otherwise, set the previous relay to the start relay of the epoch's end position.

    In the case that the previous relay is defined, set the total distance to the distance between it and this epoch's relay. Set the starting position to a new position with the previous

relay and this epoch's relay as start and destination, the traveled distance as zero and the calculated total distance. Otherwise, set each package's position to the next relay and this epoch's relay as start and destination, with the distance as zero and the total distance as their respective distance if the next relay is set. Return early with no value. If both the next and the last relay are unset, return an empty list.

– **If the epoch is of the type `RELAY_WITHDRAWAL`:**

  Similarly to the first case, we look at the end position of the epoch. If its end position is set, look at whether the destination relay of the end position is the strongest relay of this epoch. If so, convert the epoch type to `RELAY_APPROACH` and return the result of another call of this method, unless this is already a recursive call, then return nothing. Otherwise, set the next relay to the destination relay of the epoch's end position.

  If the next relay is undefined and this is not the sensor's first unevaluated epoch, return the value of the sensor's `mergeEpochPackagesAndClear` procedure. If the next relay is undefined and this is the first unevaluated epoch, return an empty list.

  Set the total distance to the distance between this epoch's relay and the next relay. Set the starting position to this epoch's relay and the next relay as start and destination, distance as zero and their total distance. Set the distance to the radius of this epoch's relay.

`applyPositionToEndpoints`($position$, id_{v_a}, id_{v_b}) acts as the inverse of `graphEdgePosition` in that it returns a position equivalent to its first argument with v_a as the start node and v_b as the end node. `setPackagePositions`($start$, $distance$) finally sets the packages of its epoch to the corresponding positions by linearly interpolating positions using the last packages timestamp and the previous epoch's end time as the duration parameter. The *distance* parameter sets the distance traveled from start to end of that epoch, and *start* sets the initial position.
I consider GRAL to be range-free since it does not use the RSSI to perform position estimation; instead, it uses preloaded data about the environment and the characteristics to do that. The RSSI, which the implementor can substitute for any other proximity indicator, is only used to determine the qualitative difference of increasing or decreasing proximity. The method is centralized because otherwise, extensive information exchange between mobile nodes, preloading them with the environment graph, and executing graph algorithms on them would be necessary. Adding checkpoints to a node to which there is no more contact would be hard in a distributed setting. A centralized backend is thus better suited for this approach.
A reference implementation of GRAL can be obtained at `https://github.com/reknih/GRAL`.

3.2 System architecture

The system needs supporting infrastructure in the actual WSN to be able to transmit all of its data to the backend running GRAL. The nodes all record their measurement packages and transmit them if a relay is visible or store them for later transmission. There is binary storage as well as a transport format to save memory on the nodes. The nodes use the MQTT protocol (Banks and Gupta 2014) to send their collected packages to relays. The nodes send each package in a dedicated message to the message queue. The system uses MQTT on the relays, and mobile nodes since the protocol is lightweight and designed for Internet of Things use cases with slow connection speeds.
The relays use a WAN wireless technology like LoraWAN to push their messages to the centralized backend. There, a bridge program feeds the received messages into an Apache Kafka message broker running atop of Apache Zookeeper. An Apache Flink application listens to a Kafka topic and decodes the incoming data to Java objects that inherit GRAL's package type. The Flink application

executes GRAL with these packages, as part of a `FlatMap`. It proceeds to output a JSON object string for each localized package. A client application may now use this output of the Flink job. The system additionally includes a distributed file system in which it saves all processed data for archival purposes. All of the applications on the backend will be containerized separately to make deployment more manageable and to prepare for upscaling of the solution.

Data: A sensor s and a index i of one of its epochs as well as whether it is called recursively
Result: A list of localized packages of that sensor to signal a premature stop or nothing

