Leiden University

Computer Science

Mobile radio tomography: Autonomous vehicle planning for dynamic sensor positions

Name:	Leon Helwerda
Date:	August 29, 2016
1st supervisor:	Walter Kosters (LIACS)
2nd supervisor:	Joost Batenburg (CWI & MI)

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Radio tomography is a collection of techniques that measures the signal strength of low-energy radio waves which are exchanged between sensors around an area, and reconstructs information about objects within that area. We show that it is possible to replace commonly-used static sensors with a mobile sensor network. Unmanned vehicles move the wireless sensors around based on routes that visit locations where they perform measurements, ensuring that they wait for other vehicles along the way.

We investigate planning algorithms that help us to search for routes, and propose new algorithms that solve problems related to assigning sensor positions to the vehicles and avoiding collisions between them. We implement a toolchain that is capable of performing the radio tomographic measurements, with support for rovers, drones and other vehicles, as well as peripheral sensors for communication and monitoring.

We experiment with the complete toolchain in order to determine the performance and effectiveness. The results indicate that planning the sensor assignments and routes is a difficult problem, but an automatically generated mission could compete with a hand-made one in terms of the quality of the tomographic image.

Keywords: *radio tomography, reconstruction, tomographic imaging, autonomous vehicles, mission planning, search algorithms, assignment problems, sensor positioning problem, swarms, multi-agent synchronization, collision avoidance, multiobjective optimization.*

Contents

1 Introduction

Radio tomography is an emerging field of science that deals with the detection of objects or persons within an area. The techniques employed by radio tomography make use of sensor measurements in order to perform this detection. There is no need for the objects themselves to have any form of active sensor device or passive marker tag.

The measurements that radio tomography techniques collect come from wireless sensors. These sensors send and receive information to each other. For each packet of information that one sensor receives, the wireless module determines what the *signal strength* is at that given moment. This may be the actual transmission power that the wireless antenna receives or an approximate indication thereof.

Each combination of wireless sensors that communicate in this way form a *link*. Assuming that we know the positions of the sensors, then we also know where the links are inside a *network* of many sensor links. Figure 1 shows an example of such a network.

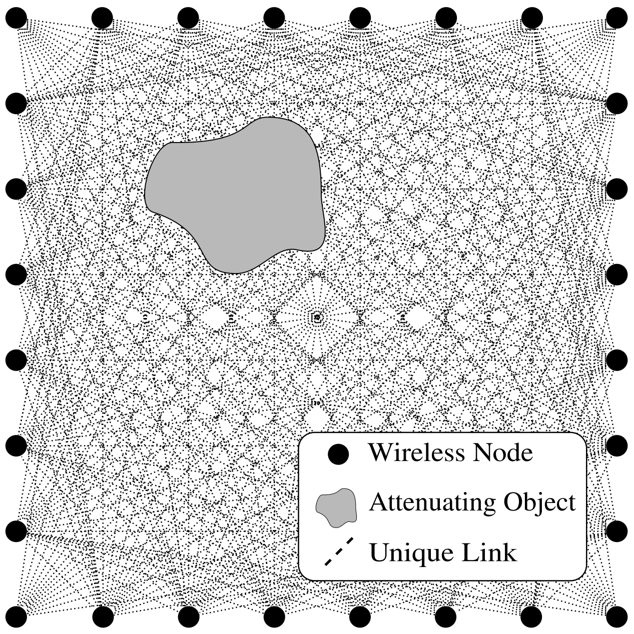


Figure 1: A network used in radio tomography. The objects inside the network may be reconstructed using the signal strength measurements of the links.

The signal strength of one link is affected by the objects that are located in between the two sensors. There are certain *paths* that the signal can take, which influences the signal strength. When the signal passes through objects of different materials, it is *attenuated* by the object. This causes the signal to be slightly absorbed. It may also be partially reflected by the object into different directions. This leads to an observable decreased signal strength at the receiving antenna.

Using certain radio tomographic *models*, we make assumptions about how the signal is attenuated. The signal strength measurements can then reveal some information about the dimensions and the types of the objects that it crosses. We however need many links to determine where those objects are located. Additionally, reflection caused by the objects as well as other types of attenuation may lead to noisy measurements, which we need to account for as well [21].

The *reconstruction* that we perform in this way allows us to create an image displaying the objects that are located within the network. These reconstruction methods are a part of an additional collection of techniques, known as *radio tomographic imaging* (RTI). Various algorithms that are used in RTI contribute toward reconstructing the measurements into a smooth image of the detected objects [36].

Thus, we can use RTI to create such cross-section images, without any other information than the signal strengths. However, one limitation of current radio tomography solutions is that they require a large number of wireless sensors that remain in a fixed location. This means that it is difficult to deploy a sensor network in any random location.

We propose a new approach to the problem of gathering wireless sensor measurements that overcomes these limitations. We call this version *mobile radio tomography.* This is a collection of adapted reconstruction methods, and additional algorithms to make it stable. The intention is that it can be easily deployed in different settings without much prior knowledge about the objects within the network as well as the surroundings.

1.1 Problem statement

The main goal of our research is to make it possible to use radio tomography in other contexts compared to previous work. We want to find out whether it is feasible to use a network with fewer sensors. Instead of placing these at fixed positions which we do not alter during the measurements, we move the sensors around to measure different links.

This simple concept does need some more elaboration before we can actually tackle the reconstruction problem. A dynamic approach also has a number of interesting subproblems on its own. This includes the question of how to move those sensors around without disturbing the measurements themselves. We also need to ensure that the sensors are placed at correct and useful locations. Another issue is to make the collection of measurements fast enough so that the objects do not need to stand still for an impractically long time. This also means that we want to receive a reconstructed image relatively quickly, even if it is only a partial result with missing measurements.

A major novelty within this subject consists of the principle of moving the sensors around. In this thesis, we primarily focus on this essential part of mobile radio tomography. To make the placement of sensors precise enough and to remove most of the manual labor during the measurements, we require the use of *autonomous vehicles*, especially those that are completely unmanned. These vehicles can take the wireless sensors along with them, collect measurements and send these to a remote station for further reconstruction.

The use of autonomously moving vehicles brings some additional complications. We need to ensure that the vehicles do not collide with objects in the network or with each other. The vehicles should stay close to the network and not lose track of their position.

In order to increase the efficiency of mobile radio tomography, we should determine how the vehicles can gather many sensor links as quickly as possible. This means that there could be an optimal route for each vehicle. Finding such a route is a problem on its own.

In Section 1.1.1, we further explain the reasoning behind the mobile radio tomography concept, and justify why this is an interesting and novel principle. Section 1.1.2 builds further upon this by providing possible applications where mobile radio tomography can help and solve practical problems.

1.1.1 Motivation

Radio tomography is a relatively new field of science, but it makes use of well-established algorithms and intuitive tomographic models. It is related to other well-known techniques, such as radio propagation and telecommunications studies. It also has close similarities with certain medical specialities, such as radiology and radiation therapy.

In particular, radio tomographic imaging shares core properties with medical imaging processes, including computed tomography (CT) and magnetic resonance imaging (MRI). The foundations of these techniques have a common principle, namely that some form of radiation is emitted by a source. These waves of energy are absorbed, amplified or otherwise attenuated by some intermediate object of interest. The resulting signal is then measured at a collector, and combined with many other sensor readings to reconstruct the (internal) structure of the objects.

Radio tomography is located at an intersection of multiple principal fields of science. It uses physics to describe the effects of radio frequency energy at an atomic level, and defines fundamental models that make the reconstruction possible using mathematical methods. We then put computer science into practice to actually perform the reconstruction in a reasonable time, and wrap everything together. The full RTI technique can then be embedded into other fields, such as biology, medicine, archaeology, paleontology, geology, engineering and social sciences.

There are thus many uses for radio tomography, and it should become accessible to end users, without requiring them to have an in-depth knowledge of how RTI works. Radio tomography also provides many research opportunities, since it can be constantly improved through interpretation of existing results and new experiments.

Mobile radio tomography adds to the usability of this technique, since it requires far fewer sensors and can be applied in more situations. It also includes certain interesting areas of computer science, such as vehicle movement dynamics, artificial intelligence and combinatorial optimization of path routing in graphs.

1.1.2 Applications

We foresee multiple potential settings where mobile radio tomography would work better than other solutions. Since normal radio tomography requires one to build a network of sensors, whereas the mobile version should be quick to deploy, the latter may perform well in situations where time is critical.

One realistic example is the case of a building that is on fire. The local fire department may deem it to be unsafe to send in firefighters when the structural integrity of the complex is uncertain. On the other hand, there might still be people inside, who may be unconscious or otherwise unable to make their presence known. To add insult to injury, the doors may be locked, and it would take too long to break into all rooms.

Thus, while starting the extinguishing, the fire department could make use of robotic vehicles that either move around the premises or drive into the building, using radio frequency sensors to measure the objects and the unfortunate individuals. The reconstructed image can be compared to a floor plan to make a thoughtful decision of whether there are people to be saved and to risk the firemen's lives.

Other public services may also have applications for autonomous remote detection. The vehicles could be sent to a dangerous location to perform surveillance or reconnaissance missions. Mobile radio tomography can be used to determine how many people are inside an enclosed building during a hostage situation or when a crowd's size needs to be controlled. It may also be useful for bomb disposal actions, such as to determine whether a suspicious item has electronic, explosive or moving parts inside it, or that it is a fake.

Robotics are also increasingly used in healthcare facilities to provide resources and entertainment to patients and elderly people. Robotic vehicles could also observe that someone trips or falls out of a bed, and alarm staff appropriately.

Conventional techniques that are used in the cases described here often invade someone's privacy. Radio tomography, on the other hand, does not make a distinction between people, who appear as "blobs" within the reconstructed image. This makes it possible to use it in public places, such as warehouses or festivals. Also, the wireless radio frequencies are not too intrusive or harmful. Thus, RTI is safe to use in these circumstances.

1.2 Approach

We set up a project dedicated to mobile radio tomography. Within this project, we investigate the possible steps that we should take, implement a toolchain and perform experiments. In the end, this leads to a fully functional and well-tested result.

The project is split up into multiple phases: planning the missions, operating unmanned vehicles [19], communicating between the wireless sensors [26], the radio tomographic reconstruction and the final visualization [27]. Figure 2 provides a high-level overview of the phases in this project.

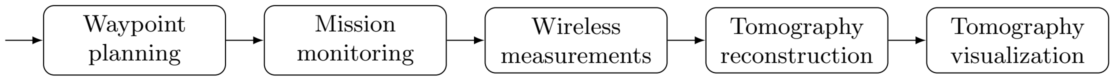


Figure 2: Flow diagram of the high-level phases of the mobile radio tomography project.

This thesis focuses on the first two phases, related to autonomous vehicle movement. These involve planning the (order of) positions where we perform tomographic measurements, and letting the vehicles automatically visit these positions without any problems. In our research, we determine our requirements, create a model of the subdomain in which we wish to solve these problems, investigate existing algorithms, implement a software toolchain that plans and monitors missions, and experiment with the entire setup.

Our main findings of this approach are that the use of search algorithms, collision avoidance and other planning algorithms can be helpful tools to augment manually created routes, and that the use of vehicles for the purpose of radio tomography is a viable strategy. These highlights form the main goal of this thesis.

1.2.1 Team

In order to investigate all the fields related to the mobile radio tomography project and to divide the tasks among the people responsible for it, we formed a research group consisting of members from Leiden University as well as CWI Amsterdam.

The group consists of the following members, in alphabetical order:

- Joost Batenburg (CWI Amsterdam): Supervisor, diverse knowledge of tomography theory and radio tomography imaging (RTI)
- Folkert Bleichrodt (CWI Amsterdam): Researcher in RTI and interests in embedded hardware and rovers
- Leon Helwerda (Leiden University): MSc student, focus on autonomous missions and vehicle planning
- Walter Kosters (Leiden University): Supervisor, diverse knowledge in the field of artificial intelligence
- Tim van der Meij (Leiden University): MSc student, focus on ZigBee packet stream and reconstruction visualization

The group is frequently assisted by Willem Jan Palenstijn, Daniel Pelt, Zhichao Zhong and Xiaodong Zhuge, all from CWI Amsterdam. They supply useful feedback on the theoretical basis of tomography, antenna properties and other research related to unmanned vehicles. We also appreciate the suggestions from Alyssa Milburn. Her help allows our research to make use of, and continue with earlier projects [19, 25, 26, 29].

This master's thesis is made in association with the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University, and the Centrum Wiskunde & Informatica (CWI) at Amsterdam, under the supervision of Walter Kosters and Joost Batenburg.

1.3 Overview

The remainder of this thesis is built up as follows. In Section 2, we perform a literature study concerning existing algorithms and implementations that can help within the context of mobile radio tomography. Section 3 provides definitions for many concepts related to autonomous vehicles within an environment where we can perform tomographic sensor measurements. Then, in Section 4, we describe various algorithms that were adapted or specifically created for planning missions and operating vehicles autonomously. Section 5 puts the definitions and algorithms into a physical perspective, making it possible to use them in reality.

We proceed with the introduction of the mobile radio tomography toolchain in Section 6, which describes how we implement the necessary components in a structured manner. Section 7 proposes experiments that we perform with this toolchain, and provides the results found in this way. Finally, we conclude in Section 8 with some remarks about the usefulness of the techniques, and state potential future work in Section 8.1.

2 Related work

To increase our knowledge and understanding of the problems that we encounter when we create a mobile radio tomography toolchain, we give an overview of some related literature.

For more information on the principles of the techniques related to radio tomography, we refer to other papers that focus on the tomographic measurement collection, reconstruction and imaging problems [25, 36, 21]. In particular, there are some recent investigations into making radio tomography more versatile by rotating sensors [35], moving them for short distances [20], or placing them in exterior environments [2].

In the remainder of this section, we focus on algorithms, equipment and other techniques that allow us to really make a mobile variant of radio tomography, using vehicles that move the wireless sensors around. In Section 2.1, we describe existing algorithms that can help us plan and optimize the routes that the vehicles take. Section 2.2 shows some ways how we can make the vehicles move around autonomously, where artificial intelligence (AI) and swarm communication play a role. We also discuss the benefits and disadvantages of certain autopilot hardware that can control the propulsion of a vehicle.

2.1 Routing algorithms

It is a well-known problem to find an optimal path between certain locations, using a restricted set of possible moves. This problem has its roots in AI, where the task is to improve the path that robots take so that they reach their goal as fast as possible.

There are many variations of this basic problem. These may include additional constraints, such as only allowing movement between certain pairs of locations or including penalties for certain paths. Depending on the specific problem that we want to solve, we may use an algorithm that is adapted for this purpose.

A family of algorithms that is often used to solve these problems, is the group of *search algorithms*. These routines accept *graphs* as input. Such a graph describes the possible connections between discrete positions. A search algorithm follows the paths between those positions in the graph, in order to construct a route from one point to another, improving it whenever possible. We investigate the search algorithms further in Section 4.1.

Another important problem in the field of combinatorial optimization is the *traveling salesman problem* (TSP). In this case, each position has a connection to another, but the connections can have different distances that add up to the length of the route. The goal is to find the shortest route that visits each position (exactly) once. We can add more restrictions, such as penalties for “turning” in a certain direction, or having to take a detour, by augmenting the distance weights [37].

A specialized case is the traveling salesman problem for multiple agents. Here, we have more than one vehicle that needs to visit certain positions. Possibly, the vehicles all have the same set of positions that are to be visited by one of them. They could also have their own sets, where the goal is to optimize each distinct TSP subproblem [9].

We increase the complexity of the problem by requiring each vehicle to visit a certain position at the same time that another vehicle visits a different position, such that they can perform measurements between each other. These synchronization problems are also studied independently, and are related to the use of swarms of agents that cooperate to solve a problem [11].

There exist various algorithms that attempt to solve TSP and related vehicle routing problems [23], known as *TSP solvers* [18]. They may use various combinatorial optimization algorithms or local minima gradient searches to improve their solution.

A different type of optimization methods are the *evolutionary algorithms* [6]. Here, we use not just one solution that we improve, but we have a *population* of multiple individuals, consisting of variables that encode a potential solution. The values of those variables can be mutated to form a new individual, which is then compared to the individuals in the existing population using a *fitness* function. This evaluation allows us to select the worst individual, which is then removed from the whole population.

The family of evolutionary algorithms includes a variant which is known as *multiobjective optimization algorithms* [13]. Here, we have multiple fitness functions, known as objectives, and also have constraints that denote that a certain individual cannot result in a correct solution. The use of more objectives allow us to improve in multiple domains, e.g., not only decreasing the length of the routes, but also enhancing other derivative properties. We continue our research of these evolutionary algorithms in Section 4.2.3.

2.2 Autonomous vehicles

Recently, there is much interest into self-driving cars. Most of the research concerns large-scale automobiles that can carry passengers while reducing the need for them to take action during their normal mode of operation. Such vehicles may even detect dangerous situations and avoid as much damage as possible. Additional problems are routing a path to certain locations and adhering to all regulations, such as speed limits and traffic signals.

Our research also focuses on autonomous vehicles, but these are of a different kind. We do not intend to operate the vehicles on public roads, especially not in the case of certain types of vehicles that are not allowed nearby such zones, including drones.

The vehicles do not have to carry a pilot or passengers, thus they can be smaller in size. We do need to carry around sensors and remain stable, also when we stand still. Finally, we do need to keep track of other unmanned vehicles that we should avoid, and cooperate with them to solve the tasks at hand.

This requires advanced artificial intelligence to model and implement the robotic movement patterns that the vehicles make use of. Such models have a intricate theoretical foundation, which includes concepts such as the degree of freedom of robotic moves, the discrete representation of the real world within the robot's memory, and other filters that make use of sensors [33].

We can also model the way that different vehicles interact as if they are a *swarm*, a group of individual robots that attempt to reach the same goal, either by preprogrammed behavior or through dynamic communication or sensor detection [14]. This communication also improves the safety of the arrangement of vehicles, since they can tell more easily where the other vehicles are and avoid them while moving.

Finally, we look into existing hardware and software packages that could aid us in solving the problems that we describe. We keep in mind that we want to support as many different types of vehicles as possible, so that the complete toolchain can be used in many contexts. This includes vehicles that can fly and those that operate on the ground [31].

We can make use of the MAVLink communication protocol, which supports these types of vehicles [24]. The DroneKit software package allows one to write programs that make use of this interface, so that they can work with various vehicles, but specifically drones and other miniature plane-like aircraft [1]. Several hardware autopilots can be controlled in this way, including the ArduPilot and Navio+ microcontrollers [5, 15].

3 Definitions

In this section, we provide definitions for the concepts related to autonomous vehicle movement and radio tomography. These definitions lay down the theoretical foundations related to the main problem of visiting sensor positions using vehicles to the benefit of a tomographic reconstruction. This allows us to express the problems, challenges and results in a clearly defined manner. This removes ambiguity about the terms, thus helping in making the goals more concrete and insightful.

Before defining a complicated model for sensor positioning in radio tomography, we first need to realize that any such model needs to take place in some kind of *space*. This space can be defined as some mathematical collection from which we can draw different *points*. The space may contain any number of points, or in fact have infinitely many points.

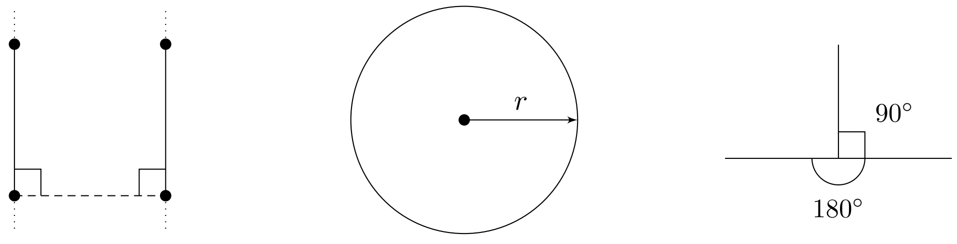
The space is the basis of the model in which we define our autonomously moving vehicles that collect tomography measurements. We alter and constrain the space according to certain rules in Section 3.1, fill it with an environment of physical objects in Section 3.2 and give some of those items certain roles, such as the vehicles themselves in Section 3.3, and their sensing features in Section 3.4. This then allows us to define the missions and goals in a formal manner in Section 3.5.

3.1 Geometry

The space and the points it contains have certain properties that follow a set of rules included in a *geometry*. We define a geometry as a system of axiomatic rule statements. Such a geometry augments our model of the space, allowing the model to behave similar to the way the physical world works. This holds at least up to a certain level of precision and in certain regions of the space.

The geometric model is thus not exactly equal to the physical world, but it is similar enough to work well for our purposes. Through the use of propositions, we can define the shape and density of the space, thus constraining its cloud of points.

One type of geometry in this sense is Euclidean geometry [12]. This axiomatic system defines five postulates that describe the properties of the geometry. The postulates bind together points, allowing other structures such as lines and circles to be defined. The postulates are basic necessities for this type of geometry; if a statement follows from the postulates, then that statement is not a postulate.



(a) Postulates 1, 2, 5: lines (b) Postulate 3: circle radius (c) Postulate 4: angles

Figure 3: Visual representations of the postulates of Euclidean geometry.

