B. Sc. Dirk Steindorf

Extended Robot Navigation Using Dynamically Generated Occupancy Grids

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

INF FAKULTÄT FÜR INFORMATIK

Intelligent and Cooperative Systems Department

Master's Thesis

Extended Robot Navigation Using Dynamically Generated Occupancy Grids

Author:	B. Sc. Dirk Steindorf
Supervising Professor:	Jun.-Prof. Dr.-Ing. Sebastian Zug
Reviewing Professor:	Prof. Dr.-Ing. habil. Sanaz Mostaghim
Tutor:	Dipl.-Inform. Christoph Steup

Summer Term   2016

B. Sc. Dirk Steindorf: *Extended Robot Navigation Using Dynamically Generated Occupancy Grids*
Otto-von-Guericke-University
Magdeburg, 2016.

Contents

List of Figures

List of Tables

Table of Acronyms

ROS	Robot Operating System
MIRA	Middleware for Robotic Applications
ASEIA	Abstract Sensor Event Information Architecture
SLAM	Simultaneous Localization and Mapping
AMCL	Adaptive Monte Carlo Localization
HMI	Human-Machine-Interface
CPU	Central Processing Unit

1 Introduction

Today, robots used in many areas where processes can be automated or where the situation would be too dangerous for humans to operate. They are used to support the construction of big machines, cars, or planes. At the same time, they are used to build tiny things with high precision that a person could not attain [Her]. Robots can repeatedly carry out the same task without getting bored by the task, distracted by the environment, or tired from exhaustion. Looking at these abilities, robots seem to be ideal working machines to carry out heavy construction tasks. Yet, they are still used in other fields outside of manufacturing. They are also used to explore places that are dangerous or otherwise unsuitable for humans. For example, diving robots are used to explore the deep sea, where humans would have problems with the high pressure. In addition to this, a huge driving factor for development in robotics are military applications, where robots can be utilized in various different scenarios. They can be simply used as transport vehicles [Dyn], for detecting and defusing bombs, autonomously building swimming platforms for helicopters to land on, or simply for scouting or observation missions.
But there is one growing field where robotics is established that has nothing to do with heavy machinery, research in remote areas, or war scenarios. Meant is the increasing use of robots in everyday situations, where they can act as a helping hand in households where they can already be found as cleaning robots, dusting the floor when nobody is home [iRo].
In the context social situations the usage of robots has been discussed more often recently. There exist projects to utilize robots for elder care [Sta] and it has been discussed to use robots as a teaching device for refugees to teach them the German language [Mat15].
One area where robots have been existent for some time is in entertainment as toys that can dance, blink, talk, or even fight each other. In short, robots are used in many different areas of everyday life, work, military, and research and their popularity is growing, judging by the increasing number of articles written about them.

A big percentage of robots works on different tasks in indoor environments, if it is cleaning the house, getting items in an Amazon warehouse [Hea]. The focus of this thesis will be on the application of autonomous robots in such indoor scenarios, and more specifically in indoor environments where some kind of dynamic aspects are present, like people that share the same space with the robots. This situation leads to interesting challenges for the robots. While they are supposed to support processes in their respective environment, they also have to make sure that they don't accidentally interrupt them by colliding with each other or by hurting people that work with them.

One task these robots have to carry out is the navigation in a given environment, where they have to be able to move through the environment autonomously without colliding with other robots or people. To ensure this, they are equipped with different sensors that enable the robot to see what happens in front of them. This, however, still leaves a gap in their perception when it comes to changes in areas that they can not observe. The approach described in this thesis aims to extend the perceptive capabilities of the robots in order to allow them to consider distant changes for planning their behaviour. With this, they are supposed to be able to notice when a passage is blocked and automatically take another route to their destination, or to notice when a person is close to the robot so it moves more carefully in order to avoid accidentally hurting this person.

Robotics has been a research topic for some time now, so there exist methods to deal with all kinds of situations that are needed to be able to handle environments with dynamic changes. Some of these existing approaches are covered in the next chapter (chapter 2), where an overview of robotic navigation is given and what can be used in order to enable robots to get information about the whole environment. The concept chapter (chapter 3) covers how these fundamentals can be applied and how they have to be extended to get a working system. The implementation of this approach and its performance in a scenario is described in chapter 4, which shows if this approach works as intended and in which areas it can be improved. The last chapter (chapter 6) summarizes this thesis and gives an outlook on possible future work.

2 Technical Background

In order to provide a new solution to the problem of dynamic objects in robotic maps, a few other problems had to be solved. These include the control of robots, their localization and navigation, the usage of sensor networks, communication between robots or nodes of a network, and more. This chapter will cover the description of some of the solutions that will be used later on for the approach described in this thesis. Some of these will be used directly, while others will be used with small modifications when the situation demanded it.

Sensor networks are covered, because the approach described in this thesis makes use of a heterogeneous sensor network. This network is used to gather information about the environment, that will then be added to a map which robots can use for navigation in said environment.