$epoch \leftarrow s.epochs[i]$;
define *distance*, *startingPosition*;
if $epoch.type =$ VOYAGE **then**
  $strongestFutureContact \leftarrow$ first relay r seen $\in \{e.contacts \mid e \in s.epochs \wedge$ index of $e > i\}$;
  if *strongestFutureContact not set* **then**
    if $s.epochs[i-1].endPosition$ *is set* **then**
      return `mergeAndClearEpochs`(s, i);
    end
    return *empty list*;
  end
  Determine $lastId$ (id of last contacted relay) and $lastKnownPosition$ from $s.epochs[i-1].endPosition$ or $s.lastPosition$;
  if $lastKnownPosition$ *or* $lastId$ *could not be determined* **and** $strongestFutureContact$ *is set* **then**
    foreach *package* $p \in epoch.packages$ **do**
      $p.position \leftarrow (unset, strongestFutureContact, 0, +\infty)$;
    end
    return *nothing*;
  else
    return *empty list*;
  end
  $totalDistance \leftarrow$ `getDistance`(*lastId, strongestFutureContact.id*);
  $alreadyGoneDistance \leftarrow 0$;
  if $lastKnownPosition.start = lastId \wedge lastKnownPosition.destination = strongestFutureContact$ **then**
    $alreadyGoneDistance \leftarrow lastKnownPosition.distanceTraveled$;
  end
  if *there is a path between* $strongestLastContact$ *and* $strongestFutureContact$ *that contains* $epoch.endPosition$*'s start and end* **then**
    $distance \leftarrow$ `applyPositionToEndpoints`(*epoch.endPosition, lastId, strongestFutureContact*)$.distanceTraveled - alreadyGoneDistance$;
  else
    $distance \leftarrow totalDistance - (alreadyGoneDistance + strongestFutureContact.radius)$;
  end
  $startingPosition \leftarrow$ (`nodeForId`(*lastId*)$, strongestFutureContact, alreadyGoneDistance, totalDistance$);
  `/* Contiunation in algorithm 4 -- See there for non-VOYAGE cases and final instructions */`

Algorithm 3: `tryCalculateEpochPositions` – Assigns positions to packages

```
   /* Contiunation of algorithm 3 -- See there for VOYAGE case            */
31 else
32     strongestContact ← nodeForId((max_RSSI(epoch[latest].contacts)).id);
33     distance ← strongestContact.radius; totalDistance ← 0;
34     Define prevRelay as the relay the node came from using the previous epochs or
        s.lastPosition;
35     nextRelay ← first relay seen ∈ {e.contacts | e ∈ s.epochs ∧ index of e > i};
36     if epoch.type = RELAY_APPROACH then
37         if epoch.endPosition is set then
38             if epoch.endPosition.start = strongestContact then
39                 if this is not a recursed call then
40                     epoch.type ← RELAY_WITHDRAWAL;
41                     return tryCalculateEpochPositions(s, i, recursed = true);
42                 end
43                 return nothing;
44             else
45                 prevRelay ← epoch.endPosition.start;
46             end
47         end
48         if prevRelay is set then
49             startingPosition ← (prevRelay, strongestContact, totalDistance −
                strongestContact.radius, getDistance(prevRelay, strongestContact));
50         else if nextRelay is set then
51             startingPosition ←
                (strongestContact, nextRelay, 0, getDistance(strongestContact, nextRelay));
52         else
53             return empty list;
54         end
55     else
56         if epoch.endPosition is set then
               /* Analogous to lines 38-46 with types swapped and considering
                  destination instead of start                              */
57         end
58         if nextRelay is not set then
59             if i > 0 then
60                 return mergeAndClearEpochs(s, i);
61             end
62             return empty list;
63         end
64         startingPosition ←
            (strongestContact, nextRelay, 0, getDistance(strongestContact, nextRelay));
65         distance ← strongestContact.radius;
66     end
67 end
68 epoch.setPackagePositions(distance, startingPosition);
69 return nothing;
```

Algorithm 4: `tryCalculateEpochPositions` (cont'd) – Assigns positions to packages

4 Results

In order to determine the performance of GRAL in various situations, I evaluated it by generating several datasets of the paths of nodes and their wireless contacts, each of which contains 200 instances. These test sets were built using a mock sensor node implemented in Rust. It discriminates between normal nodes and relays in its simulated wireless neighbourhood and saves regularly created packages until it can connect to a relay and send its backlog. An environment server keeps track of the nodes by using a time-discrete step based simulation in which, at each step, it updates the positions of all mobile nodes by adding a constant and a noise term $2 \cdot (-1)^X$ for which X is of uniform distribution in $[0; 2[$. The nodes draw their mock sensor values from this server. It also acts as the radio device of the node by transmitting which other nodes are within wireless range. For each of them, the environment server calculates an RSSI-like strength indicator and sends it to the node.