In Figure 3, we show some direct results of the postulates. The postulates themselves can be paraphrased as follows:

1. We connect points with each other using straight *lines* or segments of finite length.
2. Such lines can be extended continuously, as long as they remain finite.
3. We describe a *circle* uniquely by its center point and a *scalar radius.*
4. When two lines intersect, then there is an *angle* between the two lines. Lines that extend in opposing directions yield *straight angles*, and a line exactly in between those two directions makes *right angles* with them. This is summarized in the statement that right angles are equal to each other.
5. Finally, according to the parallel postulate, two line segments, when we extend them continuously, must at some point intersect with each other when the angle between them and a line connecting their start points, the *base*, is less than a straight angle.

3.1.1 Coordinate system

Within the geometry of Section 3.1, we can define a *coordinate system.* Each point in our space receives a unique identifier in this system. This identifier, known as a *coordinate tuple* or simply *coordinates*, is an ordered, fixed-length list of numerical scalars. Each element of this list is a value from the set of real numbers $\mathbb{R}$.

Such a *coordinate value* represents the length of a line segment between the point and another point at which the line segment makes a right angle with a certain line or a *surface* defined by multiple lines. A surface is a plane in the space at which each point in the surface has similar line segments that make right angles with every line in the surface.

These lines must be a part of the set of *axes.* The lines in this set are at right angles with each other. The intersection of these axes is at one and the same point, which is the *origin* of the space. The axes extend continuously in all directions. A coordinate value of a certain point can then be defined as the length between the point and a certain surface of the axes, excluding the axis line related to the coordinate value in question.

The number of axes is the *dimensionality* of the system. Often, a space is three-dimensional, which gives us coordinates of the form $(e_0, e_1, e_2) \in \mathbb{R}^3$. When the dimensionality is 2, the coordinate value of one axis is easier to determine than the plane construction: the value is equal to the length of a line to the other axis which is at a right angle with that axis. The coordinate values also determine the distance or *norm* between points: the generic notation $\|v\|$, for a given point $v \in \mathbb{R}^3$ from our space, means the norm used within the space, of which we show examples in Equations 3.1 and 3.2.

The use of real numbers for coordinate values allows us to have infinitely many points. Thus, a line can keep on extending, and it can always end at a point. Also, the definition of *distance* between two points (p_0, p_1, p_2) and (q_0, q_1, q_2) in our space is given using the L^2 norm:

$$\|(p_0, p_1, p_2) - (q_0, q_1, q_2)\|_2 = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2 + (p_2 - q_2)^2} \tag{3.1}$$

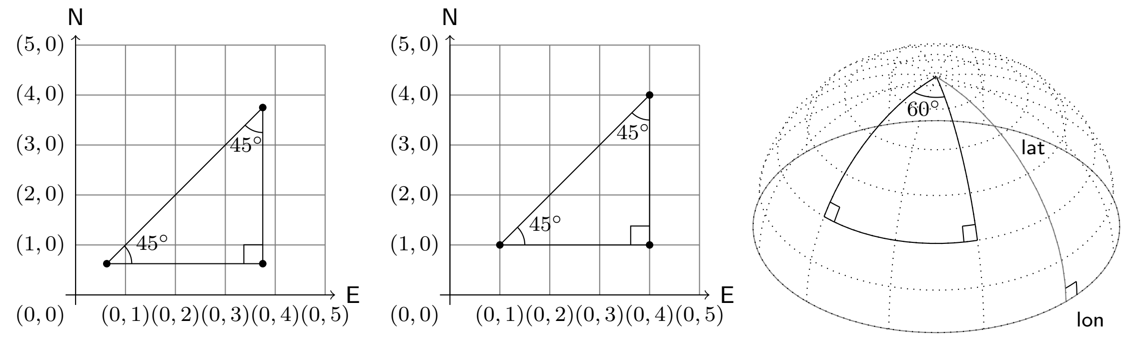
Here, we make use of elementwise subtraction to be able to handle the coordinate tuples.

We can constrain the coordinate system to only describe the discrete points of the space. For example, we restrict the coordinate values to only use integers from $\mathbb{Z}$. This gives us a *grid* space where we still have continuous line segments, they just do not always intersect at points with coordinates. The grid distance can also be defined as the L^1 norm or *Manhattan norm*:

$$\|(p_0, p_1, p_2) - (q_0, q_1, q_2)\|_1 = |p_0 - q_0| + |p_1 - q_1| + |p_2 - q_2| \tag{3.2}$$

A different type of geometry is *spherical* geometry. Here, lines may intersect even when they are at right angles with the base. The coordinate system and its associated norm do not use straight line distances. Instead, they are based on curves around a sphere at the same north-south and east-west parallel lines. The upward axis is warped, so that it is at right angles of these lines at every point.

At a local level, the spherical geometry behaves much like the other geometries, but for greater distance ranges it can be an approximative model of the Earth's sphere *geoid* [17]. This means that spherical geometry is a more applicable model for the physical world at a global level when navigating a larger distance.



(a) Normal Euclidean geometry (b) Grid geometry (c) Spherical geometry

Figure 4: Overview of differences between certain geometries in the first two dimensions, such as bounds on the usable points and significance of right angles.

Many of the similarities and distinctions between the geometries listed here can also be represented in a visual manner. We outline some of these characteristics in Figure 4. In Figure 4a, the first two dimensions of normal Euclidean geometry are shown, where any point with real-valued coordinate values is allowable.

Figure 4b shows grid geometry, where we only consider points on specified grid lines. All other properties from Euclidean geometry, such as right angles, still hold. Not each part of the line is actually a point that we can describe in the coordinate system, but this does not invalidate the line segment itself.

Finally, Figure 4c shows the curved space of spherical geometry. Note how right angles might still lead to lines that intersect, unlike in other geometries where this is axiomatically true. Distances between points follow the *great circle distance* [12].

3.1.2 Locations

The points in our geometry define *locations* at which events can take place or objects can be situated. We often describe locations by the coordinates of the points that they inhabit.

Locations are distinct from points, because we overlay the coordinate system after defining points in our space. The locations are thus a restricted subset of all predefined points; not every point may be a location.

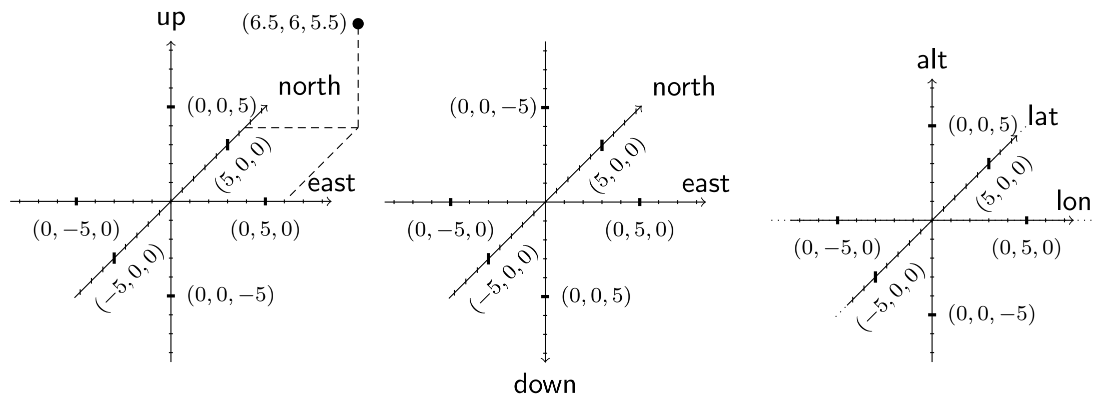
As we have seen in Section 3.1, there exist various geometries and coordinate systems. Consequently, a location can also be defined in multiple ways. There are more differences between the coordinate systems than we discuss in Section 3.1.1. The differences mentioned in this section mostly relate to the meaning of the axes in the system, which are represented in Figure 5.

A Euclidean geometry usually has three dimensions, respectively labeled as *north*, *east* and *up* axes. These axes extend into the corresponding intuitive directions as shown in Figure 5a. The axes are also often given shorthands. In our case, we respectively call them the y, x and z axes.

Although an ordering of (x, y, z) would be more aesthetically pleasing, the (y, x, z) order stems from the two-dimensional coordinate system. Usually, the y axis is directed to the top and x rightward on a flat surface. The z axis just adds a third dimension. We then have coordinates of the form $(e_y, e_x, e_z) \in \mathbb{R}^3$.

Note that the names and order of the axes may differ in systems used in other fields, such as aeronautics and flight dynamics [34], or completely different applications such as computerized imaging.

A grid geometry is similar to the normal Euclidean geometry, with coordinates from the Cartesian product $\mathbb{Z}^3$. If the grid is two-dimensional, then we leave out the z component. However, when we do use the z component, then it is actually a *down* axis extending downward, rather than extending upward in the space. This means that in comparison to the normal Euclidean geometry, the final coordinate value e_z becomes its additive inverse $-e_z$. The effect of this operation on the axes can be witnessed in Figure 5b.



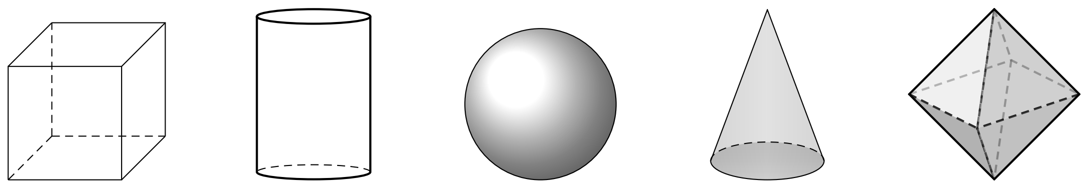
(a) Normal Euclidean geometry (b) Geometry with down axis (c) Spherical geometry

Figure 5: Axes of the three-dimensional coordinate systems in certain geometries. There are differences in axis directions.

The spherical geometry in Figure 5c also has coordinates with three components, named after *latitude*, *longitude* and *altitude* [30]. The latitude and longitude axes curve around a spheroid, while the altitude extends upward perpendicular to the current point. This means that the direction of the upward "axis" changes along with the curvature of the first two axes.

3.2 Objects

As mentioned in Section 3.1.2, a location may be the host of an object or an event which is directly caused by such an object. An *object* is a unique entity that exists within the space. Objects come in various types and shapes. They are physical objects, meaning that no other object can coincide with it at the same time.



(a) Cube (b) Cylinder (c) Sphere (d) Cone (e) Other polygon

Figure 6: Examples of various object shapes that exist within the space.

Large objects can take up multiple locations, and they may not be constrained by, e.g., grid coordinates. We can model them as cubes, spheres, boxes, cones and cylinders, to give a few examples. We can also construct objects from combinations of these shapes. We transform the object by placing the shapes at specific translation coordinates from their center location. Figure 6 shows some of these shapes.

Another way to define objects is through the use of *polygons*, which are cut-outs from *planes* in the space. A polygon can be described by an ordered sequence of coordinates $P = p_1, p_2, \ldots, p_m$. The *edges* of the polygon are the line segments between each point and its successor $(p_i, p_i + 1)$ for $i < m$, as well as the final point and the first point in the sequence (p_m, p_1). The polygon itself is the area within these m edges. For coherency, we only consider polygons that do not have edges that intersect with each other.

An object can then be defined as a set of polygons $\{P_1, P_2, \ldots, P_n\}$. Note that spherical or other circular shapes cannot be defined with a finite number of polygons. We can create a large subset of possible objects using polygons, such as the object in Figure 6e.

We use polygons to model real-world objects. We can often leave out some of the large amount of details, since these do not influence the large-scale distances between objects.

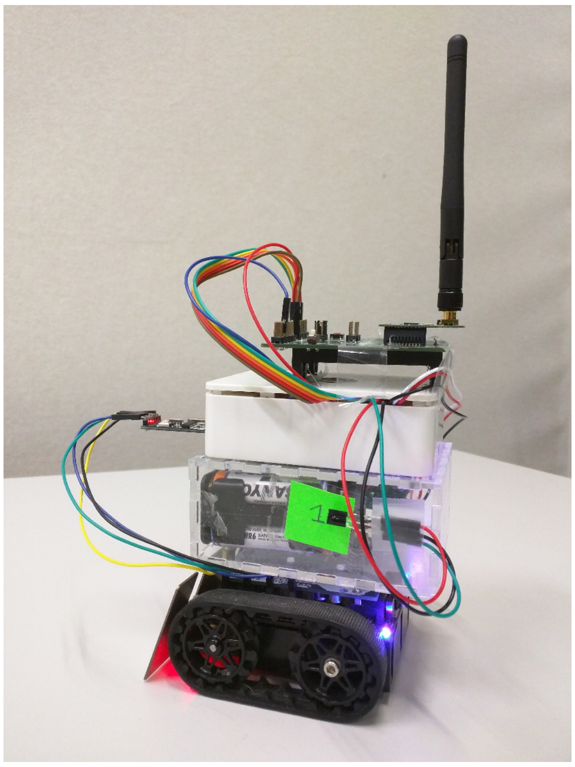
3.3 Vehicles

One specific type of object within the space is a *vehicle*. Unlike most objects that we introduce in Section 3.2, a vehicle can move in the space defined by our model. This allows it to change its current location and orientation. Thus, the vehicle has properties that define its current *state*.

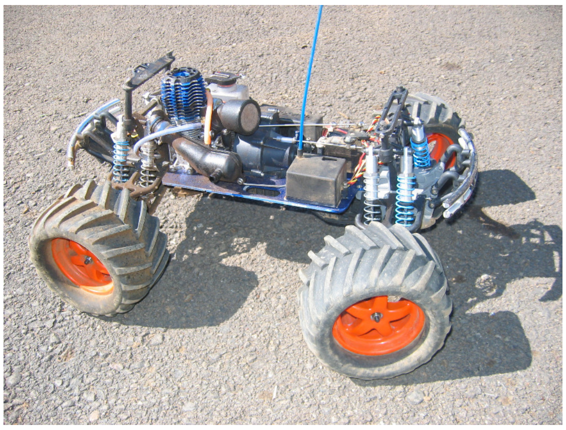
The orientation or *attitude* of the vehicle defines the rotational transformation applied on the object, i.e., the angle direction that the front end of the vehicle is pointing to, compared to each axis.

Often, the attitude axes are called *pitch*, *roll* and *yaw*, corresponding to rotations around the axes in the coordinate system, instead of using the various coordinate axis names of the geometries directly. This is because we may consider the attitude axes inside a local frame, after applying transformations and attitudes of parent objects. This plays a role in the sensors that may depend on multiple attitudes, as described in Section 3.4.

The vehicles are under the control of the creator of the model, so we are able to choose how it moves. There are however constraints to its movement patterns, determined by properties of the vehicle. The vehicle cannot just change its location to any other location instantaneously. Instead, it has a variable but limited *speed* at which we are able to move to other locations. The speed is a vector of components for every axis. The speed values are measured in coordinate values per time unit.



(a) Caterpillar track robot



(b) Rover with wheels

(c) Flying quadcopter drone

Figure 7: Physical vehicle types

Vehicles have different types. We consider a number of vehicle types, some of which are depicted in Figure 7. If we have a land vehicle such as a caterpillar track shown in Figure 7a, a bipedalic robot, or a rover with wheels in Figure 7b, then it always moves on a *ground* surface in the space. It cannot change the altitude or down component of its initial location.

Other vehicles might be able to fly, such as the drone in Figure 7c. A flying vehicle may still be affected by *gravity*, an acceleration property of the space that attempts to increase downward speed.

Depending on the vehicle type, the possible values of the speed vector may depend on the vehicle's attitude. For example, a vehicle with wheels can only move forward or turn around a certain turning radius. Tracked wheels can operate faster, but are more limited in turning.

In addition to the speed limitations and external influences, there are also constant physical constraints. Vehicles cannot move through other objects, including vehicles; they can only change the coordinates of their own parts.

Although the vehicles are under our control, we may provide the vehicle with restricted knowledge about the space it operates in. This may mean that the vehicle knows details about the bounded size and geometry of the space, but it cannot make any assumptions about the objects and their locations in order to avoid them more easily. In Section 5, we not only restrict this knowledge, but factor in uncertainty of information.

3.4 Sensors

Our model contains *sensors* that detect other parts of the model and interact with them. There are various types of sensors, but we first look at sensors that are of use for radio tomography. These *tomographic sensors* are able to send and/or receive signals that are attenuated by objects in between them, resulting in a detectable weaker or stronger signal. They can thus have a sender or receiver *role*. They can also have both roles at the same time or cycle between the two transmitter roles over time, depending on the type of sensor.

All sensors are objects, or are a part of them. A distinguishing feature of sensors is that they can be *static* or *dynamic*. A sensor that is part of a vehicle as described in Section 3.2 is called dynamic, i.e., its location can change. Otherwise, the sensor object is static. Moving a vehicle thus changes the point at which a dynamic sensor is at as well. For some sensors, we may only be interested in locations where the vehicle's speed is zero, or where the vehicle is at an actual grid location. The sensor can receive feedback about the vehicle state, as well as its own conditions.

The state of a tomographic sensor can be *active* or *passive*. In the active mode, the sensor performs signal measurements with other sensors. Through these measurements, it may learn the locations of those sensors and the signal strength of the link. In passive mode, the sensor can also send or receive packets of limited information, but the communication can contain other data than that which it sends in active mode.

Aside from tomographic sensors, there also exist other types of sensors that do not interfere with the radio tomography. One of these sensors is the *infrared sensor*, which can receive a small-sized command from the creator of the model at any time.

Another type of sensor is the *distance sensor*. This sensor detects nearby objects. In case there is an object in a line segment starting from the sensor position in the current attitude, the distance sensor determines the intersection point of the line at the object, which is the first point that is a part of the object. It then provides the distance to the detected point, using the L^2 norm from Equation 3.1.

These sensors are part of vehicles, thus they inherit their location and attitude angles. However, it is possible to change this inheritance by associating a *servo* to the distance sensor, for example. The servo has a certain angle range in which it can freely rotate. The servo always rotates in a certain attitude axis relative to the vehicle. The servo's current angle is added to this axis to result in the sensor's current attitude.

There are also other sensors that allow a vehicle to know more about its environment and current situation, such as location sensing. Also, the vehicle may have a clock that sets a time limit on the vehicle's lifetime. For example, a battery limits the amount of time that the vehicle can move and make use of its sensors. A battery monitor can be used to know how long this takes. These sensors are put in a more practical context in Section 5.

3.5 Missions

A vehicle may have a *mission* that determines what it should do inside the space during its lifetime. This mission can be defined in different ways, including specifying the vehicle's speed at each time unit.

We can describe a mission in a simple but still powerful form using *waypoints*. The waypoints are an ordered sequence of locations that the vehicle should visit, in that order. The way the vehicle reaches those locations does not need to be specified, in which case it should attempt to reach them in the fastest and safest way possible.

This gives some freedom to adjust the mission while the vehicle is following it, for example to avoid collisions with other vehicles or modeled objects that are not explicitly known to the mission beforehand. The mission can receive online adjustments to add additional intermediate waypoints, or remove ones that end up to be unreachable.

Waypoints may also let the vehicle wait at a specific location in order to let the sensor perform measurements. This can allow us to delay the continuation of the mission until we have successfully exchanged signals with at least a specific set S of sensors, determined by unique identifiers.

Using this construction, the vehicles can operate in the space using missions to perform measurements with their tomographic sensors at specific locations. The resulting set P of *sensor positions* contains information about the measurements, most importantly pairs of locations. In addition, a sensor position may have a time unit in order to provide unique timestamps to the measurements.

A *measurement* (E, F) from location E to F is inside the set P if and only if there are active tomographic sensors at both locations at the same time, and the sensor at E sends a signal that the sensor at F successfully receives. This means that, in order to add more viable sensor positions to P, we can alter the missions of the vehicles in such a way that they visit more locations and synchronize with each other. If both sensors can send and receive, then we obtain both measurements (E, F) and (F, E) in the sensor positions P.

3.5.1 Safe paths

The vehicles from Section 3.3 can move around, but they cannot move through objects. Instead, they collide with objects, which might have serious real-world implications, such as damaging the objects and causing the vehicle to malfunction. In Section 5, we take these implications into consideration. At the very least, the missions from Section 3.5 could be considered *failed* when they are obstructed, since they are unable to finish visiting all points that they should have.

Obviously, the model from Section 3.2 gives us the freedom to define objects in such a way that a predetermined mission never collides with them. The same applies to avoiding collisions with other vehicles. We can simply make sure that the missions never send vehicles to the same point at the same time, taking the size of the vehicle's object into account, with a suitable padding for the *region of influence* of the object.

In some missions, as well as in the practical world mentioned in Section 5, we might not know beforehand where each vehicle is located at every time unit of each vehicle's lifetime. For example, defining the missions as waypoints means the vehicles have a degree of freedom of moving around to such a waypoint. We could restrict this freedom to ensure that there are no concurrent intersections between the vehicles' waypoints. This also restricts the possible missions that we can have.

Another option is to remove such restrictions, and avoid impending collisions during the mission, shortly before they are about to happen. We can use the distance sensor from Section 3.4 to detect any vehicles or objects that are in the way, or even use the location information from an active sensor to determine other object's behavior.

Using this proximity information, we can then stop the vehicle, which should prevent the collision if all vehicles behave in this way. However, this strategy does not ensure that a vehicle can continue later on. This means that the mission might still fail.

When we detect an object close to us, we want to find a way to get around this obstacle and leave its region of influence, apart from the passive solution of waiting or stopping until the other object is gone. However, finding such an escape route may be nontrivial, or in some cases, impossible. A *safe path* is a sequence of locations that does not enter any known regions of influence that are caused by other objects or vehicles. In Section 4.1, we discuss the details of various methods of finding safe paths.

4 Algorithms

Using the definitions that we state in Section 3, we can use a higher level of abstraction for modeling our problems and their solutions as search spaces and algorithms, respectively.