Software frameworks for robotic applications are covered to provide an overview of how software is being made for robots with focus on reusability. This aspect is important because it prevents code being rewritten every time robots or new hardware parts are being used.

The approaches mentioned in *Basic navigation tasks* (see section 2.1) will be expanded upon in *Advances navigation tasks* (see section2.2). The section about *Remaining challenges* (see section2.3) will conclude these two and show what kind of potential for improvement these existing methods possess.

2.1 Basic navigation tasks

If a mobile robot is supposed to work autonomously in a given environment, one of its basic abilities is being able to **navigate** in said environment. This navigation task, which comes to humans naturally, is quite the challenge in robotics. To handle its complexity, it is broken down into the tasks of *self-localization*, *path planning*, *map building* and *map interpretation*.

The localization problem is also a reoccurring one, in which the robot needs to figure out its position in the environment. Solutions for this problem have been developed over time, one example being AMCL [Ger]. Together with a map, the localization can be used for path planning, where the robot computes a path to its destination. This path is, ideally, the shortest one from the robot's current position and avoids collisions of the robot with the environment. These collisions can be avoided when the dimensions of the robot are known a map of the environment is provided. A map is an abstract representation of the environment, which can be built by the robot. One problem in the mapping process is that a robot needs to know its current position to generate a map from its sensor data, but on the other hand a map is needed to determine the current position of the robot. So the codependent tasks of mapping and localization have to be handled simultaneously, which can be quite difficult. This problem is called "SLAM", or *Simultaneous Localization and Mapping* and is a common problem for mobile robots. This topic has been intensively studied, and introductory as well as further work about different SLAM techniques can be found in [Thr02, SK08, Gri].

A problem with SLAM is that dynamic objects can be included in the map, but when the map is later used for path planning, the dynamic object can already be gone. This can, without any extensions to the SLAM process, only be registered by revisiting the area. The concept in this thesis provides an approach to deal with this exact problem by getting live data about the environment. The next gives an overview of information sources that can provide such information.

2.1.1 Sources of information

A mobile robot has two sources of information at its disposal, from which it can make assumptions about its own state and the state of the environment. These information sources are can be divided into two classes - idiothetic and allothetic [JY08, WB99, Aug12].

Idiothetic information refers to intrinsic measurements like speed, acceleration, and wheel movement (odometry). Using this type of information in combination with a previously known position, the current position of the robot can be determined. This process is called dead-reckoning[Bor95, Aug12]. One problem of this approach is the accumulation of errors in measurements over time, resulting in an increasingly poor estimate of the robot's position. This accumulation of errors, called *drift*, has to be dealt with in order to get useful position estimates.

On the other hand, there is allothetic information, which refers to information of sensors that observe the environment. Examples for this can be vision-based sensors, laser range sensors, or sonar. One problem in this approach is called *perceptual aliasing* and refers to the problem that similar looking places create similar sensor readings, which can be hard or impossible to distinguish without additional information. This can lead to ambiguous position estimates. A second problem is that one area can change its appearance over time, resulting in the possibility that it cannot be correctly identified later. This problem is called *perceptual variability*.

Most robots use a combination of these two information sources to build maps of their environment, that can be used to localize the robot in said map or for path planning. When combined, these approaches can, to a certain degree, compensate the errors of each other. For example, allothetic information can be used to compensate for the accumulated errors from the use of idiothetic information, and idiothetic information can be used to disambiguate locations that seem to be identical from the use of allothetic information [JY08].

Although these two information sources can produce useful information about the robot and its environment, a limiting factor remains - the physical space on the robot. Sooner or later, a robot will run out of space where sensors can be attached. In case of bigger robots, this might not seems as a big problem, but for smaller robots this becomes more urgent.

One way to handle this situation is to position sensors in the environment and make them communicate their measurements to the robot. This enables the usage of a larger variance, and even bigger sensors in the mapping and localization process of the robot. At the same time, the sensing range can be expanded beyond the range of the visible range of the robot. This expansion of the allothetic information gathering can and has been done in form of sensor networks, which can do even more than simply collect information. According to Batalin et al., "the sensor network serves as the communication, sensing, and computation medium for the robot" [BSH04]. Some advantages that arise from the combination of sensor networks and robots are, according to Batalin et al.:

- “The robot does not have [to have] a pre-decided environment map[...].” [BSH04]
- “The environment is not required to be static.” [BSH04]
- “The robot does not have to perform localization and mapping.” [BSH04]
- “The robot does not have to be sophisticated - the primary computation is performed distributively in the sensor network, the only sensor required is for obstacle avoidance.” [BSH04]

Additionally, a sensor networks removes the risk of a single point of failure by introducing multiple different and/or redundant sensors into the sensing process and would make the introduction of sensors possible that would otherwise have no physical space on the robot, use up too much energy, or could not be mounted in the idea height to yield maximal effectivity.
These networks can be designed in a way that provides the robot with additional sensors only, or in a way that can take computation load off of the robot (in mapping and localization, for example).