The GRAL Java reference implementation was then fed the packages of in each instance in the order they were transmitted to a simulated relay by the mock sensor node. The radius parameter for each relay was set to $\sqrt{10}$, which exactly matches the relay radius that the environment server used. Each dataset was tested using the appropriate environment graph.

For evaluation, I calculated the Root Mean Square Error (RSME) for both each instance ***(iRSME)*** and the complete dataset ***(dRSME)***. If appropriate, I evaluated the algorithm with the checkpoint functionality (algorithm 2, lines 29-34) and path rectification (algorithm 2, lines 15-28) both enabled and disabled.

In the first dataset, there is one sensor node per instance. There are no multiple possible paths; the environment consists of an edge with weight 50 from a start to a central relay and another edge of the same weight from the center to the final relay. The global error is dRSME = 6.596, which is higher than the error value category that most instances fall in (see figure 4.1).

Figure 4.2 shows instances for which GRAL is notably (un)suited. In 4.2a, we see that the node stays in the proximity of the first and second relay respectively for a while, creating prolonged and flat `RELAY_WITHDRAWAL` epochs. If the sensor is not at a relay, it traverses the environment with a constant speed that can be matched by the linear approximation between the start- and endpoints of the `VOYAGE` epochs. On the contrary, 4.2b shows positional oscillations and turning back for a while without returning to or arriving at a relay create high errors as there is no info besides the

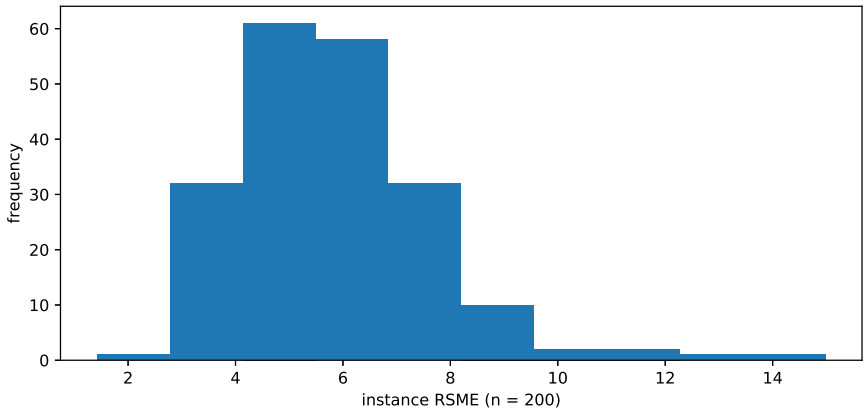
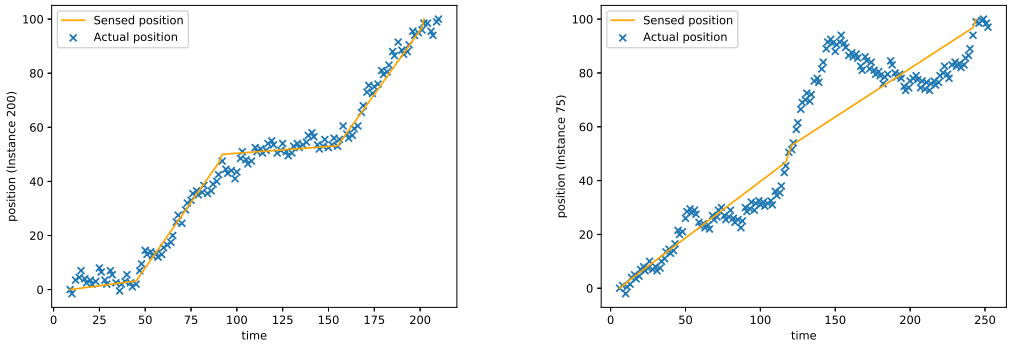
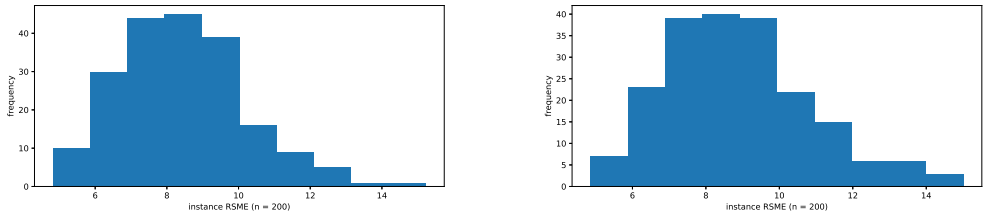