We outline some of the problems related to autonomous vehicle movement and sensor position planning in this section. In Section 4.1, we delve into the functionality of existing search algorithms, and describe them in a formal manner using our model. We also describe a new problem that needs a similar but novel kind of algorithm to solve. In Section 4.2, we return to the problem of finding as many sensor positions as possible within certain limitations, and describe a class of evolutionary algorithms that might give approximate solutions to this problem in Section 4.2.3.

We propose new algorithms that solve specific problems in a specialized manner. The same applies to the problems themselves, which may either be well-known in other fields of science, or novel approaches within the topics of radio tomography.

4.1 Search algorithm

There is a need of actively avoiding collisions with objects and vehicles to allow completing missions, as seen in Section 3.5.1. This requires an algorithm that searches for a new, safe path to the next waypoint. Such a *search algorithm* can be any program that, given the current location and target location, as well as any environmental information such as unsafe regions, returns a path that is safe according to the current information.

The safety of the given path cannot be guaranteed for the time that it takes to follow that path. We can detect previously unseen objects, or another vehicle may move onto the path. An obvious solution is to use the search algorithm again to find a new safe path, once an impeding object is detected.

The search algorithm might not provide an "optimal" safe path, neither in the sense of its total length nor in the probability that it remains safe. Of course, an algorithm that provides certain guarantees about its degree of optimality is helpful.

In the remainder of this section, we recite a number of search algorithms, demonstrate their similar structure as well as their differences, and explore the use of search algorithms in the context of our geometries and missions. The algorithms involved are breadth first search (BFS), Dijkstra, A*, iterative deepening and bidirectional search [33].

The algorithms mentioned here share the property that their input is usually a *graph*, where the possible locations are modeled as nodes with edges that indicate a direct connection between those points. The overall structure of these algorithms is shown in Algorithm 4.1.

Algorithm 4.1 Generic structure of most graph-based search algorithms.

Let $G = (V, E)$ be a graph with an operation to find a node's neighbors and an indication of the size of V. Let $s \in V$ be the starting node and $e \in V$ the end node.

```
1: procedure SEARCH($G$, $s$, $e$)
2:     initialize state variables and distance functions, if necessary
3:     allocate memory for paths, maps, sets or queues based on graph size
4:     while not all nodes have been visited do
5:         select $v^*$ from unvisited nodes according to a selection procedure, initially $s$
6:         if $v^* = e$ then
7:             return the (reconstructed) path from $s$ to $v^*$
8:         end if
9:         update state variables, mark $v^*$ expanded
10:        for each neighbor $v$ of $v^*$ according to $G$ do
11:            if $v$ has not been expanded before then
12:                add $v$ to the collection of unvisited nodes
13:                determine distance of the current path from $s$ to $v$
14:                if there is no earlier path from $s$ to $v$ with lower distance then
15:                    add edge $(v^*, v)$ to a path from $s$ to $v$, using the path from $s$ to $v^*$
16:                    update state variables, including distances
17:                end if
18:            end if
19:        end for
20:    end while
21:    return an empty sequence
22: end procedure
```

For certain geometries in Section 3.1, it may be difficult to describe the space as finite sets of nodes and edges. One exception is grid geometry, where it is trivial to convert points and grid lines to a graph.

Either way, we do not need to provide the search algorithm with the full graph in one go. Instead, when an algorithm takes a step of "expanding" a node, that is to say finding its incidental edges or following an edge, we can generate the next part of the graph on the fly. It is helpful to know the size of the space, the number of nodes, or bounds on each dimension's axis in advance, so that the search algorithm can allocate auxiliary memory.

Finally, we leave out nodes from the graph when they are too close to unsafe regions. This can also be determined just before expanding a node. Using this construction, we can describe the location points in the way they were meant to in Section 3.1.2, and only have an implicit graph during the algorithm.

We can also limit the number of edges of each node in our dynamic graph. We only consider a point to be a neighbor if it is the closest point to the current point in a certain direction. The neighbor must be in one of the *(inter)cardinal* directions, i.e., the axis directions and line segments that have *diagonal* angles, precisely between the right-angled axes.

When the search algorithm completely expands the end node e during its path creation from the starting node s, it can stop its search. We then have a safe path from s to e. However, some algorithms might not generate the fastest path possible. Even if they do, they might provide paths that never end up going to e quickly anyway. These differences often stems from the order in which an algorithm expands its nodes.

For example, BFS expands nodes in the order that it finds them, using a queue to expand nodes found earlier before newly expanded nodes. There is thus an implicit range search that slowly expands the search outward from s. There is no explicit relationship with the distance to e, thus the BFS algorithm is not a guided search method.

Another difference between the search algorithms is the degree of heuristics that they use to speed up the search. The A* search algorithm uses a tentative distance function $h(v)$, which calculates the norm between any given node v and e. The norm may be the one used in the space, for example Equation 3.1, or any other distance measure.

One condition for this heuristic is that distance function is *monotone*, i.e., $h(v)$ is not greater than the actual path distance to any intermediate node v' plus $h(v')$. Also, $h(e) = 0$. During the selection phase, the A* search algorithm expands an unvisited node whose distance from s plus the tentative distance to e appears to be minimal.

Some of these algorithms are designed for weighted graphs, where each edge indicates its distance. If all points are spread out in a grid-like manner, with a distance of c between points on one line, then this distance is c in the cardinal directions and $\sqrt{2c^2}$ in intercardinal directions when Equation 3.1 is used in normal geometry, for example. The distance weight might be helpful information for some search algorithms, e.g., Dijkstra's algorithm, but it does not necessarily lead to a better path or a faster algorithm.

In summary, we can use search algorithms to provide the vehicles with an option to avoid collisions and take a different action when it is inside the region of influence of another object. The safe paths that may be provided by a search algorithm may not be optimal, but are safe for a while and can bring the vehicle back to its original mission quickly.

4.2 Planning problem

While we may be able to use algorithms to find safe paths as shown in Section 4.1, another problem is whether we can design an algorithm that plans which waypoints we want to visit in our mission. Such a mission includes various waypoints, as mentioned in Section 3.5. At these waypoints, we can perform tomographic measurements. This helps in increasing the number of sensor positions, which is vital for making tomographic reconstruction possible in the first place, as well as improving its accuracy.

The ultimate goal is thus to find a mission that optimizes the set of sensor positions. This goal can be defined into more detail, as it can be split up into multiple, possibly conflicting, subgoals. This is because we do not only want to have more sensor positions, but also keep the mission duration as low as possible.

Other aims during the mission planning are to cover as much of the space of interest as possible, have many intersecting measurement links, decrease the length of the links themselves and improve reconstruction in certain areas of the space. These aims stem from the reconstruction algorithm [27], which has some preconditions to be able to function and can be improved by following these guidelines. These desirabilities directly influence the usefulness of the chosen mission.

We design different missions for multiple vehicles, operating in a swarm-based manner to move to the necessary sensor positions. When a certain measurement link (E, F) is desired, then one vehicle needs to be at position E, and another at F, at the same time.

It might be possible to plan the timing of the vehicles so that they are at those positions up to a certain degree, but we should still allow for dynamic adjustments during the mission, when safe paths from Section 3.5.1 need to be recalculated.

These goals and restrictions limit which missions are of use for the tomographic reconstruction, but also provides interesting challenges in allocating the work of moving and waiting at waypoints to the vehicles.

In Section 4.2.1, we delve into the problem of assigning a collection of requested sensor links to the available vehicles, and provide a greedy algorithm which is able to solve this problem. Section 4.2.2 augments this algorithm with a collision detection and avoidance algorithm. Finally, in Section 4.2.3, we introduce a family of evolutionary algorithms that can help in optimizing multiple goals, and provide an overview of the complete algorithm that attempts to optimize the sensor link positions for our purposes.

4.2.1 Sensor position assignment

As mentioned in Section 3.5, we can describe a mission by the sequence of waypoints that it should visit. Usually, this sequence of locations is already ordered when we provide it to the vehicle, so that it consecutively moves from A to B, and then to location C, if those are provided in that order. However, what do we need to do if we already have the sequence of waypoints we want to visit, but not yet the order in which to visit?

The problem statement here implies that we receive the sequence as if it were an unordered set. This means that there is probably no need to follow any specific order. Some orderings may be helpful for the tomographic reconstruction, but this is not the issue at this point.

Instead, we can rearrange the waypoints such that the mission takes less time, by visiting waypoints closer to each other before moving on to distant points.

This principle is similar to the traveling salesman problem (TSP), where we want to visit each node of a graph exactly once. Each edge in that graph has a *weight*, a numeric value indicating the distance between the nodes. We can then optimize the selected path to have the smallest total distance. This structure also shares similarities with the search algorithm's graph from Section 4.1.

In our case, the input is not just a sequence of waypoints for one given vehicle, but a set of sensor points. In fact, we have *pairs* of such sensor locations, originating from predetermined sensor positions that make up measurement links. Thus we cannot simply use a TSP solving algorithm to find the shortest path, since a vehicle cannot be at both endpoints to perform a measurement.

On the other hand, the additional constraints related to synchronizing those pairs of waypoints pave the way for us to solve the problem using a simple algorithm. Before we demonstrate this algorithm, we discuss what input it receives, and explain some consequences of the problem statement.

Assume that the vehicles are each at a given *starting location*, i.e., the location where they are placed before the mission begins. Consider the vehicles to be elements from the set $V = \{v_1, v_2, \ldots, v_n\}$, where n is the number of vehicles at our disposal. Their starting locations are grouped in a sequence of coordinate tuples $S_1, \ldots, S_n$, where S_i is the location of vehicle v_i. Later on, we update these locations to track the *current location* based on which waypoints the algorithm assigns to the vehicle.

The sensor positions are given as a set of pairs

$$P = \{(p_{1,1}, p_{1,2}), (p_{2,1}, p_{2,2}), \ldots, (p_{m,1}, p_{m,2})\}, \tag{4.1}$$

with m desired measurement links in total.

Then we determine U, a specific subset of the Cartesian product of the vehicles:

$$\begin{aligned} U &= \{(u, v) \mid u \in V, v \in V, u \neq v\} \\ &= V^2 \setminus \{(v, v) \mid v \in V\}. \end{aligned} \tag{4.2}$$

These are all the permutations of length 2 of the vehicles, i.e., all the ways we can combine one vehicle with another vehicle (excluding itself). In the case of $n = 2$, we have the vehicle pairs $U = \{(v_1, v_2), (v_2, v_1)\}$. Thus, a vehicle pair $\vartheta = (v_a, v_b)$ from U adheres to the precondition that either $a < b$ or $a > b$.

Next, we take each vehicle pair $\vartheta = (v_a, v_b) \in V$ and each waypoint pair $\rho = (p_{c,1}, p_{c,2}) \in P$ and calculate the distance of each part of the pair. This gives us $d_1 = \|S_a - p_{c,1}\|$ and $d_2 = \|S_b - p_{c,2}\|$, the distances that the selected vehicles would need to take if they are assigned the chosen sensor positions in this selection order. We use the norm related to the target space's coordinate system described in Section 3.1.

Because the vehicles have to wait for each other before they can perform the measurements at these locations, the actual *cost* of the selection is the maximum of d_1 and d_2. Afterward, the new locations of the two vehicles S_a, S_b become the selected sensor positions. We then perform the same steps to calculate the cost of the remaining positions, excluding ρ.

Since the cost of a selection depends on the current location of the involved vehicles, it is difficult to determine which sequence of selections provides the minimal total cost. We do not want to calculate every possible selection for every permutation of the vehicle pairs at every step. A greedy algorithm in this case can find a solution that is possibly non-optimal but still has a low cost.

At each step where the greedy algorithm needs to choose a selection of a vehicle pair and a sensor pair, it selects the ones that we find to minimize the following:

$$\underset{(\vartheta,\rho)\,\in U\times P}{\arg\min}\Big(\max(d_1(\vartheta,\rho),d_2(\vartheta,\rho))\Big) \tag{4.3}$$

The definitions for ρ, ϑ, d_1 and d_2 are as given before, but in Equation 4.3, they are shown in their dependent form, using whichever pairs $\vartheta = (v_a, v_b)$ and $\rho = (p_{c,1}, p_{c,2})$.

We select the pairs $\vartheta_m = (v_a, v_b)_m$ and $\rho_m = (p_{c,1}, p_{c,2})_m$ that are the result of the minimization within Equation 4.3. If there is more than one possible selection, then the greedy algorithm chooses the ones with the lowest vehicle or sensor indices, i.e, the first ones it finds when searching in and ordered fashion.

The greedy algorithm then applies this selection by updating the current locations S_a to $p_{c,1}$ and S_b to $p_{c,2}$ and removing the sensor position pair from P. It then continues with the next step with this new state. In Algorithm 4.2, we provide an overview of the greedy algorithm, where we show one method to find the vehicle and sensor pairs that minimize the norms, equivalent to Equation 4.3.

Algorithm 4.2 Structure of the greedy waypoint assignment algorithm.

Let $S_1, S_2, \ldots, S_n$ be the starting locations of n vehicles, and P a set of m pairs of sensor positions given as in Equation 4.1.

```
 1: procedure ASSIGN($S_1, S_2, \ldots, S_n, P$)
 2:     initialize a sequence of waypoints $A_i$ for each vehicle $v_i$, with $i = 1, 2, \ldots, n$
 3:     determine the vehicle permutation pairs $U$ as in Equation 4.2
 4:     while $P \neq \varnothing$ do
 5:         let $\delta_m \leftarrow \infty$
 6:         initialize $\vartheta_m$ and $\rho_m$
 7:         for all $(\vartheta, \rho) \in U \times P$ do
 8:             note: we have $\vartheta = (v_a, v_b)$ and $\rho = (p_{c,1}, p_{c,2})$
 9:             let $d \leftarrow \max(\|S_a - p_{c,1}\|, \|S_b - p_{c,2}\|)$
10:             if $d < \delta_m$ then
11:                 $\delta_m \leftarrow d$, $\vartheta_m \leftarrow \vartheta$ and $\rho_m \leftarrow \rho$
12:             end if
13:         end for
14:         note: we now have $\vartheta_m = (v_a, v_b)_m$ and $\rho_m = (p_{c,1}, p_{c,2})_m$ as per Equation 4.3
15:         add $p_{c,1}$ to the assignment $A_a$ for vehicle $v_a$
16:         add $p_{c,2}$ to the assignment $A_b$ for vehicle $v_b$
17:         $S_a \leftarrow p_{c,1}$ and $S_b \leftarrow p_{c,2}$
18:         remove $\rho_m$ from the set $P$
19:     end while
20:     return the assignments $A_1, A_2, \ldots, A_n$
21: end procedure
```

4.2.2 Collision avoidance

When vehicles are moving between their assigned waypoints, they could come across other objects that impede their progress. Even if we assume that the vehicles operate within an area of the space where there are no fixed obstacles, they could still hinder each other due to *conflicting routes* by crossing each other or blocking another vehicle's waypoint.

We could leave the vehicles to detect potential collisions on their own during the mission, as mentioned in Section 3.5. However, we want to know beforehand whether a certain planned mission can be completed safely and successfully, and avoid deadlock situations where vehicles block the waypoints of each other. Thus we integrate a form of *collision avoidance* into the objectives of the planning problems that we wish to solve.

In Section 3.5.1, the concept of a safe path is introduced. This notion allows us to define beforehand whether a route from one waypoint to another likely leads to an unsafe state. It does not provide certainty that a safe path always remains safe. In controlled situations, where there are no moving objects other than the vehicles, we assume that it is safe.

Since there might be multiple possible paths that a vehicle could take, we want to be able to deduce which one leads to the least conflicts and can be considered the "safest". We also want to know how long the chosen routes are, so we can receive a measure of mission length of the vehicles. The search algorithm from Section 4.1 helps in solving these problems, but the algorithm does not state how we keep track of the concurrent routes or how to make use of the resulting path, if it can be found.

We propose a collision avoidance planning algorithm which tracks the possible locations, delegates the safe path problem to a search algorithm and finalizes the routes of the vehicles. This algorithm keeps the synchronization points of the vehicles in mind. The current information of which vehicle has synchronized with another determines which routes we need to take into account. These are the concurrent routes which could conflict with a path toward a certain waypoint.

When two vehicles perform a measurement, they need to take separate routes to two locations. The routes should not cross each other. Afterward, their prior routes no longer conflict with any later route. Of course, a vehicle's route does not conflict with any earlier part of its own route. However, if there are more than two vehicles, a prior route may still conflict with some other route until each vehicle has synchronized with all other vehicles.

The collision avoidance algorithm integrates with the greedy sensor position assignment algorithm from Section 4.2.1. Whenever the assignment algorithm selects the vehicles to move to some positions, we check whether this is safe. The norm used in the greedy selection can also be augmented to use a more realistic distance, keeping detours in mind. The collision avoidance algorithm takes one step each time to accomplish this.

In between the calls to the collision avoidance algorithm, we keep track of the routes, locations and synchronization states of the vehicles. These global states persist between calls to the algorithm, so that a next step can continue from the previous state without having to do a lot of initialization again.

The graph of possible locations often needs to be altered during the algorithm. We track which locations are potentially dangerous, by removing the edges to them. Due to this, the search algorithm from Section 4.1 skips the dangerous areas. When a vehicle is not yet synchronized with the vehicle, then we mark its previous route as dangerous.

Note that synchronicity is not a symmetric relation. Thus, one vehicle may not have followed its entire route when the current vehicle is synchronizing with it. Therefore, we check whether the current vehicle has waited for other vehicle, not the other way around.

When we have found a route, then we start tracking this new route as well as the new location of the vehicle. If no route could be found by the search algorithm, then the visit to the given waypoint is potentially unsafe. The planning algorithm still continues, ignoring the vehicle's route, but we can mark the entire assignment invalid in some use cases.

To end this step, we place the graph back to the way it was before. The current vehicle's location becomes unreachable. The algorithm detects which vehicles are fully synchronized. Their prior routes as well the sets of vehicles they synchronized with are then forgotten.

The collision avoidance algorithm ensures that an assignment of waypoints is safe enough to put into practice. It copes with vehicles and missions that cannot avoid obstacles by themselves, and improves performance otherwise. We can thus still allow conflicting routes and predict problematic situations. The complete algorithm is shown in Algorithm 4.3.

Algorithm 4.3 The collision avoidance planning algorithm.

Let $S_1, S_2, \ldots, S_n$ be the current locations of n vehicles, initially their starting locations. Let v_p be the vehicle that we currently assign the location N_p to, and v_q a vehicle that synchronized with it there.

```
1: let $V \leftarrow \{v_1, v_2, \ldots, v_n\}$
2: let $W_1, W_2, \ldots, W_n$ be sets, where $W_i \leftarrow \{v_i\}$ are vehicles with which $v_i$ synchronized
3: initialize a graph $G$ where each node corresponds to (the area around) a location
4: remove edges from $G$ for node pairs that enter forbidden areas
5: remove incoming edges from $G$ for nodes corresponding to $S_1, \ldots, S_n$
6: initialize sequences $R_1, \ldots, R_n$
7: procedure AVOID($S_1, S_2, \ldots, S_n, v_p, v_q, N_p$)
8:     for all $v_i \in V$ do
9:         if $v_i \notin W_p$ then
10:            remove the edges for nodes in $R_i$ from $G$
11:        end if
12:    end for
13:    let $r \leftarrow$ SEARCH($G, S_p, N_p$)
14:    append $r$ excluding the goal point to $R_p$
15:    readd the edges for $S_p$ to $G$
16:    remove incoming edges for the node corresponding to $N_p$
17:    $S_p \leftarrow N_p$ and $W_p \leftarrow W_p \cup \{v_q\}$
18:    for all $v_i \in V$ do
19:        if $v_i \notin W_p$ then
20:            readd the edges for nodes in $R_i$ to $G$
21:        end if
22:        if $v_i \neq v_p \wedge W_i = V$ then
23:            clear the sequence $R_i$
24:            $W_i \leftarrow \{v_i\}$
25:        end if
26:    end for
27: end procedure
```

4.2.3 Evolutionary algorithms

In order to obtain a safe and well-ordered assignment of mission waypoints, we first need to find a good set of positions in the first place. In other words, we want to deduce which sensor positions are helpful for the reconstruction algorithm, and optimize toward a better result without increasing the mission length too much.

For this purpose, we design an evolutionary algorithm, which is based on a family known as multiobjective optimization algorithms (MOOAs) [13]. The process flow of such a MOOA is similar to the normal evolutionary strategy [6]:

1. Initialize a random *population*, where each *individual* represents a potential solution to the problem.
2. Pick one of the individuals and *mutate* it to create another individual.
3. As a final step of one *iteration*, *select* an inadequate individual and remove it from the population, so that it remains the same size.

This continues from step 2 for a given number of iterations $t_{\max}$. When the algorithm ends, we select one of the individuals objectively or subjectively, and output it as the solution. In the remainder of this section, we look into more detail how each step works, in which ways a MOOA differs from other evolutionary algorithms, and what the implications are for our purpose.

We create a population of μ individuals $X_1, X_2, \dots, X_\mu$, where a single individual X_i is represented by an ordered sequence of *variables.* We have η variables per individual X_i, namely $\alpha_{i,1}, \alpha_{i,2}, \dots, \alpha_{i,\eta}$. The variables receive values that are generated using a uniform random distribution over a given *domain* [8]. The domain may differ per variable, but its parameters are shared between individuals. Therefore, each variable $\alpha_{i,j}$ may receive real numbers from an interval $[a_j, b_j) \subseteq \mathbb{R}$, or they can be limited to the integers from $\mathbb{Z}$. A variable can also be binary, thus receiving a uniform random value from $\{0, 1\}$.