This sharing of information across the network is another problem that had to be solved. With a limited amount of information sources, as a first thought it could be reasonable to poll the sources for new information, but with an increasing number of information sources this strategy produces more and more overhead. If the information source is polled, a certain load is produced on the communication medium, which increases with higher numbers of available sources. Additionally, not every information poll has to return new and useful values to the requesting unit, which can use up a lot of the available resources on the medium. Instead, it would be better, if the unit in need of new information would be notified by the source if there was new information available.
A solution for this problem was found in Software Engineering and is known as the “Observer-Pattern” or *Publish-Subscribe*. This pattern, describing the aforementioned behaviour of notifying the unit in need of information has been implemented in various frameworks for robotics, that will also be mentioned in this chapter.
In a system using publish-subscribe, there are two distinct participants - the *Publisher* and *Subscriber*. A publisher is a unit that in some way produces new information by computing something or, in this case, by making observations about the environment. A subscriber is a unit that wants to access the information produced by the publisher and therefore declares this intention

by registering itself as a subscriber at the publisher. The subscriber is then automatically notified by the publisher when new information is present. It is also possible and common that subscribers subscribe to multiple publishers, and that publishers have multiple subscribers. Furthermore, it is possible for a unit to be both a subscriber and a publisher, so that a unit subscribes to a certain information, performs a computation with it and then publishes the result from this computation.

This communication problem is a reoccurring one, that has to be solved every time, a similar situation arises - not only in robotics and not always in the context of sensor networks, but every time information passing between different software modules is needed (e.g. UPSP [TN12] or Mires [SGV+06]). This problem is by far not the only one that had to be solved repeatedly and to prevent the implementation of them over and over again was to collect these solutions in frameworks. An overview about two of these frameworks from the field of robotics, that were used in this scenario, is presented in short in the next section, and in more detail in the implementation chapter (see 4.1).

2.1.2 Software frameworks for robotics

The previous section mentioned that solutions for reoccurring tasks were implemented and collected in frameworks in order to avoid that these solutions get implemented repeatedly. Since different robots differ considerably in their hardware equipment and their scenarios, this collection of solutions has to take this into account. By using an appropriate modularization, it is possible to provide software modules that have no (or, at least, very few) dependencies on hardware or other software modules. Using the publish-subscribe mechanism, it is possible to provide communication between modules without creating dependencies between them. The only requirement is the ability to be able to use publish-subscribe. This way, different modules can be dynamically composed, creating vastly different scenarios using the same underlying solutions. Two examples for this sort of framework are the “Robot Operating System” (ROS) [QCG+09] and the “Middleware for Robotic Applications” (MIRA) [ELS+12], which are covered in more detail in the implementation chapter (see 4.1).

Examples for such solutions are algorithms for common problems in robotic navigation, like localization, path planning, or map building. Since the concept in this thesis revolves around maps and how information can be added to them, the next section will cover the topic of maps in more detail.

2.1.3 Map representations

There exist two distinct classes of maps which have a different focus when it comes to representing the environment. The first kind to be mentioned here, called topological maps, are a sparse representation of the environment describing distinct places and their relations [Aug12]. This approach doesn't generate maps in the geometric sense, but rather graphs whose nodes represent distinct places in the environment and the arcs represent the relations and adjacencies between those places. A similar concept can be found in subway maps that focus on displaying adjacency information, rather than correct distance scales. Figure 2.1 shows an example of a topological map. This environment representations aims to construct minimal topological representations, rather than a precise image of the environment. Due to their allothetic character, these maps suffer from perceptual aliasing, creating possible ambiguities. This makes the application of topological maps less useful in indoor scenarios compared to metric maps. On the other hand, topological maps are less memory expensive than metric maps due to their highly abstract nature.

There exist, of course, hybrid approaches that aim to combine the strengths of both approaches. An advantage can be that measurement drift can be reset when a certain node is reached, as mentioned in [MF03]. Or, as explained in [Thr00], the navigation task can be separated into two steps - *high level* and *precision*. For the high level navigation, topological maps are used and for more precise navigation, the metric maps are consulted.
This combination of approaches could make the topological maps a viable option for indoor scenarios again.

The second kind, called metric map, represents the environment and its associated objects in a way that is similar to room layouts - precise and scale-proportional using data collected through metrical measurements [Aug12].