Figure 4.1: Distribution of iRSME for a dataset with one sensor node and no junctions



(a) iRSME = 3.174 **(b)** iRSME = 10.7

Figure 4.2: Instances with low or high iRSME from the first dataset



(a) Checkpoints disabled **(b)** Checkpoints enabled

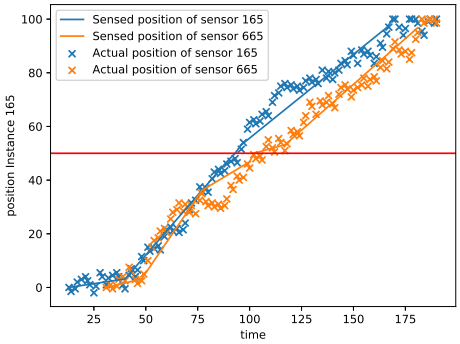
Figure 4.3: Distribution of iRSME for a dataset with two sensor nodes and no junctions

start- and endpoints in VOYAGE epochs in environments with a single mobile node to improve the estimations' accuracy.

In this dataset, I use the same environment but with two sensors at the same time. The second node starts between zero and four seconds later (randomly chosen per instance) than the first one. In this dataset, there are instances where the two sensors recorded contact with one another in several packages. This property being present means that GRAL can use checkpoints in this dataset. Consequently, figure 4.3 shows the distributions of error for localization with and without checkpoints. The global error for the former is dRSME = 6.833, for the latter it is dRSME = 6.503. To see why the error increases with the functionality enabled, consider the two instances in figure 4.4: In 4.4a, the nodes move with relatively few turbulence. At around timestamp 70, the nodes meet. At this point, without checkpoints, the position of node 665 (orange) would be estimated slightly too low, whereas the position estimate for node 165 (blue) is relatively precise. Node 165 arrives first at the center relay (red line), so GRAL uses its estimates as the base for the checkpoint position. The checkpoint thus lifts the curve of node 665, correcting the estimation error.

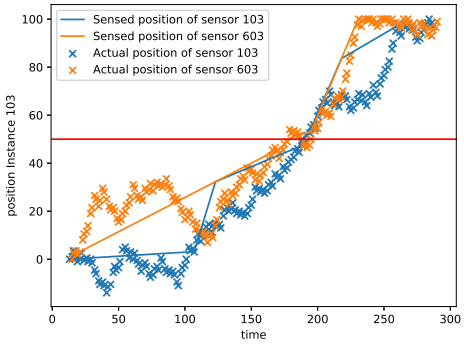
Conversely, in 4.4d the nodes oscillate during the voyage epoch. Since this behavior causes the positional estimates to be imprecise, the checkpoint position is of low quality. The nodes meet at about timestamp 120 with the orange node arriving first at the next relay. Its positional estimate ahead of the actual position by a significant amount, so the checkpoint is accordingly far off. Additionally, node 103 was relatively steady in its motion, so its estimates were of good quality, aggravating the error that is introduced by pulling its estimates up to the checkpoint.

The checkpoints can increase or diminish the precision of the algorithm, but the dRSME values suggest that it is overall detrimental to its performance. However, when checkpoints are turned on, the measurements that happened in the same area are assigned similar positions, independent of which node captured them.