After the population has been generated, we determine whether each individual adheres to all *constraints* and calculate the *objective values.* For this purpose, the planning problem from Section 4.2 needs to provide a number of constraints and objective *functions* that can be evaluated to provide a score for a given individual. A constraint determines whether the individual is *feasible*, i.e., it is an acceptable solution, whereas an objective function can be *evaluated* to receive an indication whether one individual is better in reaching a certain goal than another individual.

In our case, the variables of an individual are a representation of a certain selection of sensor positions, and the function evaluations, if necessary, convert this representation to an assignment of waypoints for each vehicle and resulting fitness values. Using the greedy assignment algorithm in Section 4.2.1, we can determine such waypoints, and also obtain an objective value for the mission duration, which we want to minimize.

We use similar approaches to determine whether the positions are valid for a tomographic reconstruction, and deduce additional constraints and objectives. This provides for each individual X_i the feasibility value $f_i \in \{0, 1\}$, which is 1 if and only if all constraints are met. We also determine the values of β objective functions $g_k : \mathbb{R}^\eta \to \mathbb{R}$ with $1 \leq k \leq \beta$, so β function evaluations per individual. These resulting feasibility values f_i and objective values $g_k(X_i)$ play important roles during step 3, when we select an individual to remove. However, we first need to perform a mutation in step 2.

The mutation step randomly selects one of the individuals $X_s = (\alpha_{s,1}, \alpha_{s,2}, \ldots, \alpha_{s,\eta})$, using a discrete uniform distribution over the μ-sized population. We apply mutation operators to the values of all variables. This operator can have different effects on the variables. One common mutation operator uses a normal distribution $\mathcal{N}_j(\alpha_{i,j}, \sigma_j)$ with standard deviation σ_j, to slightly shift a real-valued variable $\alpha_{i,j}$. This may cause the value to leave the interval $[a_j, b_j)$. We can reject such mutations using constraints. Another simple operator bit-flips a binary variable with a specific probability ρ_j [7].

We can also use problem-specific operators that use knowledge about the relationships between dependent variables, such as trying to put one sensor point on the other side of the area of interest. In any case, we should attempt to keep the values within the interval constraints, otherwise the mutated individual is immediately infeasible and thus useless. The constraints and objectives are also evaluated for this new individual, after which we enter the selection step with $\mu + 1$ individuals.

Similar to a normal evolutionary algorithm, we attempt to remove infeasible individuals, so if there are any that fail some constraints, we randomly select one. Otherwise, we have to select an individual to remove from a population that contains only feasible individuals. Now, because we have multiple objective functions, we cannot simply decide to remove a feasible individual based on one function evaluation. Instead, we factor in all objective functions by grouping the resulting individual solutions.

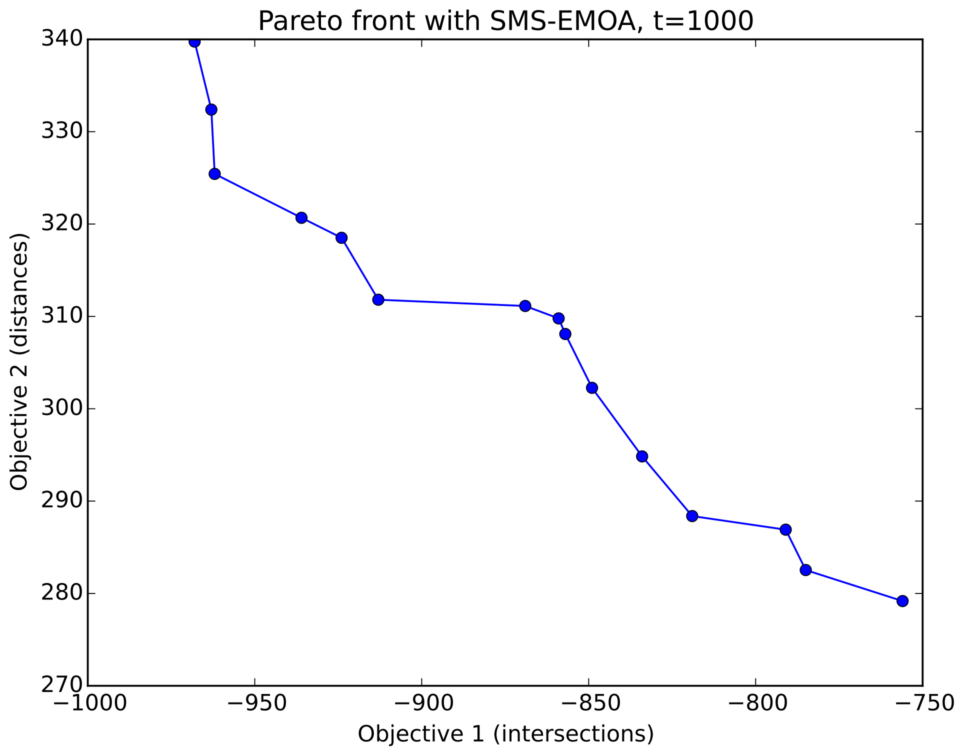


Figure 8: Example of a Pareto front with objective functions as axes.

We consider an individual X_a *dominated* if there is another feasible individual X_b, with $1 \leq b \leq \mu$ that is *strictly better* in all objectives, i.e.,

$$g_k(X_a) > g_k(X_b) \quad \text{for all } k \text{ with } 1 \leq k \leq \beta. \tag{4.4}$$

An individual is *nondominated* if there is no such individual that dominates it, according to Equation 4.4. We assume that all objectives are to be minimized. If a function g should be maximized, then we convert it to an objective by negating it, i.e., $g' = -g$.

We group the individuals into dominated and nondominated *layers*; the latter group is also called the *Pareto front*. If there are no infeasible individuals, but we do have dominated individuals in the population, then we randomly select and remove one of them.

When the MOOA has a population containing only feasible, nondominated individuals, then it uses a different selection mechanism, which removes the individual that contributes the least to the current population. Such heuristics keep individuals whose neighboring points are the furthest away, using specialized distance or contribution calculations [16]. In addition, we keep the nondominated individuals that have one currently minimal objective value, i.e., the endpoints if the objective values of this front were plotted, as shown in the example in Figure 8.

Thus, in each iteration, we perform one mutation and one selection. The aim is to *converge* toward some feasible, optimal individual, whose objective values cannot be improved in any way. The rate of convergence of the evolutionary algorithm is highly dependent on the problem's constraints and objectives as well as the mutation operators that we use.

Due to this, the MOOA might accidentally create a mutated individual that is infeasible or dominated in every iteration. This causes the Pareto front to remain motionless, even though the optimal solution has not been found. We need to formulate the problem and perhaps create specialized mutation operators to prevent such a standstill.

Once the MOOA reaches the maximum number of iterations $t_{\max}$, we do not have just one solution, but at most μ feasible, nondominated individuals. Each of them provides a useful set of sensor measurement positions and corresponding missions for the vehicles. Some individuals may be better in one objective, while others are relatively good in another.

If we only want one solution instead of a range of possibilities, then we need to make a final selection from the Pareto front. This can be done subjectively by comparing the solutions manually and taking the preferred one. Another selection strategy is to use a *knee point* [10], which is a nondominated solution that is average in all the objectives.

In the end, the resulting individual solutions heavily depend on a number of properties that we can tune: the initial population, the mutation operators and the parameters of the random distributions that they use, the selection strategy, the number of iterations that we let the algorithm run, and of course the constraints and objectives themselves.

We must thus determine whether the solutions are suitable, and if so select one to use for the final missions. If the result is unsatisfying, then we can optionally tweak the algorithm and see what another run provides. The nondeterministic nature of the stochastic process means we may receive widely different results from a new run.

5 Context

In the model that we defined in Section 3, we include certain assumptions about the space that we are in. This plays a role when we take a more practical approach and apply the model to a physical mission with actual vehicles. This allows us to get to a working implementation in Section 6.

In this section, we outline the differences between the defined model and the reality. These differences spark the need to make assumptions for the model to work reliably, or create practical solutions so we can actually make use of the system of elementary definitions.

We propose changes to our model that make it reflect reality better. This includes physical challenges related to movement and location detection. We state some possible reasons for the resulting inaccuracies and uncertainties. Finally, we create a contingency plan that provides solutions for overcoming these challenges in some cases.

As mentioned in Section 3.3, a vehicle may have limited information about the space it is in. This applies to positions that the objects are at. Also, the current location of the vehicle itself might not be directly accessible. The model tracks the precise location of each vehicle, but the vehicle cannot determine it with the same accuracy.

Similar to the other sensors in Section 3.4, the vehicle has a *location sensor* that determines a position using previously collected information and live data, such as an external positioning system, a projected grid on the ground surface of the space, or the measured speed of the vehicle. All these data sources may be inaccurate, which propagates to the quality of the location sensor and results in a fuzzy position.

It is important to keep the accuracy of the location sensor within acceptable bounds. A sensor that provides erratic results makes it difficult to know whether the vehicle has reached a certain waypoint. If the *error* or difference between the model location and detected point increases through time, then the usefulness of the location sensor decreases.

Time is also an impeding factor for sensors. A sensor could spend some time units to process the data sources. Due to this, the location sensor detects the location of an earlier time unit while moving, and the distance sensor might not detect a nearby object immediately. Certain distance sensors use the echo of an ultrasound signal to calculate the distance, for example. Thus, timing is an essential part of the model.

Another problem is when there is an oncoming vehicle with an angle that is not exactly a straight angle, or even if a sideways collision is imminent. A servo can be used to scan the surroundings to mitigate this problem. Servo angle changes are rather instantaneous in practice, but again the distance sensor takes time to process.

We need to take care that such problems, as well as other characteristics of the mission, do not make the mission needlessly long. The vehicles have a lifetime in which they can operate. This lifetime is limited by the power consumption of the vehicles. They need to use batteries to operate freely in the real world. A larger space means the vehicles need a longer range. This is only possible if the battery's capacity and current allows for a longer lifetime or a higher motor speed.

All these problems mean that we need a sort of *contingency plan.* We need to detect when the location sensor becomes inaccurate or when the battery charge of a vehicle is approaching critical levels, and take appropriate action to keep the vehicle in a safe location. Some of these problems can be solved through the use of additional sensors or heuristics, and some need a human touch to stop the vehicle remotely using a command via the infrared sensor.

The contingency plan alters the mission mentioned in Section 3.5. It adds new waypoints when the vehicle strays, or stops the mission early. The plan independently monitors external influences as well as the mission itself. Thus, there is a sort of *watchdog* that oversees the sensor data and accuracy, and acts immediately when it detects issues.

6 Implementation

As a part of the research within the mobile radio tomography project, we create a toolchain that plans missions, operates vehicles, performs and collects measurements and creates a reconstructed image using radio tomographic imaging [28].

In this section, we describe our contributions to this mobile radio tomography toolchain, which includes a control panel for planning and monitoring missions from a ground station, as well as the autonomous vehicle movement which can act upon information from environmental sensors. The definitions from Section 3 form a useful basis that already describes a large portion of the implementation details.

We provide an overview of the components created for these purposes in Section 6.1. We then focus on the types of autonomous vehicles in Section 6.2. The functionality of the sensors is outlined in Section 6.3. We show various kinds of missions that help to position the tomography sensors at specific locations in Section 6.4.

Section 6.5 then studies optimization algorithms for creating waypoint-based missions. These algorithms should solve the problem of sensor positioning. We focus upon the details of the specific planning problem and relationships to the reconstruction problem as well as properties of the geometry.

6.1 Overview of the components

We implement a toolchain that consists of multiple parts. A major part is the vehicle's environment, which provides mission control and runs as a service on the vehicle's central computer without direct user interaction.

Independent of the vehicle part, we have a control panel, which is a graphical user interface on a ground station, e.g., a desktop or laptop computer. The researcher can find out various status information before and during a mission, or use planning and reconstruction tools. These tools satisfy different needs within the phases of the mobile radio tomography project mentioned in Section 1.2. Parts of the ground station are described in Section 6.6.

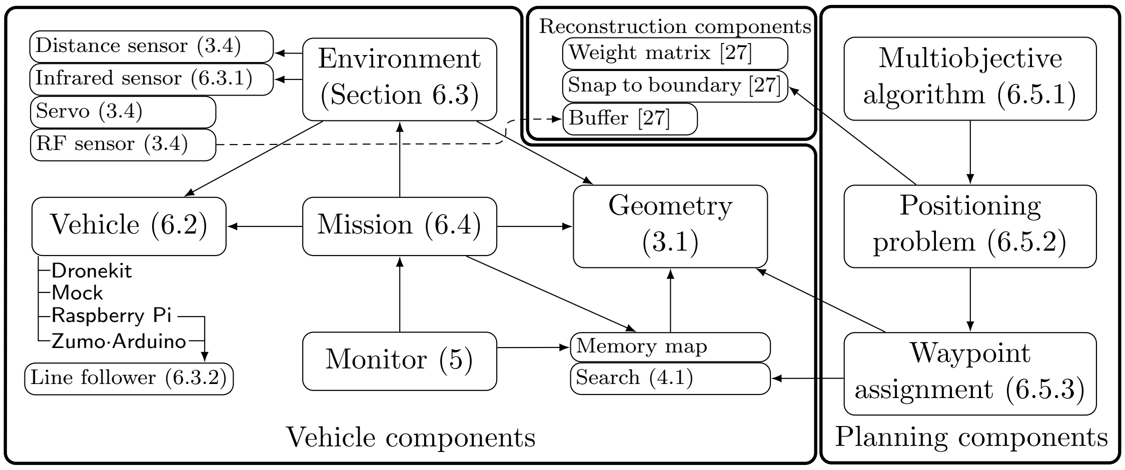


Figure 9: Diagram of components in the toolchain. The numbers refer to sections where the components are described in detail. An arrow indicates that the component at the starting end makes use of the far end component.

The overall toolchain consists of a number of components that each part can make use of. These may be related to the distance sensor, determination of the current location, the missions, the vehicle and its environment. There is also a planning component for the control panel. Each component may be used in simulation or inside the control panel, as well as have a function in a physical environment with the actual vehicle hardware.

For these components, there is often a main purpose for which it was created. For example, sensor and other vehicle-based components are mostly used in the vehicle part, while reconstruction and planning components are useful for the ground station control panel. However, some components depend on the functionality of another component, such as the planning problem making use of reconstruction and geometry details. This is possible within the cohesive structure of the toolchain. We provide a diagram of the components and the dependencies between them in Figure 9.

The components often have a well-tested basis and specialized versions for different uses. Thus, it is easy to replace one version with another. This also means that we reduce the risk of one version not working, since the common code already behaves as expected. Section 6.7 describes how we make use of tests to ensure that the code works correctly.

A component can depend on another component even when the latter is a member of a different part, e.g., a planning component using a vehicle or reconstruction component. Because the component has an accessible interface and works well in its own part, it is easy to reuse that component elsewhere. This makes our toolchain solid as well as reusable, which is essential for using it in multiple contexts.

6.2 Vehicles

An important component of the mobile radio tomography toolchain is the vehicle interface. This component makes it possible to provide instructions to a physical vehicle in order to make it arm its motors, move, rotate, lift off, adjust its settings, travel to a certain waypoint, and so on. These instructions are collected and converted to signals that the specific vehicle is capable of understanding. Additionally, we can request the vehicle's status, e.g., its current mode, position and orientation, and act upon problems such as low battery power when they are detected by the vehicle's control circuit.

The vehicle component consists of several implementations for different types of vehicles, which are introduced in Section 3.3. Each version supports a compatible interface. This makes it possible to use the same missions on different hardware platforms.

6.2.1 Hardware communication

The communication between the high-level mission components and the physical hardware is established by the vehicle component. This means that the other components can send requests to the vehicle component in order to take a certain action, retrieve status information or update it. The vehicle interface processes the action specification, such as moving to a given location at a specific speed. We convert the information so that it is understandable for a lower-level hardware device. This can mean that we enable some sort of motors for a certain amount of time. The exact process differs between the vehicle types.

As an example, many simple rover vehicles have a motor speed controller that receives a number of input signals. These signals determine how fast it should turn its motors, and if so whether each of them should turn forward or backward. The latter type of signal can be easily provided using logic level signals, where there are two possible signal voltages: a high value and a low value. Often, the motor speed is provided as a pulse width modulation (PWM) signal, which uses signals at high and low power at specific durations to signify the PWM value. A Raspberry Pi [32] has no hardware PWM support, but can provide PWM signals using fairly accurate software timers.

In some cases, it might be desirable or necessary to use intermediary hardware. This can be the case when the motor speed controller uses different logic level voltages than the Raspberry Pi does, for example. The intermediary hardware may simply be a conduit to convert the logic level voltage, or it might be programmable so that it can provide a more sophisticated backend interface to the vehicle component.

Such middle man hardware might make it easier to develop the vehicle component. The Zumo shield [31], for example, can be programmed using an Arduino [4] with an existing library, thus we only need to create a communication interface between that and the vehicle component, and perform the right actions based on the corresponding commands.

The intermediary hardware may also be a full-fledged autopilot, such as the ones described in Section 2.2. This means that it already has some intelligence in choosing a path to reach a location, and fixing the location when it is imprecise. The autopilot hardware may provide the needed support for battery monitoring, connecting additional peripherals and sending PWM values to the motors.

6.2.2 Interfaces

From the software perspective, the vehicle component is an interface that allows other components to exchange status information with the vehicle. This means that all components can share the same location and related elements of the vehicle's state.

For this reason, the vehicle interface must either provide its own location based on some internal inference system, or it can use another component, such as the line follower in Section 6.3.2 to retrieve a location. The location can be represented as a coordinate tuple from the coordinate system in Section 3.1.1. Some interfaces may add in more or less details in the location, such as excluding the altitude component or tracking the location using multiple coordinate systems at once.

The same applies to other properties of the interface, which the vehicle must always have some knowledge about. The vehicle has a *home location* where it starts its mission, and may provide an automated way to return to this home location at any point. The vehicle has different *modes* in which it can operate. This includes an automated mode where the vehicle follows a sequence of *commands*, such as waypoints, and a guided mode where the mission component presents actions on the go. The vehicle must be *armed* before it can do anything, and it can be disarmed whenever it would be safer to stop all motors than it is to continue.

We can also access live information from the interface, including the vehicle's *speed*. The speed can either be provided as one unit in the current direction of the vehicle, or as a velocity in the three components of the coordinate system. The speed is the approximate distance that the vehicle travels in one time unit. The *attitude* of the vehicle, i.e., the direction in which it would travel, is also provided as rotations compared to the three components, known as the *roll*, *pitch* and *yaw*.

Each vehicle interface has some way to provide commands. The vehicle should eventually follow these in the order that they are given, unless they are cleared by some other command. At the very least, the vehicle must be able to go to a specific waypoint and wait at that waypoint until a different command arrives. Depending on the type of vehicle, we can take off to a specific altitude or rotate at one point.

Figure 10 demonstrates the various interfaces that we support. The group of MAVLink vehicles support a specific internal communications protocol for receiving commands [24]. They are more geared towards flying drones, although rover vehicles are also supported. An internal autopilot determines the best way to process a certain command.

The group of robot vehicles are interfaces to rovers that can follow lines on the ground. The coordinate system is based on the grid geometry from Section 3.1, so only discrete points can be provided in the waypoint commands. The guided mode allows some more freedom, but the main mode of operation is following lines, arriving at intersection points and rotating to different directions.

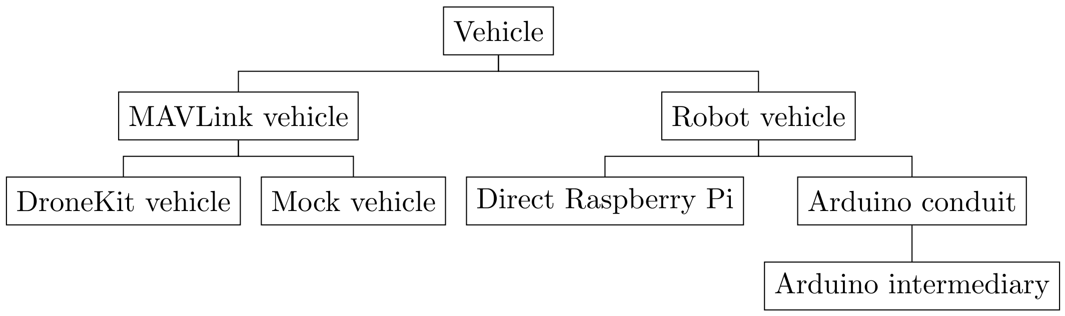


Figure 10: Inheritance diagram of the vehicle interface.

6.2.3 Simulation

Some of the vehicle interfaces support *simulation*. In simulation mode, the interface does not actually communicate with actual hardware as mentioned in Section 6.2.1, but instead allows an external simulator to follow what the vehicle is doing.

This makes it possible to enact mission and determine its likely outcome, without actually sending a vehicle into a physical environment. The vehicle is then an "engine" for the simulator, keeping track of what happens based on external commands and configurable environmental hazards. Thus we can find out how a mission would act in certain situations.

While the simulator itself is not a part of the vehicle, it is tightly connected to its state. The simulator can show a map of the current vehicle's location, or provide a first-person display of the space. This takes into account which direction the vehicle is facing. We can build in additional safety checks to see if the vehicle would lose track of its location, or detect whether it collides with another object.