This representations makes it relatively easy to localize a robot by recognition of geometric shapes, but these maps can grow very large, depending on the chosen discretization of the environment. Growing maps are then becoming more and more expensive to calculate. Due to this fact, metric maps are

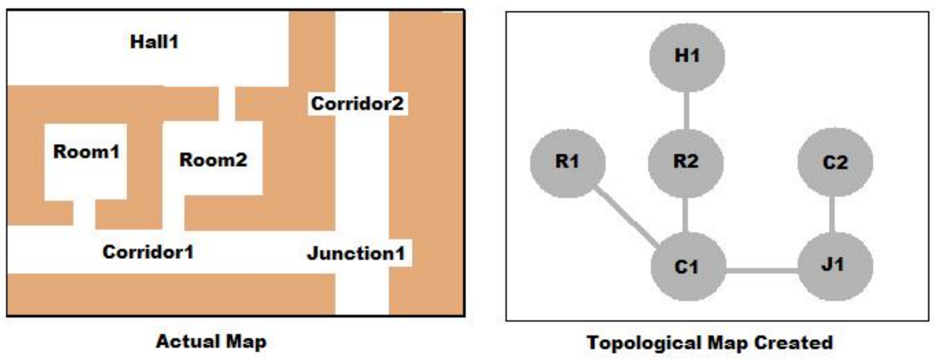

Figure 2.1: Example for a topological map [Rob]

particularly useful in indoor scenarios, but less useful in outdoor scenarios with vast environments.

Because the approach presented in this thesis is focused on indoor scenarios and aims to expand on different pre-existing approaches for these scenarios, the topological maps will not be used any further in this document. Instead, the sole focus will be on metric maps. Widely used metric maps for indoor scenarios are the so called *Grid Maps*, which now shall be explained in a little more detail.

2.1.4 Grid Maps

In the grid mapping approach, the environment is represented by a “two or three dimensional regular grid” [MM96]. Each cell in this grid stores a value for the evidence about a certain feature being present in the observed area. Depending on which kind of feature is being observed, the name for this approach varies. In its most generic form it is called “Evidence Grid”, where it just stores the evidence for some feature being present. This can go as far as each cell containing an evidence vector for different observations for each cell. In the form it is widely used in robotic mapping, navigation and localization it is referred to as “Occupancy Grid”, because it stores the probability that the corresponding space in the environment is occupied by something that the robot can not navigate through. Figure 2.2 shows an example of an occupancy grid.

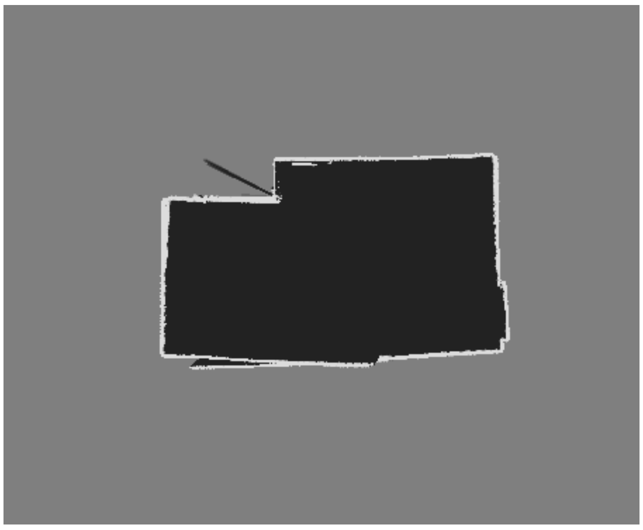

Figure 2.2: Example for an occupancy grid

In an early version, this Occupancy Grid contained two values per cell. The first one was the probability that the cell is occupied and the second one that the cell is empty. Initially, when nothing was known about the environment, both values were set to zero. Contradictory information from different sensors about the same cell would lead to both values being close to one.
At this time mostly sonar sensors were used in the mapping process because of their relatively low price. The way sonar sensors map their readings onto “occupied” and “empty” lead to an asymmetrical handling of the two grid values. If the sensor doesn’t register anything in its cone, the whole observed area is marked as empty, but in order to mark the observed space as occupied, there only needs to be a small object somewhere in the area and it has to be just big enough to be detected by the sensor. This different handling of occupation information lead to “occupied” values being computed as $1 - empty$ and normalized before being added to the grid. [MM96]
A later analysis [MM96] revealed that it is sufficient when only the probability that the cell is occupied is stored and that no significant performance difference could be noticed in experiments. The only difference to the first version, beyond using only one value, is that the values are not normalized any more.
Probabilities from different sensor system have to be integrated into one map when a robot uses more than one type of sensor. This can be done using an estimation method called *independent opinion pool*[AG92, Ber13] in which the evidence of the grid cells is summed and normalized appropriately.[MM96]

Since its introduction in 1989, the Occupancy Grid has been used in various projects different approaches, some examples will be shown in the next section.

2.2 Advanced navigation tasks

The previously mentioned mapping approaches assume a static environment in order to work correctly, but in reality static environments can only be guaranteed in rare cases. Most of the time robots have to deal with some degree of dynamism in their environment. That could be, for example, people crossing their paths in hallways, other robots that fulfil tasks in the same environment, doors that are opened and closed, or simply chairs that change their position every time someone leaves or uses them.