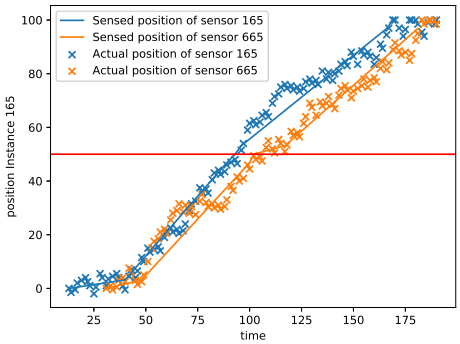
(a) iRSME = 5.788

checkpoints enabled



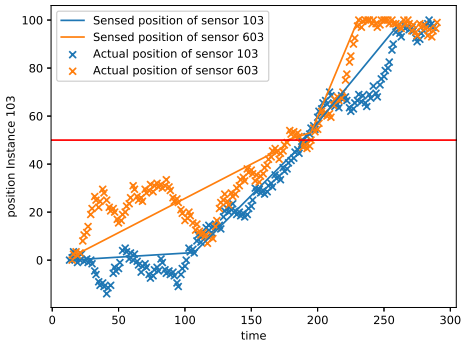
(b) iRSME = 14.953

checkpoints enabled



(c) iRSME = 6.076

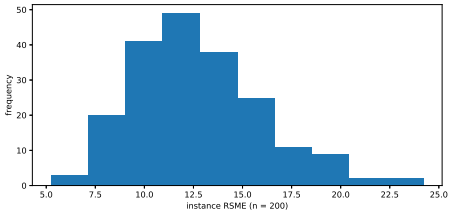
checkpoints disabled



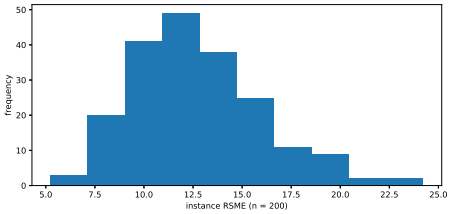
(d) iRSME = 11.708

checkpoints disabled

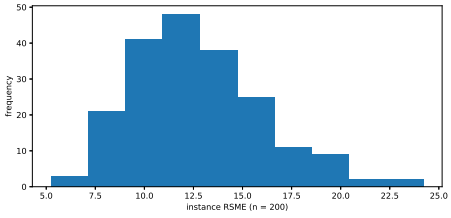
Figure 4.4: Instances with low or high iRSME from the second dataset



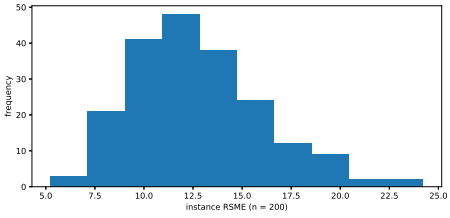
(a) Checkpoints disabled, rectifcation disabled



(b) Checkpoints enabled, rectifcation disabled

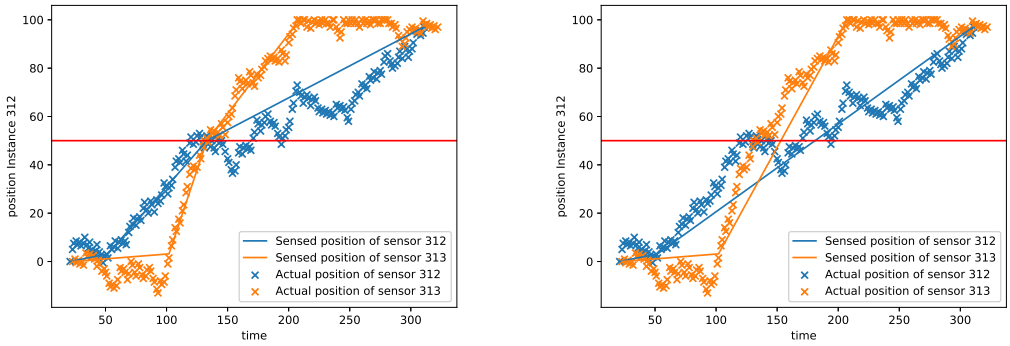


(c) Checkpoints disabled, rectifcation enabled



(d) Checkpoints enabled, rectifcation enabled

Figure 4.5: Distribution of iRSME for a dataset with two sensor nodes and a junction without a relay in the middle