Some vehicles may behave randomly or erratically in some situations. During simulation, it is sometimes helpful to make this behavior more deterministic. This allows us to test the missions without variations between test runs. The mock vehicle in Figure 10 reflects the behavior of complicated autopilot, but it is only dependent on timing, not on other sources of randomness. This gives us a solid foundation for testing missions, which we expand upon in Section 6.7.

6.3 Environmental sensors

The vehicle has a number of sensors with various types. We need to control these sensors through standard interfaces, so that we may replace the hardware with similar sensors if necessary, or use simulated versions of the sensors. Additionally, we want to keep the interfaces of the sensors in one place, so that it is easy to use them from the mission components.

For this reason, we initialize the sensors within an *environment*, which is a software layer that provides access to information from the space that the vehicle exists in. It provides a convenience access to the vehicle's interface, including some functions that perform distance and angle calculations.

The environmental sensors interact with the space in different ways, which we introduced in Section 3.4. The tomographic sensor, also known as *radio frequency* (RF) sensor, makes it possible to exchange packets of information with other RF sensors at the ground station of Section 6.6, as well as with the other vehicles. They also provide a received signal strength indication (RSSI), which can differ due to attenuation and absorption by objects of different materials that lie in between the sensors [27]. A standardized interface gives us the possibility to change between different modes, such as sending and receiving specialized packets, or continuously measuring the RSSI. Internally, the packets are forwarded to the physical sensor device.

The distance sensor has an interface to measure the distance to the first object along a line segment starting from the vehicle's location at its current attitude. A physical distance sensor might work using ultrasound signals. It then measures the time t between such a signal and the corresponding echo signal, and converts this to a distance $d = t \cdot c_a$, where c_a is the speed of sound in air.

We can simulate the behavior of a distance sensor for experimental purposes. We define a number of simulated objects, such as the ones from Section 3.2. We then calculate the minimal distance to the intersection point at each object for the line segment, if there exists such an intersection. This requires some geometric models for detecting intersections of lines with polygons, planes, other lines, as well as the different base types of objects [19].

The distance sensor usually points in the same direction as the attitude of the vehicle. However, it is possible to alter the behavior of this sensor using a *servo*. This object cannot sense anything from the environment, except its own rotational angle. Often, the servo has a limited range or *duty cycle* in which it can quickly rotate between. The servo receives a PWM value that corresponds to the requested angle. When combined with a distance sensor, we can detect nearby objects in different directions, without having to rotate the entire vehicle.

6.3.1 Infrared sensor

The infrared (IR) sensor is similar to an RF sensor in that it can receive a packet from another source. An IR sensor cannot send any data, and it loses the whole signal when it is blocked due to scattering, rather than receiving it at reduced power. Because it operates in a separate light spectrum and works independently from the other sensors, the IR sensor does not interfere with measurements of the RF sensor or the distance sensor.

IR sensors are simple hardware accessories, that do not need much more than pass through the light pulses that it receives. These can be converted back to specific code sequences, which may or may not correspond to button presses on a remote control. We load the configuration of code sequences for one remote type, so that the vehicles listen to commands from that remote.

The commands that we send to the IR sensor include starting and stopping the mission. This is important, since we need a safe way to stop a mission remotely, without interference of RF sensor measurements. We can also send commands to go to specific waypoints or follow a certain mission. This is mostly for experimenting with new missions or tuning the line follower in Section 6.3.2. The button presses are limited to simple commands, so actual data transmission must fall back to the RF sensor.

6.3.2 Line follower

The robot vehicles from Section 6.2.2 can track their location using an additional sensor called the *line follower*. This sensor is actually an array of infrared emitters and light-sensitive diodes. The light sensors detect how much of the light is reflected from beneath the sensor, which is mounted on the front of the rover, facing downward.

A darker surface absorbs more light compared to a light surface. Thus, a reflectance sensor above a black line receives a detectable lower intensity compared to another sensor above a white background. Internally, this detection works similar to the distance sensor's echo detection: a charge in the circuit decays based on the light intensity received on the diode, and we can time how long it takes before the charge appears to be lost. This gives us an approximate *grayscale* value of the surface.

We combine the values of the sensors, which gives us a way to detect where there is a line, compared to a background. This edge detection may work in any situation where we have a surface with grayscale color differences, but it works better in the restricted case of thick, black lines on a white background. We can determine a *threshold* where any grayscale value below this threshold is not a line, while every value above it is considered to be a line.

This allows the robot vehicle to follow lines, detect intersections, and rotate to one of the cardinal directions on a surface with a printed line grid. If the vehicle is not moving exactly straight on a line, then it can detect that it is diverging from the line and slightly adjust the motor speeds. Once the reflectance sensor detect only black values, then we consider the location to be an intersection, which corresponds to a discrete coordinate tuple. At an intersection, we can perform stable measurements, or rotate the vehicle into another direction, for example to move to the next waypoint.

6.4 Missions

The vehicle's normal mode of operation is to follow the instructions provided by the mission component. This component takes the status of the vehicle into account and sends commands to move toward certain points in succession. In this section, we describe how the component is set up, which modes it supports and how we make use of different kinds of missions to reach our goals.

Before the vehicle can move around, the mission undergoes its first phase of verifying the automated *preflight* checks. The mission must ensure that the vehicle is ready to arm before it starts flying or driving. This can include waiting for a signal from the ground station, via the RF or IR sensor from Section 6.3. If the mission has a fixed sequence of waypoints, then we must wait until the waypoints are loaded and any old data is cleared.

The mission can then arm the vehicle, which should be a simple phase of powering and starting up the motors. Depending on the type of the vehicle, the mission then takes off to the altitude where we want to start the actual mission. This optional phase can also be used for other purposes, for example to calibrate its line follower sensor of the robot vehicle from Section 6.3.2.

The mission can work in different modes that alter the vehicle's mode and state. In *guided* mode, the mission works in steps: at a scheduled interval, the mission checks whether the vehicle's location is correct, and alters its speed, attitude and other properties, if necessary. This gives the mission a lot of control over the vehicle, which gives us more freedom in designing the mission but removes the focus away from visiting specific sensor positions.

The *auto* mode still allows us to frequently monitor the mission, but the vehicle component is fully responsible for moving to predetermined waypoints. We can detect when we are close to a waypoint or alter the sequence in case of unexpected safety problems, for example to avoid collisions. Thus we can easily design a fixed mission and still allow some leeway.

During the actual mission, the RF sensor enters its active state, where it continuously gathers measurements. Thus we can only use the measurement packets for communicating between the vehicles and the ground station. Other packets for configuring the mission and the other components must be provided during the passive state of the RF sensor. This means we can only alter the mission before the arming phases.

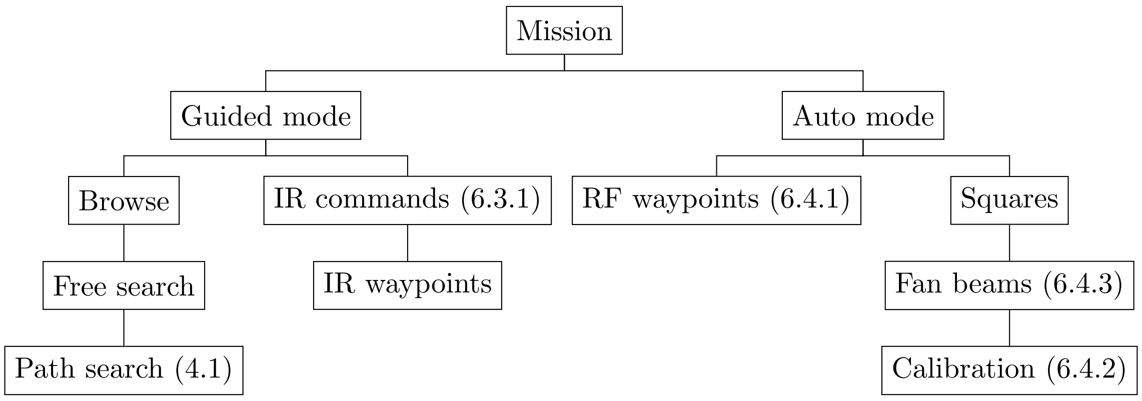


Figure 11: Inheritance diagram of the mission components.

Some of the missions that we implement are included in the overview in Figure 11. The IR sensor missions are described in Section 6.3.1. The simplest guided mission *browses* the surroundings of the vehicle, by holding it at the same position but rotating the vehicle or its distance sensors.

Since we do want to move around with the vehicle, another mission uses the browsing mission as a building block: the *search* mission moves in one direction and browses its surroundings when the vehicle is within the region of influence of another object. It then chooses the safest direction to move toward, while trying to stay close to the objects. This allows us to scan the objects with the distance sensor and the RF sensor.

We can replace this heuristic approach with a mission that uses a search algorithm from Section 4.1. It follows a certain list of waypoints, but when it is too close to another vehicle or object, then it uses the search algorithm to find a new path to the next waypoint.

The missions that use the automated mode usually assume that there is a safe zone around the area of interest, but using safe paths is also an option. The sequence of waypoints are designed to improve the RF sensor measurements for the reconstruction algorithm. We look into more detail how these waypoints work in Section 6.4.1, and present a number of waypoint sequences in implemented missions in Sections 6.4.2 and 6.4.3.

6.4.1 Waypoints

A waypoint is a location which should be visited by the vehicle it is assigned to, during its mission. It is a core element of all automated missions and some of the guided missions. We define waypoints in this context in Section 3.5. In this section, we look into more types of waypoints, which are specifically helpful when collecting RF sensor measurements.

The reconstruction algorithm requires measurements for links between sensors. During the mission, we visit these sensor positions such that there is a vehicle at each end of the link. This imposes restrictions on how we design the mission of the individual vehicles.

Firstly, the reconstruction algorithm assumes that the positions in all sensor links are on the same altitude. This is because the reconstruction creates a “slice” of the area of interest. In this slice, we need a large number of intersections between a dense network of links. We thus want to have the links to be on the same plane. A flat plane is the simplest and most useful to have in our use cases.

This means that we should make the vehicles operate on the same altitudes. For flying vehicles, we can take off to the desired level, and potentially chain the same mission on different altitudes. For other vehicles, we skip this part of the mission which occurs after arming the vehicle. However, one could alter the composition of the vehicle so that the sensors are mounted at the required altitude.

During the mission, we want to gather reliable measurements. This helps the reconstruction algorithm to generate a radio tomographic image that clearly reflects the physical objects inside the network. Thus, RF sensor measurements while the vehicle is moving can be considered unstable. Additionally, we need to synchronize with other vehicles to ensure we actually measure at the requested sensor link positions.

We can let a vehicle wait for the other vehicles once it reaches a waypoint. The RF sensor can send a packet containing enough data to synchronize the vehicles in this way. However, sometimes we just want to provide a waypoint to the vehicle so that it follows a certain path, such as a safe path from Section 3.5.1.

There are thus multiple types of waypoints, including ones that cause the vehicle to *wait* once it reaches such a waypoint, and ones that allow the vehicle to *pass*, without taking concurrent vehicles into account for the purposes of collecting measurements.

A mission can consist of a combination of these types of waypoints. A frequent pattern in some of the missions discussed here is to move the vehicle along a straight line or a more complicated path, and perform measurements at regular distance intervals. We can then have waypoints where the vehicle waits, mixed with passable waypoints.

If the series of waiting waypoints are at a fixed, evenly spaced linear interval between the starting point and the final waypoint, then we call these points a *range*. We can implement a range by only stating the final waypoint and the total number of waiting points in the range, assuming we know the previous location. Note that waiting at the same location a number of times is also considered to be a range.

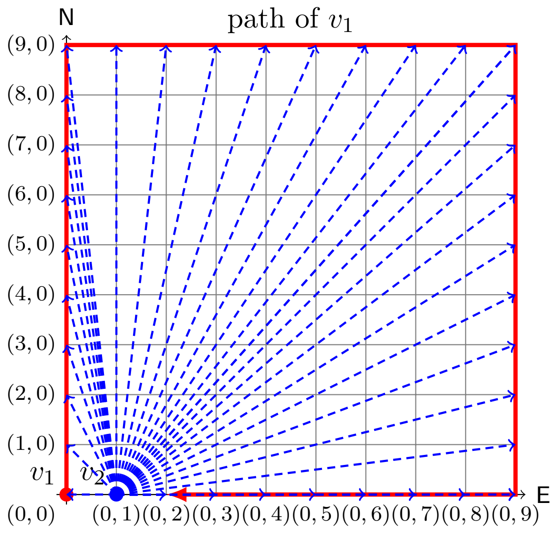
6.4.2 Calibration

Certain reconstruction algorithms make use of *baseline* measurements. This means that we need to perform measurements for all possible links beforehand, while the area of interest is devoid of the objects that we want to detect eventually.

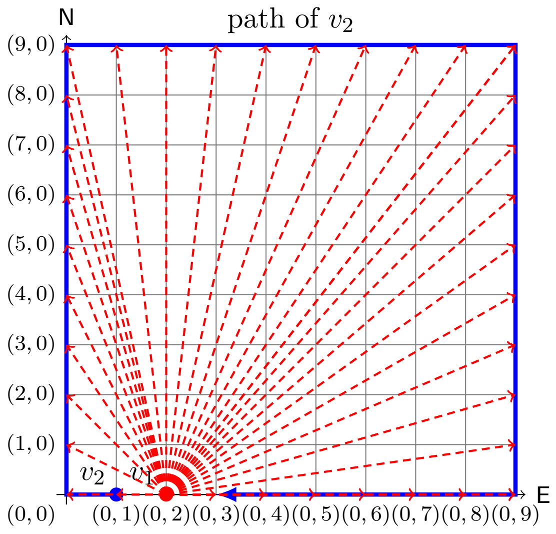
During the baseline measurements, the area may contain uninteresting objects, such as walls or other static objects, which are then ignored during the reconstruction phase. This is done through a comparison between the baseline measurement and the actual measured link strengths when the area is filled.

In some cases, we may be able to perform the same mission twice for this purpose, first to collect the baseline measurements, then the actual mission of gathering link strengths. However, if we want to compare the influence of different missions, or reuse a stable set of baseline measurements later on, then this limited collection sweep may not be sufficient. For completion, we should measure every possible link that we could ever have, to ensure that the reconstruction has a full set of baseline measurements.

This brings us to the design of a *calibration* mission. This mission works in a grid-like space by visiting each position that we could perform measurements at. It makes use of two vehicles, that each swap roles between two kinds of cycles: a movement cycle and a stationary cycle. Each time, one vehicle moves clockwise from its current location around the area of interest. The other vehicle stands still on the location just one grid cell away from the first vehicle, when seen counterclockwise.



(a) Cycle with v_1 from $(0, 0)$ through $(0, 2)$



(b) Cycle with v_2 from $(0, 1)$ through $(0, 3)$

Figure 12: Examples of calibration cycles with two vehicles, where one vehicle remains stationary and generates measurements with all other grid edge points.

The first vehicle ends its cycle one cell away from the second vehicle's location. An example of this cycle for a 10 by 10 grid with starting locations $(0, 0)$ and $(0, 1)$ for the two vehicles, respectively, is shown in Figure 12a. After this, the second vehicle starts its movement cycle, with the first vehicle playing the stationary role. Such a cycle is shown in Figure 12b.

These cycles measure different links. However, once we start repeating this for the whole network, continuing along with the clockwise cycles as if it is a relay race, then this may result in duplicated measurements. This is because a measurement from one location to the other automatically gives us the measurement in the other direction as well.

However, the baseline measurements may have a use for these twin measurements. This is because the vehicle at each sensor point may differ during the calibration mission. If the vehicles are equipped with RF sensors that have slightly different antenna properties, then the reconstruction algorithm can account for this if it has sufficient measurements from both vehicles in all permutations.

Time is not a huge factor during the calibration, although we do want to limit the time needed per measurements and the total number of measurements. The mission duration can grow quadratically for larger network dimensions, thus we ensure that the performance of the calibration mission is adequate.

6.4.3 Fan beam and straight line patterns

During the design of missions for the purpose of collecting measurements for tomographic reconstruction, there are some patterns that appear to be useful. Such patterns provide a large volume of different measurements within a small time period. Such patterns often let the vehicles travel some distance, while being on, e.g., opposite sides of the network, and the resulting measurements are spread out in such a way that they cover a large portion of the network.

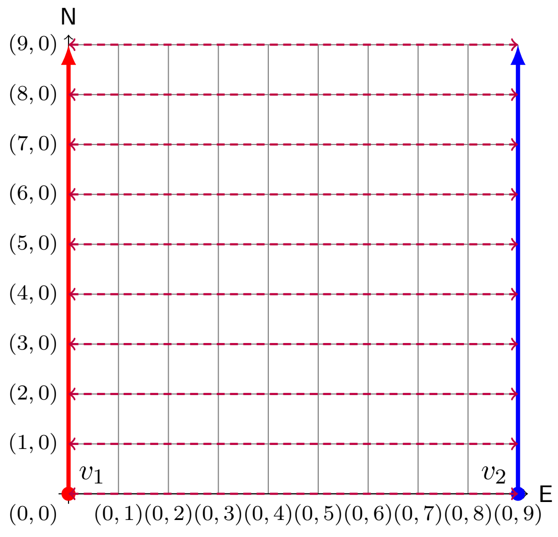
A mission may be built up from a combination of such patterns. One problem is that going from one pattern to another might waste time, in case that they do not fit together exactly. We could move the vehicles at full speed between these positions, or we can make use of certain other patterns that bridge the gap and provide link coverage in other parts of the network.

In this section, we look into two specific patterns, as well as describe some related patterns that augment them. The first and simplest pattern that we present is the straight line pattern. Here, two vehicles move in the same direction along opposing edges of the network. They move from one end point of such an edge to the other, synchronizing with the other vehicle at each sensor point in between.

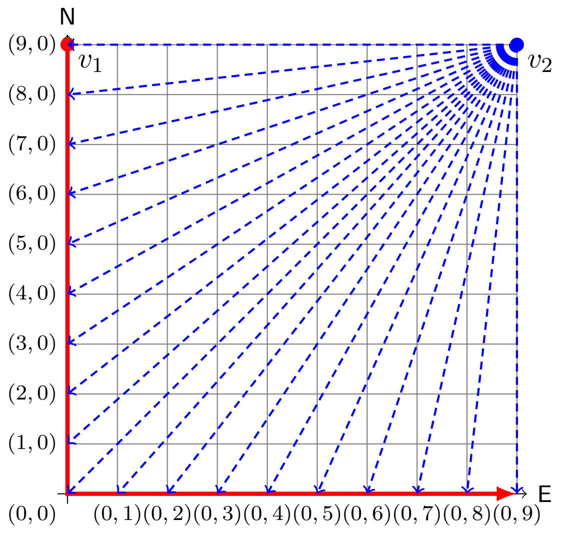
This results in measurements that cross the network. These links are parallel to one of the first two dimensions of the space, or at least relative to the area of interest. If the vehicles move northward and southward, then we receive measurement links parallel to the eastward axis, and vice versa for eastward or westward movement. Movement that is not in one of the cardinal directions results in angled lines, which may also be an option. An example of northward movement providing straight lines is shown in Figure 13a.

A related pattern makes the vehicles move from one corner of the network, across distinct edges, to the corner on the far end of a diagonal. This works quite differently from the straight line pattern, but results in measurement links that are laid out like diagonal lines.

Straight line patterns are quite simple, and they are fairly similar to the waypoint ranges discussed in Section 6.4.1. A straight line pattern can be defined by two waypoint ranges, where there is one range for each vehicle. The waypoints are of the waiting type, and the start and end points of each range only differ in one coordinate value. Waypoint ranges can thus be useful as a building block for patterns involving multiple vehicles, even more complicated patterns that still involve some straight lines or stationary points.



(a) Lines, v_1: $(0,0) \to (9,0)$, v_2 : $(0,9) \to (9,9)$



(b) Fan beam with v_1 from $(9,0)$ through $(0,9)$

Figure 13: Examples of straight line and fan beam cycles. Two vehicles move forward in a parallel fashion. Then, one moves along the sides that the stationary vehicle is not on.

In Section 6.4.2, an example of a more sophisticated pattern is shown, where calibration cycles lead to a fan-like form starting from a stationary sensor point. We can adjust such a fan beam pattern to include fewer links. The links that cross an edge on which the vehicles move around is not very relevant for the tomographic reconstruction, so we can start the movement cycle from a corner instead. When we put the stationary vehicle in the other corner of that edge, then we can stop the movement cycle when the vehicle has reached the start of the other connected edge.

This movement pattern leads to a fan beam which intersects with many grid cells, and has a high density in one corner. Figure 13b provides an example of this. We can also place the stationary vehicle in the middle of an edge, and let the other vehicle move along the three other edges, which provides yet move fan beam patterns. We can sometimes skip parts of those edges when they provide few different measurement.

A fan beam pattern can often be joined together with other fan beams, or we can alternate it with a straight line pattern before performing yet another pattern. The order in which we perform certain patterns can determine how quickly we obtain enough intersecting links in the entire network for the reconstruction algorithm to function well.

If we want to combine such patterns with a fan beam where the stationary vehicle is in the middle of an edge, then we can perform certain corner patterns. Two vehicles moving on perpendicular edges result in some more unique intersecting links around that corner. We can also skip these measurements and quickly move into the correct positions for the fan beam pattern.