One naive approach to handle dynamic situations could be to simply ignore them and keep on using the same methods as before and still assuming the environment to be static. Unfortunately, simply ignoring these aspects would not only not solve the underlying problem, but also generate a new one, namely *deteriorated navigation*. This can mean that if the robot works in an environment and needs to move between different areas, it notices a blocked path exactly in the moment when it can see the obstruction of its path. Due to the lack of information in the path planning phase, the robot chooses a path that seems to be a good, when in reality a better path could have been chosen when the necessary information had been present earlier. Without this information, the robot can only plan an alternative route to its destination when it is right in front of the obstacle and can detect it with its own sensors.

If the underlying map for navigation is not updated according to the changing environment, then the robot will suffer from the aforementioned problem of perceptual variability, which can become worse up to the point where the robot is not able to successfully localize itself in the environment anymore.
The problem arises when either dynamic aspects manifest after a map of the environment has been created, or when dynamic aspects influence the mapping process. In the first case, the robot notices at some point in time that there is an object in its path that is, according to the map, not supposed to be there. If this happens more often the robot could lose its ability to localize itself in the environment.
In the second case a moving object could cross the path of the robot while

it is mapping the environment, therefore including the objects into the map, although a few moments later it is already disappeared again. Later on, when the robot uses its map for navigation, this place can not be identified clearly, since the mistakenly included object cannot be found anymore [MT08].

In order to avoid this loss of its localization ability, there have to be more sophisticated methods that keep the underlying map up to date to handle the dynamic aspects of the environment. Some of them will be mentioned here to give a rough overview about this problem can be dealt with and to show which parts of these approaches could be taken and combined for the concept of this thesis. These approaches all revolve around the usage of occupancy grids, which, according to Mitsou and Tzafestas [MT08] suffer from the problem, that each cell stores only the current state of the environment and has no means of "remembering" older states. Thus, when a change in the environment is noticed, the corresponding cell will simply be overwritten with a new value, "forgetting" the old one. Mitsou and Tzafestas [MT08] have identified three categories that proposed solutions for this problem fall into:

1. *Occupancy grids with different time scales*, which uses a number of occupancy grids where each grid is updated at a different rate.
2. *Temporal occupancy grid* mapping, which extends the occupancy grid to efficiently preserve the history of the evolution across the time axis.
3. The *static-dynamic grids* mapping approach which uses two grids to differentiate the static from the dynamic areas of the map.

In addition to the presentation of these three approaches, another approach will be mentioned afterwards that proposes a method to learn different spatial configurations of the environment, which provided some inspiration for this thesis.

2.2.1 Occupancy grids with different time scales

In this approach, more than one occupancy grid is used (see [AHM02, BD$^+$05]), where every grid has its own particular time scale that indicates the refresh rate of the map (e.g. every 20 seconds) [MT08]. By examining the values of cells across different times, the character of the cell can be determined. A cell is static, if it has only one state (either occupied or free) or in all time scales.

On the other hand, it can be concluded that the cell was once occupied by a moving object if its state changed in some occupancy grids, but not in others. An approach using this method is called *temporal occupancy grid* (TOG) [AHM02]. It extends the common occupancy grid by a time dimension which is used to classify the cells based on their occupancy values across different points in time [MT08]. The TOG can be modelled as a matrix with two spatial dimensions, one time dimension and additional dimensions for the different time scales (one dimension per time scale). While the original occupancy grid stores one probability value for each cell, this approach stores multiple values for each cell. Each of these values represents the state of the cell at a certain point in time. Using these values, the nature of the cell can be identified, for which three possibilities exist - static object, free area, and moving object. A cell is classified as a static object, when it is occupied in every occupancy grid. If the cell is not occupied in any of the occupancy grids, it is identified as free area instead. Finally, when it was only occupied in some occupancy grids, it was occupied by a moving object.
While this sounds like a good way to deal with dynamic aspects, the problem of this approach "[...]lies in the fact that the optimal number of different timescales, an important parameter of the algorithm, cannot be computed in a formal way but must be intuitively selected by the user" [MT08].

2.2.2 Extended temporal occupancy grid

The extended temporal occupancy grid (eTOG) improves on the original TOG by making a more efficient use of occupancy values, which stores more occupancy values than actually needed [MT08]. The first change to its name-giving predecessor is that the eTOG approach only uses one grid. Instead of storing occupancy values, the grid cells contain an index structure (time index - see [EWK90]) that keeps track of the occupancy probability of the cell. Using this index, a complete history of the occupancy probability can be stored. For the whole map this means that it is possible to store the complete history of changes of the environment [MT08]. The classification is done as in the TOG approach by reviewing all stored values for a cell. The identified nature of the cells can, again, fall into the categories *static object*, *free area*, and *dynamic object*.
The problem of this approach, however, is the increasing use of memory. In static environments, the memory usage is about equal to the traditional oc-

cupancy grid, but in dynamic environments the memory usage increases with the size of dynamic effects, where it can become “extremely large” [MT08].