(a) iRSME = 9.685

rectifcation enabled

(b) iRSME = 11.889

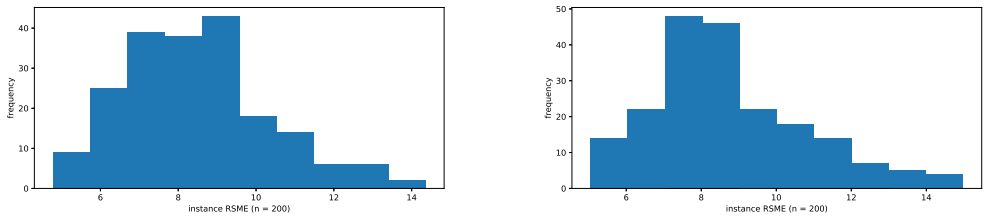
rectifcation disabled

Figure 4.6: Comparison between enabled and disabled path rectification

The dataset of 4.5 lacks the center relay at position 50 that the other sets have. Instead, it features a Y-shaped topology: There are two relays, at each of which one sensor node starts its journey. The two starting relays are connected by an edge of weight 50 to the central junction without a relay respectively. At that junction, there is another edge to the final relay of the same weight. Both sensor nodes start at an independent, randomly chosen timestamp between zero and four seconds after the start of the simulation. In this scenario, path rectification becomes relevant as nodes may encounter other nodes that did take the same path as them before encountering a shared relay. Therefore, four error distribution diagrams are provided. One for both checkpoints and path rectification disabled (dRSME = 10.034), only checkpoints enabled (dRSME = 10.034), only rectification enabled (dRSME = 10.024) and finally for both techniques enabled at the same time (dRSME = 10.04). The global error is higher than in the previous datasets because of the uncertainty introduced by the missing center relay.

Let's consider an example for an instance in which path rectification occurs: In figure 4.6, two sensors have different speeds before and after the junction where they first meet which is signified by the red line. They also, importantly, encounter each other before the next relay. When rectification is enabled, the algorithm recognizes the lower bound position that the sensors from the different relays have to be at to meet and splits the epoch into two with the earliest possible confluence as the end of the first one. This technique strictly improves accuracy for the local data point since, if it is activated, the position estimate had to be an underestimation. Path rectification only increases the position to the first theoretically possible location; therefore, there is no possibility to 'overshoot' the correct position. However, there is the slight possibility that it increases global error if the nodes' positions oscillate wildly and the tighter fit removes the mutual correction that the straight line provided for over- and underestimations. In my test set, only three instanced triggered rectification. However, a test with more than two sensor nodes in the same environment would likely yield much higher usage of the technique because having more nodes increases the likelihood of a qualifying encounter.

Note that if a relay is placed at the junction, it becomes impossible for nodes to have an encounter that qualifies them for path rectification since they always contact the relay as well as their peers if they arrive at a new junction. Figure 4.7 shows the distribution of error in such a dataset with a Y-shaped environment and three relays (`dRSME` = 6.670 for checkpoints enabled, `dRSME` = 6.410 without). The error values and the distributions are similar to those found in figure 4.3.



(a) Checkpoints disabled

(b) Checkpoints enabled

Figure 4.7: Distribution of iRSME for a dataset with two sensor nodes and a junction with a relay

5 Conclusions

In this thesis, I proposed the novel Graph-based Localization algorithm for Wireless Sensor Networks and evaluated it using multiple sets of noisy problem instances. Overall, GRAL has been shown to be able to provide range-free localization for environments that can be modeled by a tree graph. The algorithm's performance depends on the variance of the motion speed of the nodes but is generally sufficient for applications like surveying. GRAL is appropriate for WSNs with reference nodes that cannot use expensive dedicated localization hardware on their nodes and cannot afford to use the nodes to do significant computational work. GRAL leverages the backend, which is present in most deployments anyhow to do its computation. The approach is, however, not suited for applications that require position data for their nodes in real-time. The thesis proposed a scalable and robust scheme for the implementation of GRAL.

In contrast to other range-free approaches, GRAL can even deliver location estimates for moments where a node has no contact to any other nodes as the simulations have demonstrated. The algorithm is furthermore capable of compensating for a sparse deployment of reference nodes if there are many mobile nodes, and the environment is frequently branching using path rectification.

While the checkpoint variant of GRAL enables stronger guarantees that readings which some nodes have stored at the same position and time will be assigned the same position, it did unfortunately not increase overall precision of the system for the test sets.

Future improvements to GRAL may include the use of exclusion zones for nodes that receive a node situated within the radius of a relay but not the relay itself as, e.g., in the work of Baggio and Langendoen 2008. Another intriguing possibility is to use ranged techniques like TDoA while contact with a relay is possible to increase accuracy. Also, non-linear interpolation between epoch start and end positions could be a solution for creating position estimates that fit the actual motion better.

Bibliography

Arandjelovic, Relja, et al. 2016. "NetVLAD: CNN Architecture for Weakly Supervised Place Recognition". In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Piscataway, NJ, USA: IEEE.

Baggio, Aline, and Koen Langendoen. 2008. "Monte Carlo localization for mobile wireless sensor networks". *Ad Hoc Networks* 6 (5): 718–733. doi:10.1016/j.adhoc.2007.06.004.