6.5 Planning

The vehicles can move around using hand-crafted missions, but we also want to automate the planning part. The planning problem from Section 4.2 can be solved using algorithms that we demonstrate there. There are some more details that we need to make explicit in order to present a full implementation of the planning component. We slightly alter some of the algorithms to obtain information that can be used elsewhere. We also state the precise objectives and constraint functions that we use. Finally, we discuss performance considerations that help make the implementation viable to run.

We divide the implementation details into specific sections. In Section 6.5.1, we describe the novel mutation operators, constraints and objective functions that we use within the evolutionary multiobjective algorithm. The evaluation of the individuals and generation of sensor positions is explained in Section 6.5.2. We then discuss the assignment of waypoints while trying to avoid collisions in Section 6.5.3.

6.5.1 Multiobjective optimization

We make use of an evolutionary multiobjective optimization algorithm to evaluate, mutate and select a population of individual solutions. The iterative nature of the algorithm means that we might converge toward certain optimal or nondominated solutions using such an algorithm. This does greatly depend on several factors, such as how we limit our search space, how we design our objectives and which operators we use for mutation and selection.

The constraints and objectives are tightly related to the problem that we wish to solve; a different problem would need other evaluation functions. There are some basic constraints that ensure that the variables of one individual remain within the bounds of their domains when we mutate them, as mentioned in Section 4.2.3.

Additional constraints help the algorithm so that it searches in the right direction of a large search space. The constraints reject individuals that are clearly infeasible. The reconstruction algorithm provides us with a weight matrix containing information about sensor links that intersect with certain pixels: this weight matrix W_i of an individual X_i is an $\ell_i \times \psi$ matrix, where ℓ_i is the number of links from the individual that actually cross the network, and ψ is the number of "pixels" within the network. Furthermore, $w^{(i)}_{j,k}$ is the value in row j and column k of the weight matrix W_i as provided by the reconstruction algorithm [27]. We then have the following predicate:

$$Q_{1,i} : \exists j : \forall k : w^{(i)}_{j,k} \neq 0 \tag{6.1}$$

If some pixels are not intersected at all, then the reconstruction cannot determine a suitable pixel value, which we avoid with the constraint in Equation 6.1. Each column corresponding to some pixel has at least one link that crosses it.

Another constraint is that we need enough links to consider the mission feasible. It may occur that the algorithm misplaces many links along the edges of the network so that they do not intersect with the network. It is acceptable to discard some of the measurements, but the number of correct measurements should be above a certain threshold γ, dependent on the total number of links that we want. This gives us the second constraint in Equation 6.2 and the combined feasibility value in Equation 6.3:

$$Q_{2,i} : \ell_i \geq \gamma \tag{6.2}$$

$$f_i = \begin{cases} 0 & \text{if } \neg Q_{1,i} \vee \neg Q_{2,i} \\ 1 & \text{if } Q_{1,i} \wedge Q_{2,i} \end{cases} \tag{6.3}$$

The objectives of the reconstruction planning problem are similar to the constraints. Many of the pixels in the weight matrix should be intersected by multiple links, otherwise the reconstruction algorithm would not be able to determine whether a signal strength for one link actually influences such a pixel. We can wrap this into an objective function by taking the sums of the filled weight matrix entries as the function value in Equation 6.5:

$$h(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{6.4}$$

$$g_1(X_i) = -\sum_{j=1}^{\ell_i} \sum_{k=1}^{\psi} h(w_{j,k}^{(i)}) \tag{6.5}$$

The second objective focuses more on link distances and the traveling distances during the mission. We want the links to be short, because long sensor links result in RSSI values that are less meaningful for the large volume of pixels that they intersect. The sensor link positions are provided to us in a set of pairs $P = \{(p_{1,1}, p_{1,2}), (p_{2,1}, p_{2,2}), \ldots, (p_{\ell_i,1}, p_{\ell_i,2})\}$. We also desire a short mission, which we determine from the distance T_i provided by an adapted greedy assignment algorithm from Section 4.2.1. We weigh these two distances into one objective using a parameter δ with $0 \leq \delta \leq 1$:

$$g_2(X_i) = \delta \cdot \left(\sum_{j=1}^{\ell_i} \|p_{j,1} - p_{j,2}\| \right) + (1 - \delta) \cdot T_i \tag{6.6}$$

We describe in more detail how the domain constraints and sensor positions are defined in Section 6.5.2, and how we obtain the traveling distance for Equation 6.6 in Section 6.5.3.

There are multiple reasons for having exactly two objectives instead of more. One reason is that it is easier to visualize a Pareto front for two objectives. Also, we have *conflicting* objectives that optimize toward different kinds of missions. The first objective can be improved with individuals that have lots of measurements with long-distance sensor links. The second objective attempts to reduce these instead.

Another reason is that the selection process requires more processing time when we have more than two objectives. To determine the solutions that are nondominated within the current population, we use the Kung, Luccio and Preparata (KLP) algorithm, which finds the maxima or minima of a set of vectors [22]. The complexity of the KLP algorithm for the two-dimensional problem is lower, compared to the three-dimensional variant.

Similarly, the selection for a population consisting of only nondominated solutions is also simplified. NSGA-II [13] and SMS-EMOA [16] are evolutionary multiobjective algorithms that determine a contribution measure of each individual toward the Pareto front. The implementations for both algorithms is simplified when only using two objectives.

6.5.2 Positioning problem

While we can describe the constraints and objectives in a simple manner in Section 6.5.1, we do need to transform the variables within the individuals so that we can actually calculate the weight matrix and the waypoint assignment. To do so, we also define what our variables actually mean and how they influence the result.

We consider two types of reconstruction planning problems: one *continuous* version where sensors can be positioned anywhere along the edges of the network, and a *discrete* variant where the sensors are placed at certain points that are evenly spread out around the edges.

In the continuous problem, we describe a sensor link with two variables: the distance of the link's line from the origin of the network, and the angle or slope that the line makes compared to the eastward axis.

Thus, if we want at most λ measurements and the network dimensions are $p \times q$, then each individual X_i has variables $b_1, b_2, \dots, b_\lambda$ with domain $[-\sqrt{p^2+q^2}, \sqrt{p^2+q^2})$, and $a_1, a_2, \dots, a_\lambda$ with domain $[0, \pi)$ in radians. Using one b_j and one a_j, we can define a line $y_j = \tan(a_j)x + b_j$. We consider two points on the line that intersect with the edges of the network to be the sensor points of the link. When $a_j = \frac{1}{2}\pi$, the line extends northward and b_j is the eastward coordinate value of the points.

If the two sensor points do not have correct positions, for example when the vertical offset is negative and the slope is more than $\frac{1}{2}\pi$, then this sensor link is *unsnappable*. If there are many unsnappable links, then the solution becomes infeasible. To help finding stable solutions, we add a binary variable c_j for each link in X_i. When the angle a_j is close to a cardinal direction and $c_j = 1$, then the line's angle becomes equal to that direction.

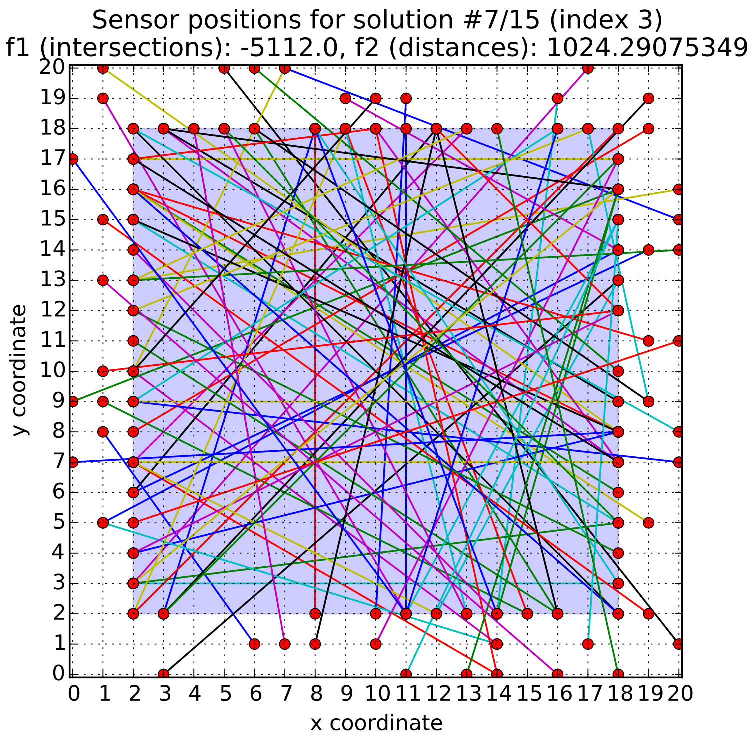


Figure 14: Example discrete solution with some padding.

The discrete problem uses four variables per link instead of three. There are two variables per sensor point, paired with another point to form a link. The variables encode northward and eastward coordinates from the origin, which are natural numbers within the domain of the network sizes. This includes any *padding* around the edges where we can also perform measurements. If a point ends up inside the network, then we attempt to place this point along the same line of the link outside the network, which may or may not end up somewhere within the padding. If the resulting points are not discrete, then we consider the link to be unsnappable.

In Figure 14, we show a discrete solution generated by the evolutionary algorithm, which makes use of padding. The resulting links appear to be quite chaotic, but they intersect with practically every pixel.

We can make the result more orderly by creating a mutation operator that works similar to the additional variable of the continuous version. This operator causes the links to prefer certain kinds of angles by moving one of the sensor points to another edge of the network.

6.5.3 Waypoint assignment

After we have deduced the positions of sensors as mentioned in Section 6.5.2, we can use them as input for other algorithms to find out how well the sensors perform. This includes passing the sensor pairs to the reconstruction algorithm to find out how much the reconstructed image pixels will be intersected. This intermediate step also makes use of the greedy waypoint assignment algorithm from Section 4.2.1 and the collision avoidance algorithm in Section 4.2.2 to find out how the vehicles can reach these sensor positions.

The implementation of these components does not differ much from the representation of the algorithms specified in those sections, but the data structures do. We do not make use of graphs to describe the area where the vehicles are located.

Instead, this map of obstacles is implemented using a matrix, known as the *memory map*. Each cell within the matrix describes a point in the space, so we can have one cell to an actual location for grid-based geometry defined in Section 3.1. For other geometries with a much larger volume of points within the physical space, we can adjust the *resolution* of the map. The resolution determines how many cells we have per meter for one dimension, and thus per square meter within one slice of the space. We can also define a region of influence for each cell that contains an object, marked with a nonzero value within the matrix. When the distance of a vehicle to such an object is less than the radius of a circular region of influence, then we consider it to be unsafe.

The unsafe areas in the memory map are removed from the possibilities that the search algorithm from Section 4.1 considers. During the collision avoidance algorithm, we also mark the inner edge of the network to be disallowed, so that the vehicles have to take a detour if they wish to move to another side of the network. The algorithm removes and reinserts the routes that unsynchronized vehicles traverse in a slightly different order, in an attempt to reduce the number of alterations to the memory map.

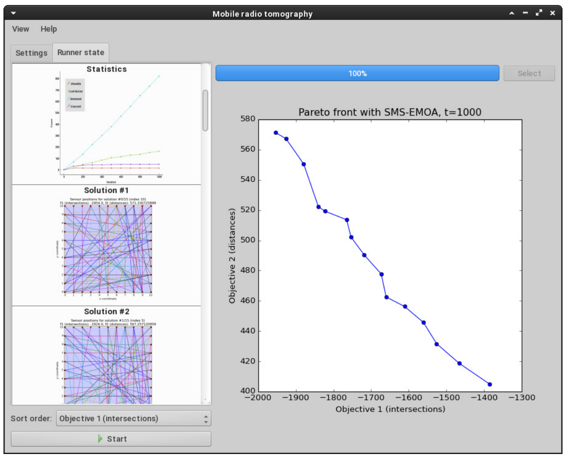
The collision avoidance planning algorithm adjusts the distance that the greedy waypoint assignment algorithm associates with a selected vehicle pair and sensor position pair. This however happens after the greedy algorithm has selected the pairs that minimize the greedy distance, i.e., when we have ϑ_m and ρ_m in Algorithm 4.2. Thus it will not improve the order of the assignment in which the vehicles move to the waypoints.

Still, the distance information is of use for the evolutionary algorithm from Section 6.5.1. The adapted greedy algorithm not only provides the assignment of waypoints, but also the total traveling distance, which is an indication of the duration of the entire mission. Such a measure is used for the objectives of the algorithm, so that we can improve our solutions to take less time or have fewer conflicting movements.

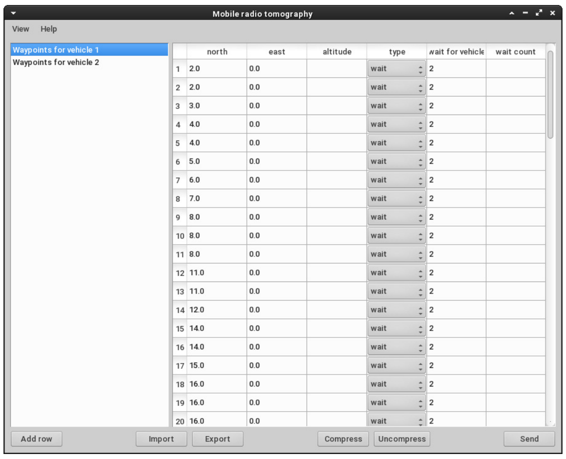
6.6 Ground station

A large portion of this thesis focuses on how the vehicles operate in the environment in order to collect measurements, with the goal of reconstructing what objects exist within some area of interest. The vehicles send signals to each other in order to achieve this goal. However, some parts of these intentions simply cannot be accomplished with just a few vehicles. For example, the planning algorithms of where the vehicles can be resource intensive, and thus needs a full computer to run.

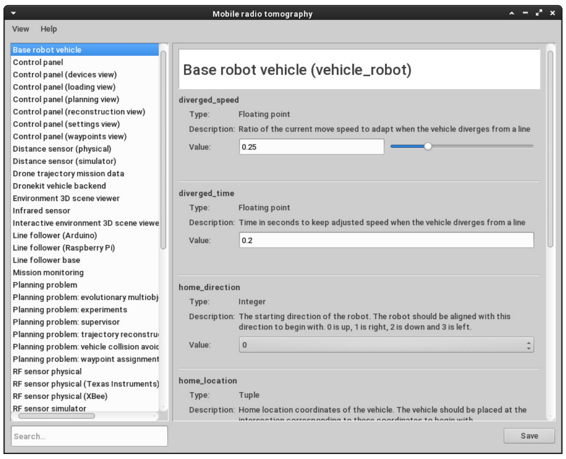
For this purpose, we implement a part of the toolchain which operates on the *ground station*, a machine that coordinates the collection of measurements. It provides an interface to a researcher to alter the state of the vehicles and track the progress of the reconstruction.



(a) Planning view, showing current state



	north	east	altitude	type	wait for vehicle	wait count
1	2.0	0.0		wait	2	
2	2.0	0.0		wait	2	
3	3.0	0.0		wait	2	
4	4.0	0.0		wait	2	
5	4.0	0.0		wait	2	
6	5.0	0.0		wait	2	
7	6.0	0.0		wait	2	
8	7.0	0.0		wait	2	
9	8.0	0.0		wait	2	
10	8.0	0.0		wait	2	
11	8.0	0.0		wait	2	
12	11.0	0.0		wait	2	
13	11.0	0.0		wait	2	
14	12.0	0.0		wait	2	
15	14.0	0.0		wait	2	
16	14.0	0.0		wait	2	
17	15.0	0.0		wait	2	
18	16.0	0.0		wait	2	
19	16.0	0.0		wait	2	
20	16.0	0.0		wait	2	


Mobile radio tomography
View
Help
Base robot vehicle
Control panel
Control panel (devices view)
Control panel (loading view)
Control panel (planning view)
Control panel (reconstruction view)
Control panel (settings view)
Control panel (waypoints view)
Distance sensor (physical)
Distance sensor (simulator)
Drone trajectory mission data
Dronekit vehicle backend
Environment 3D scene viewer
Infrared sensor
Interactive environment 3D scene viewe
Line follower (Arduino)
Line follower (Raspberry Pi)
Line follower base
Mission monitoring
Planning problem
Planning problem: evolutionary multiobj
Planning problem: experiments
Planning problem: supervisor
Planning problem: trajectory reconstru
Planning problem: vehicle collision avoic
Planning problem: waypoint assignment
RF sensor physical
RF sensor physical (Texas Instruments)
RF sensor physical (XBee)
RF sensor simulator
Search...
Save

Base robot vehicle (vehicle_robot)

diverged_speed
Type: Floating point
Description: Ratio of the current move speed to adapt when the vehicle diverges from a line
Value: 0.25

diverged_time
Type: Floating point
Description: Time in seconds to keep adjusted speed when the vehicle diverges from a line
Value: 0.2

home_direction
Type: Integer
Description: The starting direction of the robot. The robot should be aligned with this direction to begin with. 0 is up, 1 is right, 2 is down and 3 is left.
Value: 0

home_location
Type: Tuple
Description: Home location coordinates of the vehicle. The vehicle should be placed at the

(b) Waypoints view

(c) Settings view

Figure 15: Graphical user interface of the control panel on the ground station, showing the planning and configuration displays.

One part of this *control panel* is to manage a large volume of *settings*. The settings influence how the vehicles act in certain situations, or how the implementation of a certain algorithm is tuned. All settings are grouped by the component they belong to, and the control panel displays these in a sorted manner as shown in Figure 15c. We then store the revised settings locally or send them using the RF sensors to the vehicles.

Similarly, we can plan the missions using the control panel. There are in fact two methods with which we choose which waypoints we add to the mission. The planning algorithm is one of them. The interface in Figure 15a allows reviewing and tuning the parameters before starting the execution of the evolutionary algorithm. While this process is running, we regularly receive updates of the current status of the population that we are mutating. This includes the Pareto front of the nondominated solutions, as well as other statistics. Once the maximum number of iterations has been reached, we can compare the solutions visually by inspecting which sensor links are used in them. We can then choose one of the solutions to assign waypoints to the vehicles.

We can also manually assign waypoints, for example when we want to adapt an existing mission or perform other experiments. Similar to the settings, we can store the waypoints on the ground station for later use, and send ranges of waypoints in a compressed form to the vehicles when we are done assigning them. Figure 15b shows this interface.

6.7 Test coverage

In order to ensure that the vehicles operate in a way that is expected according to the initial design of the framework, we extensively test the implementations. This includes free-form experiments and high-level reliability checks, but also unit-based automated tests.

The testing framework consists of the tests themselves as well as a test bench, which evaluates the tests and provides statistics. We design the unit tests so that they correspond to parts of components in the framework. Our implementation is based on a object oriented programming (OOP) structure, where *classes* are the implemented parts. Each test class has an analogous actual class, whose *interface* is tested by individual test units.

The test units are separated in such a way that each one tests some part of the behavior of the actual class. The output is compared to the expected outcome of this behavior. The test unit can then fail if the result is not similar (enough), which means there is either a problem with the actual class, or with the test unit itself.

The test bench loads all the test units, executes them and reports the results. It can also include statistics about running time, code quality and code coverage. The *coverage* of a set of test units is a measure of how well much of the behavior of the actual class is tested by this set. The code coverage can be defined in multiple ways, and they usually take more details of the interface of a class into account.

The class interface consists of several *methods*, which are a means for other parts of the same component or other components to communicate with that class. This causes the class to change some state or take an action, such as ordering the vehicle to move in a certain way or calculating the outcome of a certain algorithm.

A method is made up of several *statements*, similar to (but usually more expansive than) the lines of an algorithm. The statements tell the computer which instructions it should perform, which should lead to the expected outcome of the method.

We consider two variants of code coverage: statement coverage and method coverage. In *statement coverage*, we look at each line of each method and see whether it is executed during a test. The coverage percentage is then the number of executed statements divided by the total number of statements. In *method coverage*, we attempt to match each test unit to one or more specific methods, and divide the number of matched methods by the total, including the unmatched ones. Aside from the coverage measure, the code coverage techniques can also determine which statements or methods are not yet covered. This allows us to alter the test units or add another one to cover more code [3].

Statement coverage might seem to provide more granularity in its coverage measure than method coverage. However, the statement coverage tracks all statements that are executed during the test, not just the ones in the method that the test unit claims to cover.

For example, a statement in a method is not covered by its own test unit, but is reached by another test unit. Such a statement may go undetected in statement coverage. Method coverage does detect the missing test unit. Thus a combination of both coverage measures helps in finding coverage deficiencies optimally.

Designing the test units to cover the statements may be difficult. The test unit should not be too granular, otherwise it is just repeating the low-level statements. A method may also be complicated, or it depends on other components. We can then *mock* some parts of the method so that those statements always perform a certain action. Thus we can get the program into a certain state and test whether this case works as expected. If there are more possible cases, then we mock multiple times to increase the statement coverage.

7 Experiments

We want to determine whether the implementation from Section 6 works well in practice. We objectively compare the results that we receive from a number of experiments.

The experiments focus on the planning and execution of missions that we provide to the vehicles. We look at how well such missions collect measurements that are used for a tomographic reconstruction algorithm. This includes objectives that predict how well the links cover the network. We also take other properties of the missions into account, such as how long it takes and how safe it is with regards to collisions between vehicles.

In Section 7.1, we describe how we set up these experiments, which consist of simulations, algorithm runs as well as physical tests. We provide an overview of the numerical results and comparisons in Section 7.2, and describe the outcome of two missions in Section 7.3.

7.1 Setup

During our simulations, we focus on the planning algorithms from Section 4.2 to see if our specific implementations mentioned in Section 6.5 perform as expected.