2.2.3 Static-dynamic grids

Most approaches handle dynamic aspect differently than the two approaches mentioned before by using at least two occupancy grids (see [TK06, WTT03, WS04]). While the first grid is used to store the static aspects of the environment, the second grid is used to store dynamic aspects. The cell values in the first grid represent the probability of a static object being present, as it is in the traditional occupancy grid, and the values in the second grid indicate the probability of the presence of a moving object [MT08]. The classification of static and dynamic objects in the approach proposed by Wolf & Sukhatme [WS04] works as follows. Static parts of the environment never change their position in each of the grids and thus serve as a reference to distinguish between sensor readings from static and dynamic objects. Wolf & Sukhatme [WS04] introduced new models to compute the probabilities in the cells across the two maps and to distinguish between static and dynamic cells.
Other than in the previously TOG and eTOG approaches, in this method no historical information about the dynamic aspects is stored, which means that “only the current position of the dynamic objects can be available” [MT08].

2.2.4 Learning possible spatial configurations

The approach presented in [Sta09] is targeted to specifically to office environments. While the aforementioned approaches filter out dynamic objects, here only the objects that are currently in motion are discarded. Instead, low-dynamic objects or quasi-static states are explicitly modelled with a limited number of configurations for any given dynamic object. The paper mentions two examples for low-dynamic objects, the first one being doors that, most of the time, are either completely closed or completely open, and the other one being parking spaces that can be occupied or not. Both examples have dynamism in them, but only over a larger time frame than, for example, people walking in corridors.

The aforementioned approaches could filter out all dynamic objects, including parked cars or doors, which in turn would lead to maps misrepresenting

the current state of the environment. This, again, leads to deteriorated localization for robots. By modelling different configurations of dynamic objects localization can be improved, if high-dynamic objects are ignored. Including them into maps could lead equally quickly to deteriorated localization, because these objects have a high probability of being somewhere else when the robot returns to the position where it first spotted them. So, this approach works best for in low-dynamic environments.
The idea behind this method is to map the environment at a certain point of time and to learn configurations of low-dynamic objects by revisiting these places at different points in time. The map is divided into several sub-maps and for each one of them the configurations of the existing objects are learned. Figure 2.3 shows an example for different configurations of doors.

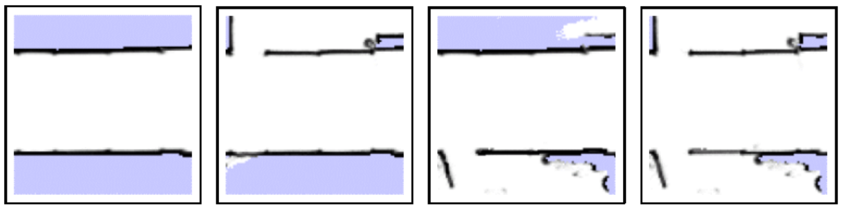

Figure 2.3: Different configurations of doors in a corridor [Sta09]

This map segmentation can work in different ways. One way could be to store the whole map for each possible configuration. This naive approach has the problem that the number of stored maps is exponential to the number of dynamic objects, which leads to an immensely fast growing need for memory to store these maps. Another, more usable approach used in the paper is to segment the environment into local areas and store these segments as sub-maps. These sub-maps can vary in size, the extremes being one sub-map having the size of the overall environment, or having the size of a single grid cell. Both extremes are not practical for use for different reasons. Having the size of the overall environment leads to the aforementioned problem of exponential growth in the number of maps and having the size of a single grid cell works against the assumption of Occupancy Grids that cells change their state independently from others, which is not justified in a scenario with dynamic objects bigger than a single grid cell. The paper mentions an upper bound of $20m^2$ for the size of sub-maps, as a balance between those extremes.

After an initial recording of the environment to generate a map, the robot has to observe the environment again at different points of time. When it detects contradictory observations it is safe to assume that these are typically caused by dynamic objects. Then, this area has to be observed to determine the whole region which is affected by the dynamic aspect. Several of these regions are then merged together to form a sub-map. This is used as a way to handle complexity. If single dynamic objects were used as a sub-map, then a lot of small maps would have to be stored, and if too many objects were in one sub-map, a lot of bigger maps would have to be stored. Sub-maps of a maximum size of $20m^2$ are used as a middle ground. An example for a resulting patch map is shown in figure 2.4.