Bahl, P., and V. N. Padmanabhan. 2000. "RADAR: an in-building RF-based user location and tracking system". In *Proceedings IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, 775–784 vol.2. Piscataway, NJ, USA: IEEE. doi:10.1109/INFCOM.2000.832252.

Banks, Andrew, and Rahul Gupta. 2014. *MQTT Version 3.1.1*. Visited on 09/04/2019. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

Buratti, Chiara, et al. 2009. "An Overview on Wireless Sensor Networks Technology and Evolution". *Sensors* 9 (9): 6869–6896. doi:10.3390/s90906869.

Dag, Tamer, and Taner Arsan. 2018. "Received signal strength based least squares lateration algorithm for indoor localization". *Computers & Electrical Engineering* 66 (1): 114–126. doi:10.1016/j.compeleceng.2017.08.014.

Dawoud, Safaa. 2012. *GNSS principles and comparison*. Berlin, DE: Technische Universität Berlin. https://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/gnss-principles-and-comparison_dawoud.pdf.

Haug, Hartmut. 2006. *Statistische Physik*. 2nd ed. Vol. 1. Berlin, Heidelberg, DE: Springer. ISBN: 978-3-540-25629-8.

He, Tian, et al. 2003. "Range-free Localization Schemes for Large Scale Sensor Networks". In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, 81–95. MobiCom '03. New York, NY, USA: ACM. doi:10.1145/938985.938995.

Hightower, J., and G. Borriello. 2001. *A Survey and Taxonomy of Location Systems for Ubiquitous Computing*. Seattle, WA, USA: University of Washington, Computer Science and Engineering. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.8117&rep=rep1&type=pdf.

Hu, Lingxuan, and David Evans. 2004. "Localization for Mobile Sensor Networks". In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, 45–57. MobiCom '04. Event-place: Philadelphia, PA, USA. New York, NY, USA: ACM. doi:10.1145/1023720.1023726.

Karl, Holger, and Andreas Willig. 2007. *Protocols and Architectures for Wireless Sensor Networks*. 4th ed. West Sussex, UK: John Wiley & Sons. ISBN: 978-0-470-51923-3.

Krishnamachari, Bhaskar. 2005. *Networking Wireless Sensors*. New York, NY, USA: Cambridge University Press. ISBN: 978-0-521-83847-4.

Kuang, Jian, Xiaoji Niu, and Xingeng Chen. 2018. "Robust Pedestrian Dead Reckoning Based on MEMS-IMU for Smartphones". *Sensors (Basel, Switzerland)* 18 (5): 1391. doi:10.3390/s18051391. JSTOR: 29724003.

Lin, J., Y. Liu, and L. M. Ni. 2007. “SIDA: Self-organized ID Assignment in Wireless Sensor Networks”. In *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, 1–8. Piscataway, NJ, USA: IEEE. doi:10.1109/MOBHOC.2007.4428604.

Mao, Guoqiang, Barış Fidan, and Brian D. O. Anderson. 2007. “Wireless sensor network localization techniques”. *Computer Networks* 51 (10): 2529–2553. doi:10.1016/j.comnet.2006.11.018.

Niculescu, D., and B. Nath. 2001. “Ad hoc positioning system (APS)”. In *GLOBECOM’01. IEEE Global Telecommunications Conference*, 5:2926–2931. Piscataway, NJ, USA: IEEE. doi:10.1109/GLOCOM.2001.965964.

Parker, T., and K. Langendoen. 2004. “Refined statistic-based localisation for ad-hoc sensor networks”. In *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004.* 90–95. Piscataway, NJ, USA: IEEE. doi:10.1109/GLOCOMW.2004.1417555.

Patwari, N., R. J. O’Dea, and Yanwei Wang. 2001. “Relative location in wireless networks”. In *IEEE VTS 53rd Vehicular Technology Conference, Spring 2001. Proceedings*, vol. 2, 1149–1153 vol.2. Piscataway, NJ, USA: IEEE. doi:10.1109/VETECS.2001.944560.

Priyantha, Nissanka, et al. 2003. *Anchor-Free Distributed Localization in Sensor Networks*. Tech Report 892. Cambridge, MA, USA: MIT Laboratory for Computer Science. http://cricket.lcs.mit.edu/papers/TechReport892.pdf.