We find out whether the resulting individuals of the multiobjective optimization algorithm from Section 6.5.1 are a representative group of nondominated solutions. This means that they should converge toward near-optimal solutions according to our objectives. We also want to make our results reproducible. The constraints and objectives should lead the evolutionary algorithm into this direction, even though it is a stochastic process.

The evolutionary algorithm has a number of parameters that we can tune, which we summarize in Table 1. We look into the influence of the maximum number of iterations on the convergence. We also tweak the population size, and examine how the mutation operators affect the results.

We not only look at the evolutionary algorithm itself. We can make use of the results of the objective functions to see whether our formulation of the sensor positioning problem from Section 6.5.2 and the waypoint assignment implementation from Section 6.5.3 help in improving our solutions.

The positioning problem has its own configuration, such as the intended size of the network, the maximum number of measurements we want to perform, and other weighting and tolerance levels. The waypoint assignment is less tightly connected to the main objectives of the reconstruction planning, but its detection of potential collisions can still influence which results are acceptable.

This setup gives us a large number of experiments. There are 12 settings that are of interest, and we want to test between two and four different values for each setting. We also want to see what the result is of certain combinations of configurations. If we run the 70 possible combinations five times, then this results in 350 experiments.

Parameter description	**Domain**	**Tested values**
	Multiobjective algorithm parameters	
Number of iterations $t_{\max}$	$\mathbb{N}_{>0}$	1000, 5000, 10000*, 100000
Population size μ	$\mathbb{N}_{>0}$	10, 15*, 20
Algorithm	Boolean	NSGA-II, SMS-EMOA*
	Waypoint positioning problem parameters	
Network size	$\mathbb{N}^2$	10×10, $20 \times 20^*$
Network padding	$\mathbb{N}^2$	$0 \times 0^*$, 1×1
Positioning variant	Boolean	Discrete*, continuous
Number of measurements λ	$\mathbb{N}_{>0}$	50, 100*, 200
	Constraint and objective function parameters	
Ratio of unsnappable links $\frac{\gamma}{\lambda}$	$[0.0, 1.0]$	0.5*, 0.8
Weight of second objective δ	$[0.0, 1.0]$	0.2, 0.5*, 0.8
	Additional algorithm and operator parameters	
Specialized mutation operator	Boolean	Disabled, enabled*
Collision avoidance algorithm	Boolean	Disabled, enabled*
Penalty for unsafe paths	$\mathbb{R} \cup \{\infty\}$	0, 20, 40, ∞^*

Table 1: Parameters that we test in the experiments. We take all combinations of values within the same group, and otherwise use the default values listed with a star.

We use step sizes of 0.25 and 0.025 for the first two types of variables in the evolutionary algorithm. These parameters signify the standard deviation of a normal distribution used in the mutation operator. We determined that these values work well for the continuous version of the problem, where the variables signify the offset and slope of each line. This version also uses 0.25 as the probability of bit-flipping the third type of variable, which determines whether to change the angle to a cardinal direction when it is close to it.

Larger step sizes mean that the evolutionary algorithm can rapidly alter its individuals, which has the downside that it might step over an optimal solution. Thus these step sizes seem like a good middle ground. In Section 7.2, we describe the results for the experiments of the parameters in Table 1 rather than the tuning of the step sizes.

We also compare the missions provided by the planning algorithms with missions that we design by hand. Section 7.3 describes a physical setup and a qualitative analysis.

7.2 Planning results

We summarize the results of our experiments with the planning algorithms. Due to the large volume of results that we obtain, we only look at the results that signify interesting revelations of our algorithm, such as its performance, stability and effectiveness in finding good solutions for the sensor positions.

Because of the nondeterministic nature of the evolutionary multiobjective algorithm, it may produce different individual solutions between runs. Multiple runs with the same parameters allow us to retrieve the mean and standard deviation of the resulting values.

We use the KLP algorithm, which is the same sorting procedure used during the algorithm, to select an average knee point result from the objective values of one solution, calculated from Equations 6.5 and 6.6. These can be seen as the mean results of the experiment.

Because we minimize the objectives, lower values (toward negative infinity) are better, but the values may not be necessarily comparable between the two objectives. In Figure 16, we show the objective values of all average knee points of the 70 experiments. Here, the outlier "best" value in the bottom left is a run of the continuous variant and otherwise default values, which more easily optimizes the objectives. Other results that are part of the Pareto front of this scatter plot respectively use a smaller network size, fewer measurements, and the highest value tested for the penalty for unsafe paths, which improves the coverage of intersecting links but has an average objective value for the total distances.

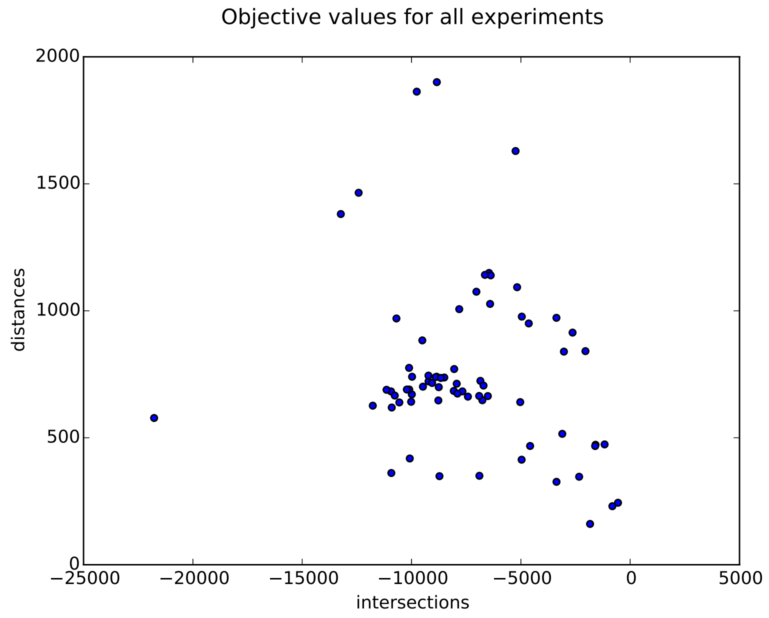
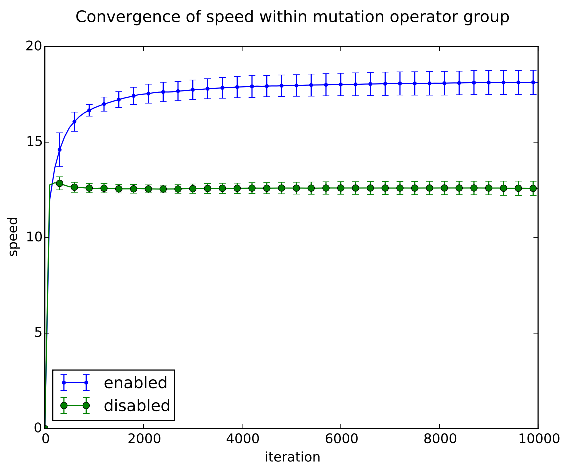


Figure 16: Overview the objective values of the average knee points of all experiments.

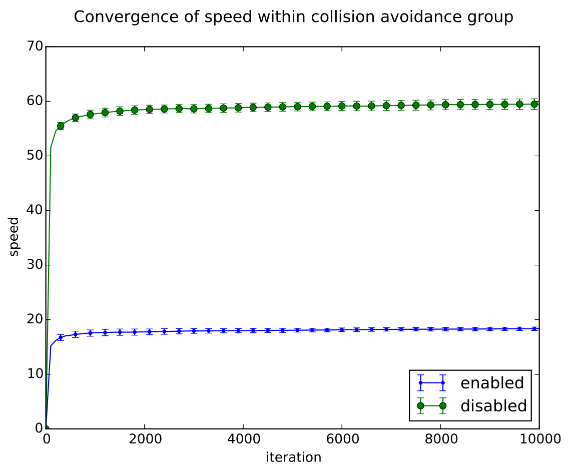
We now look at some parameters used within the experiments. The specialized mutation operator that we describe in Section 6.5.2 performs certain actions that are different from the randomization of a normal distribution, so we want to compare how fast and useful these operations are.

The number of iterations per second or "speed" of the algorithm is shown in Figure 17a, where we show a run with the specialized mutation operator enabled and a run where it is disabled. Surprisingly, the specialized mutation operator appears to require less time to mutate the dependent variables, compared to the usual mutations. The reason for this speedup is unclear; the artificial placement of sensors might simplify the work of other algorithms, such as the greedy assignment.

In Figure 18, we show the objective values of the average knee point at different iterations, in a run with 100000 iterations. We observe that the specialized mutation operator helps in decreasing the objective values, although it eventually results in unstable knee points.

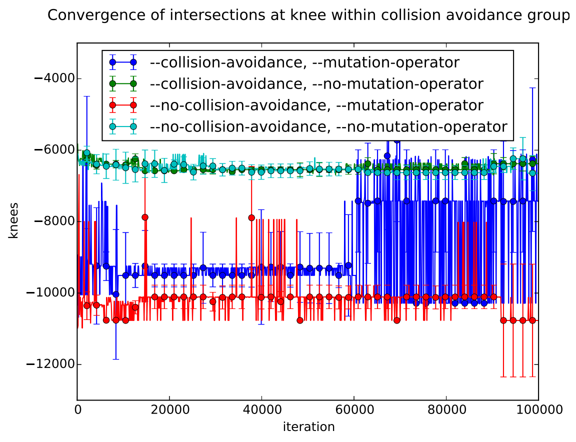


(a) Specialized mutation operator

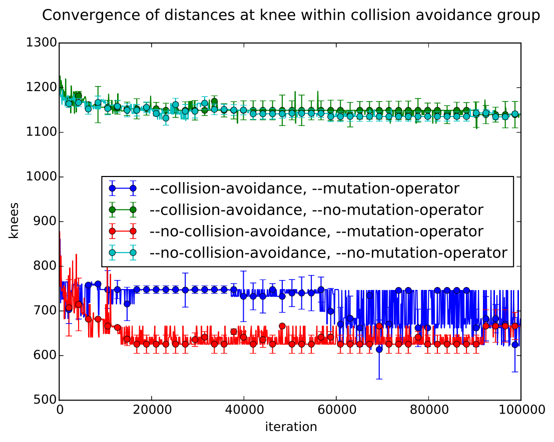


(b) Collision avoidance

Figure 17: Convergence of speed when parts of the algorithm are activated, measured in iterations per second.



(a) First objective: intersections



(b) Second objective: distances

Figure 18: Convergence of objective values, with various parts of the algorithm activated.

The collision avoidance algorithm from Section 4.2.2 also appears to work, in that it determines whether an assignment of waypoints is safe. However, its effectiveness within the evolutionary algorithm is limited. The collision avoidance algorithm rejects routes that conflict anywhere, but such solutions can be stumbled upon anywhere in the search space. We thus reject individuals which could lead to better results if we slightly mutate them.

As shown in Figure 17b, enabling the collision avoidance algorithm significantly reduces the speed of the entire planning algorithm. In the runs shown in Figure 18, there are no great differences between objective values when we enable or disable the collision avoidance algorithm. It thus hinders the performance without clear advantages.

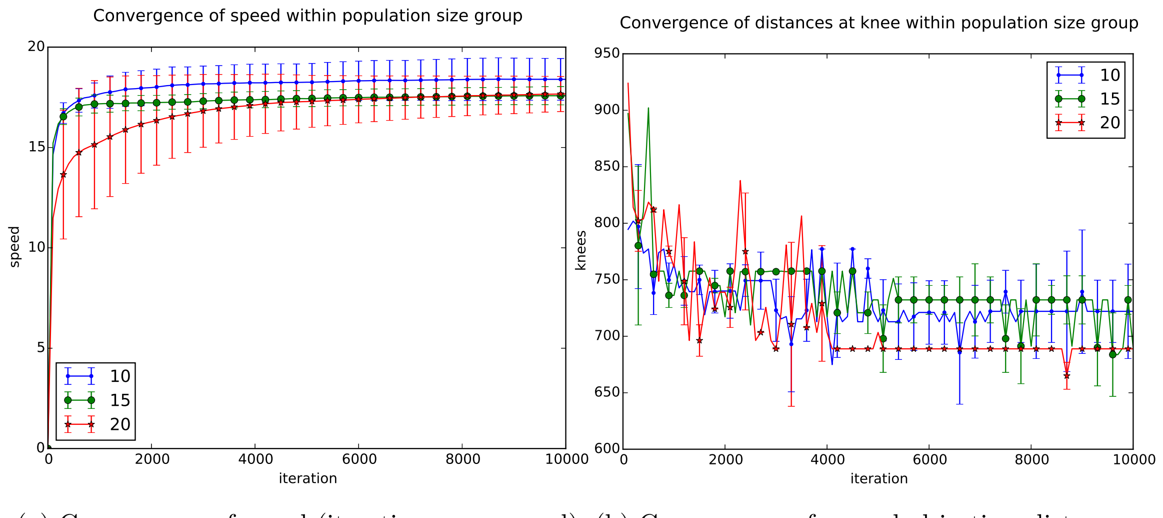
We can alter the algorithm to penalize unsafe solutions with an additional cost within the second objective, instead of outright rejecting them. Even so, this only has an advantage if we have space for detours. It might be more favorable to run the collision avoidance algorithm afterward to detect unsafe solutions.

The greedy assignment itself, using an adapted version of Algorithm 4.2, provides a useful measure of mission length. We use this within the second objective related to distances. Another factor of the second objective is the sum of the lengths of all sensor links.

The two factors of the second objective are combined using a weight δ from Equation 6.6. In Table 2, we see that both objective values decrease when δ is low, such as 0.2, which causes it to assign more weight to the mission length instead of the sensor link lengths. This might be because the sensor links are the basis for both objectives, thus we need to assign more weight to the travel distance to optimize both objectives.

δ	g_1: intersections	g_2: distances
0.2	-10080.0 ± 511.0	418.4 ± 4.2
0.5	-8851.0 ± 310.7	740.3 ± 18.2
0.8	-10691.0 ± 0.0	969.7 ± 0.0

Table 2: Comparison of knee objective values between values of the weight δ, where lower values are better.



(a) Convergence of speed (iterations per second) (b) Convergence of second objective: distances

Figure 19: Influence of the population size on the evolutionary algorithm's behavior.

The algorithm attempts to optimize both objectives, but it has a limited work space to do so. One solution is to add more individuals to the population. The speed of such a run, measured in number of iterations per second, ends up to be similar to runs with small populations when they run for a long time, as shown in Figure 19a. This is because the algorithm always mutates one individual per iteration, and other steps have a linear complexity in terms of number of individuals.

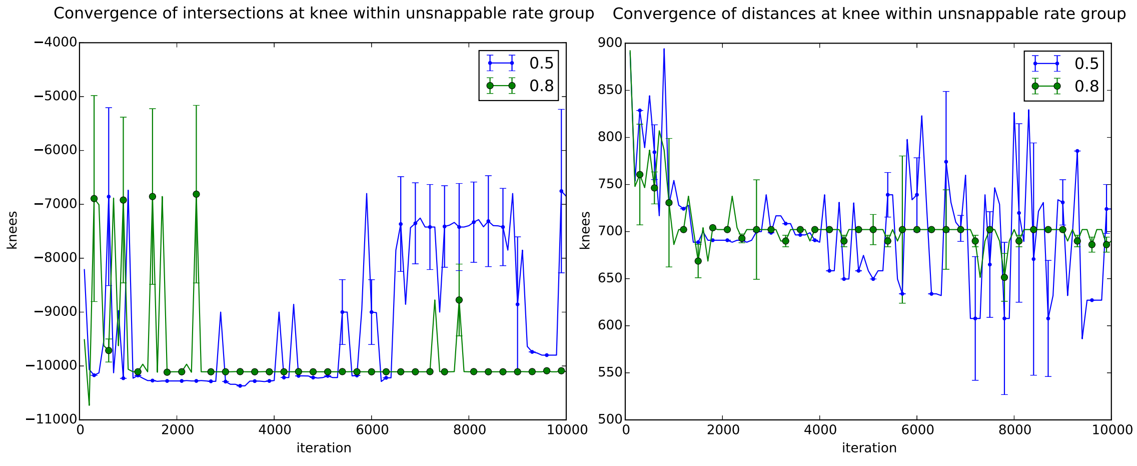
The use of a population with more individuals only slightly decreases the objective values of the knee points in Figure 19b. Having more individuals means that we have more possible solutions to choose from, but it does not necessarily provide better ones.

A more pressing constraint is the number of measurements that we want to achieve. We need more measurements to create more complicated missions and to visit more sensor positions for the reconstruction, but this requires more variables within each individual. The addition of more variables greatly reduces the speed of the algorithm. This means that the effectiveness of our algorithm to search the entire search space is limited.

Of course, the possible measurements that we can perform is limited if we use discrete points for them. When we use the continuous version of our problem, then we are able to find better solutions, at least according to the objectives that we minimize in Table 3. This version is possibly even more sensitive to the configuration and the stochastic process, because it often creates measurements that have difficulties with crossing the network.

Variant	g_1: intersections	g_2: distances
Discrete	-7933.0 ± 1241.4	712.8 ± 52.5
Continuous	-21773.0 ± 117.7	577.8 ± 21.8

Table 3: Comparison of knee objective values between discrete and continuous variants.



(a) First objective: intersections (b) Second objective: distances

Figure 20: Convergence of objective values for different unsnappable rates.

Still, under the right circumstances, we may receive interesting results from the algorithms. We can alter the threshold at which we accept an individual that has a number of links that do not cross the network. The *unsnappable rate* determines the percentage of measurements that must cross at least one pixel. This rate is the value $\frac{\gamma}{\lambda}$, where γ is the threshold mentioned before in Section 6.5.1, and λ is the maximum number of measurements.

One would expect a low unsnappable rate to allow way too many inferior results, but we observe from Figure 20 that this may allow the algorithm to take more risk, resulting in deviating runs. A high rate results in stable knee points which end up to be relatively good, but do not improve. Thus, we find that selecting the right set of parameters is difficult and very dependent on context, but the algorithms may give us good results if we do find the right values.

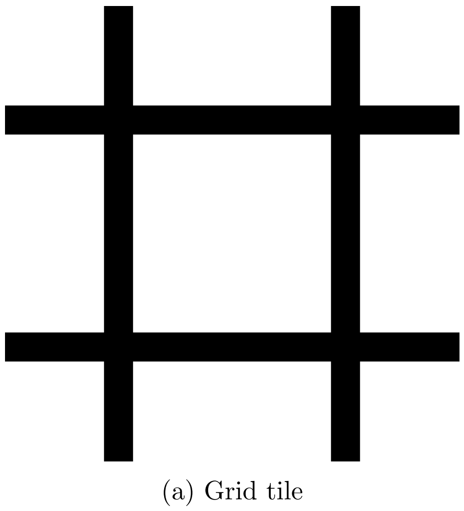
7.3 Physical trials

After delving into the planning of missions, we want to actually put the missions into play. We let various missions collect their measurements and observe which ones result in a smoother reconstructed image of the same environment of objects.

We also compare the missions based on their running time and number of times that two vehicles are too close to each other according to an objective proximity factor. This gives us a measure of how safe a given mission is.

First of all, we describe how we set up the controlled space that we use for these physical experiments. We use an indoor location that is enclosed by unused rooms, and the experiment space is also mostly empty. This minimizes most forms of noise from the environment that could disturb the RF sensor measurements.

When we are indoors, location detection systems such as GPS function quite poorly, which we also determined empirically for the building where the experiments are located. Due to this, the vehicles need a different method of detecting its position. The vehicles have a line follower sensor, which we describe in Section 6.3.2. With it, we can follow black lines on a white surface. We use tiles that can be printed on A3 paper that give us intersections of lines that are 19 mm thick and 130 mm apart when they are cut out and repeated. These tiles are shown in Figure 21a.


(a) Grid tile

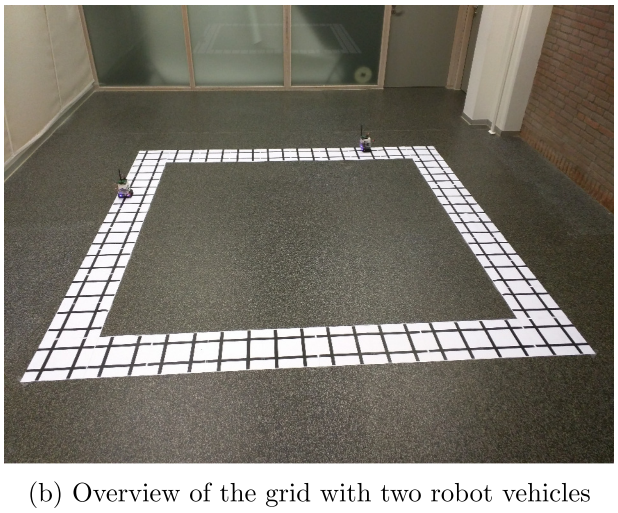

(b) Overview of the grid with two robot vehicles

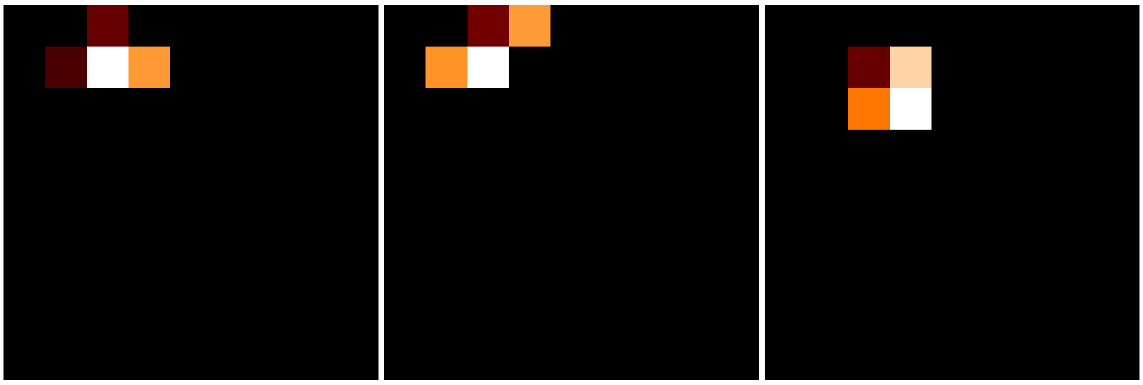
Figure 21: The experimental setup of the grid where the vehicles may move.