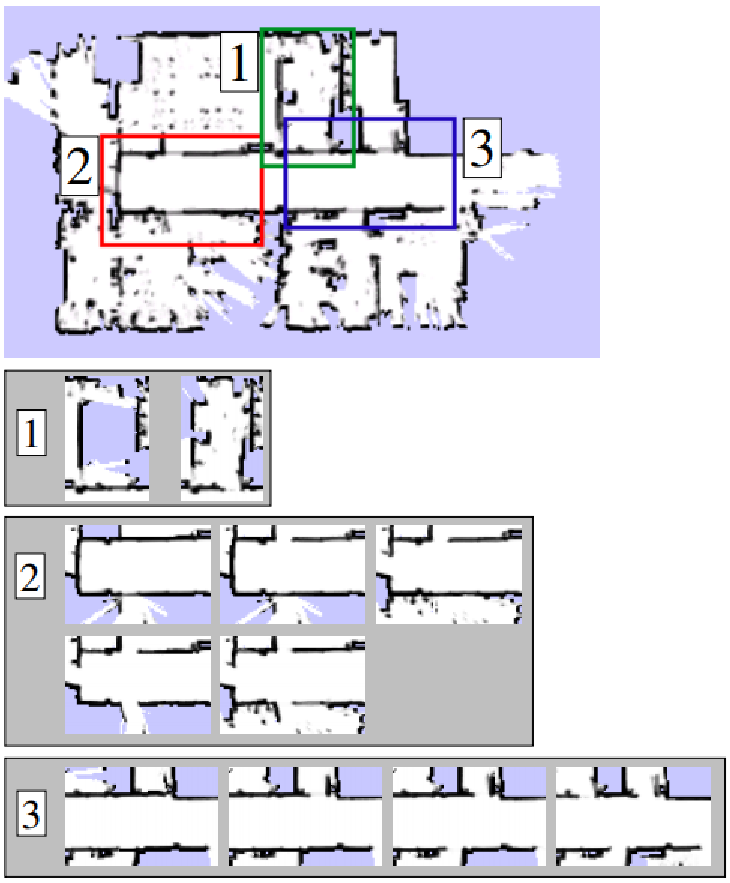

Figure 2.4: "The patch-map represents the different configurations learned for the individual sub-maps in a typical office environment." [Sta09]

The problem that remains with this approach is that these quasi-static states don't always suffice to represent the environment good enough. For example, a door between the states of "completely closed" and "completely open" is currently not mentioned by this paper. These in-between states, however, can be very important depending on the abilities of the robot. If it is able to push open a half-open door, then this door can be viewed generally as "open" whereas it should be regarded as closed if the robot is not able to push doors. One way to deal with this could be to add a few explicitly modeled in-between states to this approach. This, however, would again increase the needed memory to store all these learned configurations. Another way to approach this will be mentioned in the concept chapter of this thesis (see chapter 3).

2.3 Remaining challenges

The previously mentioned approaches still have their peculiarities and drawbacks. Managing multiple maps with different time scales needs either experimentation or experience, and on top of that additional adaptations are required for each new environment or when the dynamics of a certain environment change fundamentally. For example, if an addition is made to a building and a former blind end becomes a passage to that new part of the building, the time scales have to be adapted in order to capture the new aspects of dynamism introduced by this change (e.g. greater amounts of people, when before only robots were present).

In the case of the eTOG approach, the memory usage becomes even greater than in the traditional occupancy grid approach, making it difficult or even impossible to use it on smaller robots which have limited computing power or memory. Paired with a sensor network that, as mentioned before, can take computational load off of the robot, the same principle applies here - the sensor nodes in the network have to have a certain computational power to compute the grids or dynamic aspects of the environment. Thus, making the overall cost to implement such a network greater, which can severely limit the size of the network when single nodes become too expensive to use in greater amounts.

The problem of the last approach, where configurations of the environment are learned and then represented as patch maps is that this approach takes time to react to new found configurations and can only learn them while actively observing them. Another problem here is to find a certain threshold to determine when a dynamic object is in a new configuration. For example, for doors there can exists a huge number of different configurations depending on the appointed granularity. By this, it is meant that if the door opens for another degree and this change can be measured, the question arises if this state is a new configuration of the door or if it falls in an interval of an already known configuration. Determining the distinction between these configurations is again something that needs experimentation or pre-existing experience and changes from object to object. In addition to this, each configuration has to be stored in a separate patch map, which further increases the memory usage. Although a solution for this problem has been proposed by Stachniss [Sta09] by limiting the size of these patch maps, they still have to be stored and with increasing number of configurations the memory usage increases as well.

One limitation all the mentioned approaches have in common is that they only incorporate observations made by the sensors on the robot and thereby limit themselves to its field of view. As mentioned before, this means that robots may take a suboptimal path to their destination, because of a lack of information it looked better than it is in reality. While the mentioned approaches can handle dynamic aspects in the SLAM phase or prevent deteriorated localization by learning different configurations, they do not provide a method to update the robot's map with information about dynamic aspects. This information could be useful for path planning, so unnecessary replanning can be avoided by knowing more about the changes happening in the environment.