Rao, Ananth, et al. 2003. “Geographic Routing Without Location Information”. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, 96–108. MobiCom ’03. Event-place: San Diego, CA, USA. New York, NY, USA: ACM. doi:10.1145/938985.938996.

Rappaport, Theodore S. 2002. *Wireless communications: Principles and practice*. 2nd ed. Prentice Hall communications engineering and emerging technologies series. Upper Saddle River, NJ, USA: Prentice Hall. ISBN: 0-13-042232-0. 2019-08-25.

Rashid, H., and A. K. Turuk. 2015. “Dead reckoning localisation technique for mobile wireless sensor networks”. *IET Wireless Sensor Systems* 5 (2): 87–96. doi:10.1049/iet-wss.2014.0043.

Rault, Tifenn, Abdelmadjid Bouabdallah, and Yacine Challal. 2014. “Energy efficiency in wireless sensor networks: A top-down survey”. *Computer Networks* 67 (1): 104–122. doi:10.1016/j.comnet.2014.03.027.

Ray, S., et al. 2004. “Robust location detection with sensor networks”. *IEEE Journal on Selected Areas in Communications* 22 (6): 1016–1025. doi:10.1109/JSAC.2004.830895.

Reinhardt, Andreas, Sebastian Zöller, and Delphine Christin. 2014. “Wireless Sensor Networks and Their Applications: Where Do We Stand? And Where Do We Go?” In *Proceedings of the 13th GIT/ITG Fachgespräch Sensornetze*, 13:1–3. Potsdam, DE. https://www.cs.uni-potsdam.de/bs/research/docs/slides/2014/sc14.pdf.

Sadler, B. M., and A. Swami. 2006. “Synchronization in Sensor Networks: an Overview”. In *MILCOM 2006 - 2006 IEEE Military Communications conference*, 1–6. Piscataway, NJ, USA: IEEE. doi:10.1109/MILCOM.2006.302459.

Singh, Santar Pal, and S. C. Sharma. 2015. “Range Free Localization Techniques in Wireless Sensor Networks: A Review”. *Procedia Computer Science* 57 (1): 7–16. doi:10.1016/j.procs.2015.07.357.

Souza, Luciana Moreira Sá de, Harald Vogt, and Michael Beigl. 2007. *A Survey on Fault Tolerance in Wireless Sensor Networks*. Karlsruhe, DE: SAP Research. https://pdfs.semanticscholar.org/4ba5/152a65bb10a96a3fb6481554f75a99944548.pdf.

Tubaishat, M., and S. Madria. 2003. “Sensor networks: an overview”. *IEEE Potentials* 22 (2): 20–23. doi:10.1109/MP.2003.1197877.

Want, Roy, et al. 1992. “The Active Badge Location System”. *ACM Transactions on Information Systems* 10 (1): 91–102. doi:10.1145/128756.128759.

Ward, A., A. Jones, and A. Hopper. 1997. “A new location technique for the active office”. *IEEE Personal Communications* 4 (5): 42–47. doi:10.1109/98.626982.

Zhang, Y., N. Meratnia, and P. Havinga. 2010. “Outlier Detection Techniques for Wireless Sensor Networks: A Survey”. *IEEE Communications Surveys Tutorials* 12 (2): 159–170. doi:10.1109/SURV.2010.021510.00088.

Acknowledgements

I would like to thank the Complex and Distributed IT Systems group of Odej Kao, particularly my supervisor, Felix-Lorenz Valentin, who was supporting me overcoming the inevitable technical difficulties. He did also assist me in navigating the dos and don’ts of academic writing with exhaustive notes. My parents were there when I churned out another chapter draft for my thesis. My father, Hartmut Haug, helped me to improve its quality by sharing knowledge from his longstanding scientific expertise. Furthermore, I’d like to thank my fellow student and friend, Laurenz Mädje, who did provide motivation for me by developing a brilliant thesis of his own. Crucially, he also did not stop to scrutinize this text and its underlying ideas by relentlessly telling me what he did not get and why. Finally, I’d like to thank Christine Solnon at the Institut National des Sciences Appliquées de Lyon and her course on practical algorithm design for calling my attention to the possible pitfalls in statistical evaluation of programs. This know-how was an indispensable foundation for chapter 4.