The tiles need not fit exactly, as one can leave some white space between the lines by letting the vehicles take some leeway when the line follower sensor loses the line. The intersections are then still at equal distances from each other.

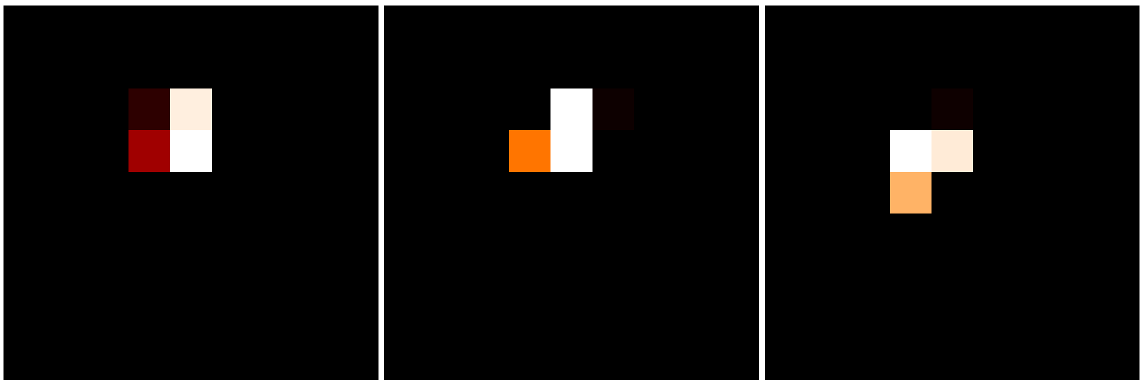
The grid pattern resulting from the tiles has the advantage of clearly denoting the “pixels” of the area corresponding to the reconstructed tomographic image. Thus we can check whether the image shows the objects in their correct locations.

For our physical experiments, we create two grids: one full grid with 10 by 10 pixels, and one grid that is 20 by 20 in total but only has pixel lines at the edges of the area. This gives us two horizontal or vertical lines per edge, depending on the direction of the edge. The latter grid is shown in Figure 21b. We use two vehicles in all experiments.

We perform some preliminary experiments on the smaller 10×10 grid, using different kinds of missions. All of these missions employ the same kind of waypoint patterns, which we describe in Section 6.4.1. This includes straight lines, fan beams originating from corners or from the center of an edge, diagonal lines and other line patterns where the robot vehicles are driving along different edges. We attempt to connect these patterns so that there is as little downtime between measurements; this leads to a full mission with a specific order of patterns.



(a) Top left; straight lines mixed with corner fan beams (b) Top left; straight lines mixed with center fan beams (c) Top left; straight lines, center fan beams and corner fan beams



(d) Center; straight lines mixed with corner fan beams (e) Center; straight lines mixed with center fan beams (f) Center; straight lines, center fan beams and corner fan beams

Figure 22: Visualizations of tomographic reconstructions for one person standing at two different locations within a 10×10 network, using various patterns in missions that collect the measurements. We observe that (c) and (f) supply the most realistic and stable results.

The quality of the result may depend on where the objects are located in the network; some areas may receive many intersecting measurements pretty quickly, while others take a long time to settle. Therefore, we perform two experiments for each of the three missions that we compare.

In the first experiment, a person is standing in the top left corner of the network. All missions start the vehicles at the lower edge of the network, which means that they need to travel longer to get good coverage in this corner. The second experiment has one person standing in the center of the network for further comparison.

We show these final visualizations in Figure 22. We observe that there are qualitative differences in the reconstructed image for these fairly similar missions, when we run them under the same circumstances.

Some tomographic reconstructions take longer to provide an image that is visually similar to what we expect. During this time, they only show unstable reconstructions. This is because there are not enough sensor measurements in certain areas, or the links do not cross each other enough at that point.

Eventually, the mission that combines straight lines, fan beams from the center of the edges, and fan beams from the corners, in that order, quickly supplies the most stable result locations of objects that we testes, which we observe from the reconstructed images.

We now create a mission using the planning algorithm and compare it to the hand-made mission. We allow the algorithm to place sensors at discrete locations for up to $\lambda = 400$ measurements, and require that at least 80% of them are correctly positioned such that they cross the network. After $t_{\max} = 7000$ iterations, we end the run, which gives us a Pareto front. We pick a knee point, in this case the seventh solution in the front.

This solution performs 382 measurements. Using ranges of waypoints, we can compress this mission to about 70% of the original size of the assignment of waypoints. The collision avoidance algorithm determines that this assignment is safe, but we do not make use of the padding. We visually confirm the safety using a tool that shows in which order we receive the links from a predefined assignment.

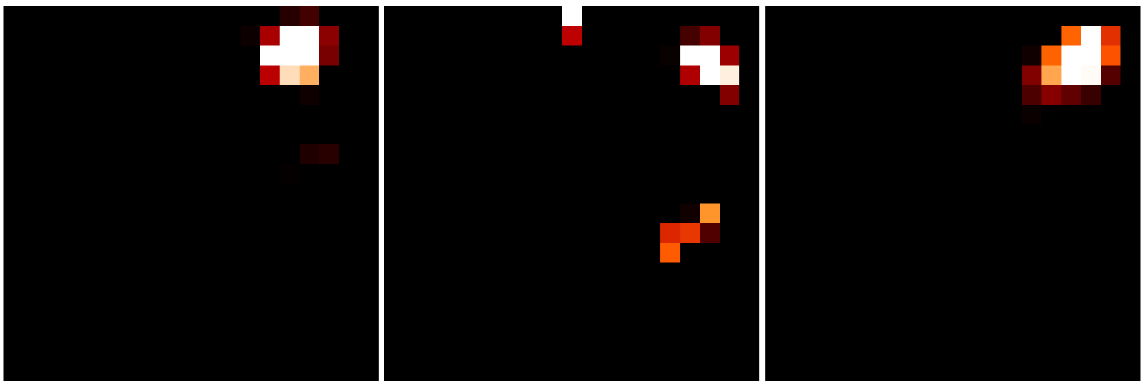

(a) Partial result from mission that was automatically planned (b) Partial result from mission that was hand-made (c) Final result from mission that was hand-made

Figure 23: Visualizations of tomographic reconstructions for one person standing in the top right corner of a 20×20 network, during a planned mission and a hand-made mission.

When we run the planned mission, we obsertve about halfway through that Figure 23a distinctly shows the person who is standing in the top right corner. In comparison, the hand-made mission would finish faster for the 20×20 grid, but halfway through the reconstruction still shows some “ghosts” of objects that are not in the area, which can be seen in Figure 23b. The final result in Figure 23c does show the object distinctly.

8 Conclusions

In this thesis, we introduced mobile radio tomography as a novel collection of techniques that can be used to deploy a network of wireless sensors in a new location without much prior information about the environment. We describe the foundations of geometry and other concepts that are relevant for this purpose. This allows us to make use of existing algorithms and augment them with new ones. These algorithms help us plan a mission consisting of positions where we need to take sensors to, assign these positions to various vehicles and ensure that they do not collide with each other.

The algorithms form the basis of a planning component within a larger toolchain, which is able to direct the vehicles to move to the correct directions and oversee the entire mission. We make use of information from auxiliary sensors, which allows the vehicle to perform measurements, communicate with other vehicles as well as the ground station, receive commands that the vehicle processes immediately in any circumstance, and keep track of its location and other status information.

The complete toolchain allows us to experiment with our mobile radio tomography setup. We look at the parameters that determine how the planning algorithms work, and also test the missions in an actual setting, keeping in mind that our goal is to quickly receive a stable and good reconstructed image of the area.

From our results of the large number of experiments with the planning algorithm, it appears that we need to carefully tune the configuration of these algorithm for them to function optimally. The evolutionary multiobjective optimization algorithm has problems with finding better results. This may also be caused by the objectives themselves, i.e., they are not easily optimizable because the underlying functions are too complicated, or they do not provide a good measure of quality.

Additionally, the algorithms that are used to generate these measures work well on their own, but they are slow and do not provide the expected end results when used within the evolutionary algorithm. We speculate that this is due to the additional constraints laid down by these algorithms, such as the collision avoidance algorithm rejecting or penalizing solutions that are unsafe. The evolutionary algorithm can then unexpectedly find worse solutions and is not able to salvage this situation.

The greedy assignment algorithm works well for a small number of pairs of sensor positions. When we need to obtain over 200 measurements in order to receive a good reconstruction, then the algorithm still functions the way it is supposed to, but certain movement patterns are not sensible for human observers.

One problem is that the travel distance determined by this algorithm is based on the sum of the lengths of the shortest (safe) routes, and does not contain other factors that increase the duration of the mission. Still, the contribution of the greedy algorithm toward the optimization problem and final assignment is considerable.

Indeed, when we do find good parameter values, the planning algorithm is able to provide intriguing results. As seen in Section 7.3, the automatically planned mission is able to compete with a hand-made mission, which follows certain patterns that were theorized and tested to serve well for the tomographic reconstruction. The speed of the mission is still an issue, but the quality of the reconstructed image shows that it is possible to put an automatically planned mission into practice.

The vehicles make use of the line follower sensor to detect their location in a discrete grid setup. This makes it possible to take a preprinted partial grid, which might consist of only "rail tracks", and place it in a new environment as a simple, controlled setup.

We let the vehicles follow their own mission autonomously. This independence does not mean that they cannot take the other sensor-carrying vehicles into account. This allows them to avoid collisions, and lets them synchronize the collection of measurements. Also, a ground station is a necessity, but only for the planning and reconstruction work. Finally, one needs to supervise the entire setup to ensure that everything works as intended. Still, a lot of the work is automated, which is pleasing especially for the more laborious and repetitive tasks.

We find that the overall approach of mobile radio tomography is successful in obtaining a reconstructed image of an area with fairly little prior knowledge about its properties, using far fewer sensors and measurements that one would need in a static setup. We show that the addition of various algorithms and other sources of information can help in making this approach more stable.

In conclusion, we can split up mobile radio tomography into different parts, which all work the way we expect them to. Even though it may be problematic to achieve the expected result in some cases, these difficulties are not insurmountable. These parts allow us to create a fully functional toolchain, which leads to the desired end result of reconstructed images that clearly show the location of objects within the area of interest.

8.1 Further research

While we present a toolchain for mobile radio tomography in this thesis which accomplishes our basic needs, this does not necessarily mean that this field of research is completely explored. We have several ideas, theories and proposals for additional features that can be helpful to improve the stability, quality and overall usability of mobile radio tomography. We focus on the topics related to unmanned vehicles, but this also includes novel reconstruction work.

One particularly interesting concept is 3D tomographic reconstruction. If we make use of more than two axes in our space, then we can gather measurements from many more locations. This means that the vehicle must be able to fly around or otherwise change the altitude of its sensor, e.g., through the use of a telescoping pole mounted on a robot. We could perform measurements that intersect with objects detected earlier on even more, by placing sensors at different altitudes and angles.

A simplification would be to measure similar links at *slices* of the space, each slice having one altitude that is equal between all measurements in this slice. The slices can then be stacked, and we then connect the detected object pixels, which are actually three-dimensional *voxels*. This would make it possible to display an interactive 3D visualization of the reconstruction.

The level of detail of the reconstructed images is not only limited by the number of discrete locations where we perform measurements, but also by the granularity of signal strengths, which is inherent to the wavelength and power of the wireless antennas. It may thus be problematic to improve the resolution of the reconstructed images, but there could still be opportunities that can enrich the reconstruction with more detailed information.

One option is to scan the entire network with an initial reconstruction. After enough measurements are received, we obtain knowledge about where the objects of interest are located. We could then “zoom in” on one of the objects by moving the vehicles closer to this object, but far away enough to move around it. We can then start another reconstruction of this object, where we decrease the distances between sensors proportionate to this new subnetwork.

The algorithms that we propose in this thesis can also be improved. The search algorithm from Section 4.1 that we use, is based on the A* search algorithm. Variants exist that improve performance and decrease complexity for specific domains. This could be used to make the application of the other algorithms that use the search algorithm more viable. The collision avoidance algorithm from Section 4.2.2 could be made less strict by modeling what a vehicle could do to prevent a collision, such as halting or making a shorter detour than it can detect right now.

The greedy algorithm, which assigns sensor positions to different vehicles in the form of waypoints, could also be extended further. The algorithm that we describe in Section 4.2.1 performs a selection that is not always optimal. The greedy algorithm could take more information into account to make a better decision while keeping the same time complexity. We could track the direction in which the vehicle is facing at every point, so that we can favor moving in this direction rather than turning to go to a position behind it.

The evolutionary algorithm could also be enhanced with objectives that better fit our desires. We could improve the sensor positions by defining variables that create different types of waypoints or complete patterns, such as straight lines and fan beams from certain center points. We can use the hand-made missions as a baseline to optimize further, and use it as a reference point in our Pareto front for different selection strategies.

In our physical setup, one would rather have the vehicles find out their location on their own, with no knowledge of the area at all. This excludes the grid overlay, which lets the vehicles position their sensors at precise but fixed locations. The grid restricts the movement to certain directions, even when diagonal movement would be faster, for example.

A major bottleneck here is that the reconstruction requires precise location information, and relying on external positioning information is too fragile. A solution that uses a global positioning system (GPS) would not work reliably when there is no open air, and even when outdoors it may be inexact and shifting the position incorrectly.

Other location detection systems such as ultrasound or infrared sensor information may be possible but require additional setup or assumptions about the environment. Another option is letting the vehicles determine their location through the use of the tomographic measurements themselves. When there are enough sensors in our network, and we have some baseline information such as initial positions and a realistic speed indicator, then a triangulation approach could be possible.

Finally, we mostly consider vehicles that can freely move around in our conception of mobile radio tomography. If we want a permanent setup with fewer sensors, then we can also use other robotic systems to transport the wireless sensors. One could consider a guidance rail mounted against a wall which holds sensors in certain positions and drags them around to measure different links. This gives us the freedom to position sensors wherever we want along these rails, without requiring a large number of them or depending on completely independent, battery-powered vehicles.

List of figures

The following figures are included in this thesis. The actual figure can be found on the corresponding page. We also specify the source of the work here; unless otherwise specified, the figures are own work.

[1] `http://www.texample.net/tikz/examples/dome/`
[2] `http://tex.stackexchange.com/a/117175`
[3] `http://tex.stackexchange.com/a/182998`
[4] `https://en.wikipedia.org/wiki/File:Traxxas_t-maxx.no_body.triddle.jpg`

List of tables

List of algorithms

References

[1] 3D Robotics. DroneKit. `http://dronekit.io/` (accessed July 20, 2016).

[2] C. Alippi et al. *RTI goes wild: Radio tomographic imaging for outdoor people detection and localization.* IEEE Transactions on Mobile Computing 99 (2015). DOI: `10.1109/TMC.2015.2504965`.

[3] P. Ammann and J. Offutt. *Introduction to Software Testing.* Cambridge University Press, 2008.

[4] Arduino. `https://www.arduino.cc/` (accessed August 19, 2016).

[5] ArduPilot developers. Open source autopilot. `http://ardupilot.com/` (accessed July 20, 2016).

[6] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.

[7] T. Bäck and H. Schwefel. *An overview of evolutionary algorithms for parameter optimization.* Evolutionary Computation 1 (1993), pp. 1–23. DOI: `10.1162/evco.1993.1.1.1`.

[8] N. Balakrishnan and V. Nevzorov. *A Primer on Statistical Distributions.* Wiley & Sons, 2004.

[9] T. Bektas. *The multiple traveling salesman problem: An overview of formulations and solution procedures.* Omega 34.3 (2006), pp. 209–219. DOI: `10.1016/j.omega.2004.10.004`.

[10] J. Branke, K. Deb, H. Dierolf, and M. Osswald. *Finding knees in multi-objective optimization.* In: Proceedings of the International Conference on Parallel Problem Solving from Nature (LNCS 3242). Springer, 2004, pp. 722–731. DOI: `10.1007/978-3-540-30217-9_73`.

[11] D. Bredström and M. Rönnqvist. *Combined vehicle routing and scheduling with temporal precedence and synchronization constraints.* European Journal of Operational Research 191.1 (2008), pp. 19–31. DOI: `10.1016/j.ejor.2007.07.033`.

[12] H. S. M. Coxeter. *Introduction to Geometry.* Wiley, 1961.

[13] K. Deb. *Multi-objective Optimization using Evolutionary Algorithms.* Wiley & Sons, 2001.

[14] F. Ducatelle et al. *Cooperative navigation in robotic swarms.* Swarm Intelligence 8.1 (2014), pp. 1–33. DOI: `10.1007/s11721-013-0089-4`.

[15] Emlid. Navio+. `http://www.emlid.com/` (accessed July 20, 2016).

[16] M. Emmerich, N. Beume, and B. Naujoks. *An EMO algorithm using the hypervolume measure as selection criterion.* In: Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (LNCS 3410). Springer-Verlag, 2005, pp. 62–76. DOI: `10.1007/978-3-540-31880-4_5`.

[17] C. Fowler. *The Solid Earth.* 2nd edition. Cambridge University Press, 2004. DOI: `10.1017/CBO9780511819643`.

[18] B. L. Golden, S. Raghavan, and E. A. Wasil. *The vehicle routing problem: Latest advances and new challenges.* Vol. 43. Springer Science & Business Media, 2008.

[19] L. Helwerda. "Mobile radio tomography: Object detection using autonomous unmanned vehicles". Master's research project report, Leiden University. 2016.

[20] P. Hillyard et al. *Through-wall person localization using transceivers in motion.* CoRR 1511.06703 (2015). `http://arxiv.org/abs/1511.06703`.

[21] O. Kaltiokallio, M. Bocca, and N. Patwari. *Enhancing the accuracy of radio tomographic imaging using channel diversity.* In: Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS). 2012, pp. 254–262. DOI: `10.1109/MASS.2012.6502524`.

[22] H. T. Kung, F. Luccio, and F. P. Preparata. *On finding the maxima of a set of vectors.* Journal of the ACM 22.4 (1975), pp. 469–476. DOI: `10.1145/321906.321910`.

[23] G. Laporte. *The vehicle routing problem: An overview of exact and approximate algorithms.* European Journal of Operational Research 59.3 (1992), pp. 345–358. DOI: `10.1016/0377-2217(92)90192-C`.

[24] L. Meier. MAVLink Micro Air Vehicle Communication Protocol. `http://www.qgroundcontrol.org/mavlink/start` (accessed July 20, 2016).

[25] T. van der Meij. "Constructing an open-source toolchain and investigating sensor properties for radio tomography". Bachelor's thesis, Leiden University. 2014.

[26] T. van der Meij. "Mobile radio tomography: Constructing an open-source framework with wireless communication components". Master's research project report, Leiden University. 2016.

[27] T. van der Meij. "Mobile radio tomography: Reconstruction and visualization of wireless sensor networks with dynamically positioned sensors". Master's thesis, Leiden University. 2016.

[28] T. van der Meij and L. Helwerda. Open-source mobile radio tomography framework. `https://github.com/timvandermeij/mobile-radio-tomography` (accessed August 19, 2016).

[29] A. Milburn. "Algorithms and models for radio tomographic imaging". Bachelor's thesis, Leiden University. 2014.

[30] National Geospatial-Intelligence Agency. *Department of Defense World Geodetic System 1984, its definition and relationships with local geodetic systems.* Technical report. NIMA TR8350.2. 1997.

[31] Pololu. Zumo Robot for Arduino. `https://www.pololu.com/product/2510` (accessed August 17, 2016).

[32] Raspberry Pi. `https://www.raspberrypi.org/` (accessed August 26, 2016).

[33] S. J. Russell and P. Norvig. *Artificial Intelligence: A modern approach.* Third edition. Prentice Hall, 2010.

[34] H. Schaub and J. Junkins. *Analytical mechanics of space systems.* AIAA Education Series. American Institute of Aeronautics and Astronautics, 2003.

[35] B. Wei et al. *dRTI: Directional Radio Tomographic Imaging.* In: Proceedings of the 14th International Conference on Information Processing in Sensor Networks (IPSN). ACM, 2015, pp. 166–177. DOI: `10.1145/2737095.2737118`.

[36] J. Wilson and N. Patwari. *Radio tomographic imaging with wireless networks.* IEEE Transactions on Mobile Computing 9.5 (2010), pp. 621–632. DOI: `10.1109/TMC.2009.174`.

[37] S. Winter. *Modeling costs of turns in route planning.* GeoInformatica 6.4 (2002), pp. 345–361. DOI: `10.1023/A:1020853410145`.