The goal of this thesis is to avoid these problems and propose a different approach to the problem of building maps of dynamic environments. Using a combination of ideas of existing approaches can, so the assumption, yield better results than these approaches that explicitly try to handle dynamism. By combining the occupancy grid approach for indoor environments with a sensor network that increases the field of view from a robot-based one to one that can cover whole areas it is possible to observe all dynamic aspects when they happen. Using these live observations, it is now possible to inform the robot about blocked passages and moving objects. These observations then have to be translated into map information and integrated into the map of the environment.

In conclusion, this chapter gave a short overview of common methods used in robotic applications, frameworks used in robotic scenarios, and how sensor networks can be used in combination with robots. This provided the foundation for further research in this thesis, which will include the combination of some of the mentioned approaches. Chapter 3 will use the information from this chapter to show how these approaches can be built upon to create a new method to create maps that cover information about the whole area covered by the robot and a sensor network.

3 Concept

The goal of the approach presented in this thesis is, in short, to enable robots to navigate in environments without having any former knowledge about it. Traditionally, robots use their on-board sensors to create a map of the environment, as mentioned in the previous chapter. On top of that, it is also possible to let a sensor network collect additional information about the environment with a variety of different sensors and to make this information available to the robot. This approach explains how this information from the sensor network can be used to extend the robot's map and update it with live data from the environment. The map can be extended beyond the sensor range of the robot using this information.

By having information about areas that the robot is not able to "see" by itself, it is possible to avoid temporarily blocked passages more effectively. In scenarios without external information, the robot would plan a path using its current observation and its old data about the environment, therefore making it possible to drive down a passage that is temporarily blocked. This would, of course be noticed by the robot and it could act upon it accordingly, but by utilizing information from a sensor network, the robot would be able to plan around it before taking the blocked path first.

Because of the heterogeneous nature of the sensor network, there is also the ability to filter this information in order to get only the information that is currently relevant to the robot. So, for example, it is possible to limit the incoming information to sensors in areas that are adjacent to the current path of the robot.

3.1 Overview

The approach presented in this chapter draws inspiration from the patch map approach in a way that the one global static map is recorded at the beginning

of the scenario and that parts of the map can be updated without recreating a map for the whole environment. These little maps are built by using sensor measurements provided by a sensor network. Each sensor provides enough information in addition to the measurement that these maps can be merged into the global map to create one occupancy grid that contains all the information. This map can then be used in the same way as traditional occupancy grids are used, with the exception that the application using this system needs to be able to merge new information into the map at runtime.
This approach aims to be simple enough, that only an initial setup phase is needed, in which a sensor network is installed in the environment. After that, different dynamic aspects can be detected without learning new configurations or distinguishing between different degrees of dynamism. Depending on how much of the environment is covered by the sensor network, it might not be necessary to provide a global static map at the beginning of the scenario. If the sensor network covers enough of the environment to build a global map on its own, this map does not have to be provided individually. In addition to this, there is no learning phase to identify certain states of the environment, or a difficult calibration phase where classifications between different kinds of dynamism have to be determined. At the end, there will be only one map that is constantly updated with exactly the information that the application requested.

The next sections of this chapter cover the basic assumptions that are made in the context of this thesis. This includes how the information for the patch maps is gathered and what kind of sensors can be used in which way to get information about the environment. In addition to that, it mentions how the application (e.g. robot) can choose which kind of information is delivered to it, and how this information is transformed into occupancy grids.

3.2 Assumptions in this thesis

Robotic scenarios that take place outside of robot labs have to deal with a great amount environmental factors that can influence every part of the scenario. One example for this can be that sensors are influenced by certain factors. Depending on the sensor, these influences can be visual (e.g. sunlight), auditory (e.g. motors emitting sound), or just structural conditions that lead to degradation of the sensor data like tiles on the ground that lead

to vibrations of the robot and its sensors.
To cover these influences would be enough material for a whole thesis on its own. So in order to keep this thesis within the scope of robotic mapping, a few assumptions had to be made.

The first of these assumptions made is that a relatively stable environment is provided. That means in this case that no environmental changes happen that would influence the measurements made by any of the sensors. So, for example, conditions like humidity and air pressure stay the same, which would otherwise disturb readings of ultrasound sensors. Also, sunlight is not assumed to be an influence when infrared sensors are used.

The second assumption is that sensors work without erroneous values, like glitches that produce sudden outliers. Anything that would need error correction is assumed to be absent, so the focus of this thesis will not shift in this direction.

A third assumption is that the underlying communication network delivers every message that is produced in any scenario, so that at all times the necessary information is available to be included into the map.

3.3 Information gathering

As mentioned at the beginning of the chapter, the environment has to be prepared by installing a sensor network, so that changes all over the environment can be measured, published and transformed into suited occupancy grids that can be included in the global static map. The application (information sink) and the sensors (information source) are using a special mechanism to dynamically create communication channels. This differs from the traditional approach where the communication is established statically by having information sources and sinks communicating over a predefined channel. The system providing this mechanism will be covered in more detail in the next chapter.
This section will cover the process of gathering information that can later be transformed into map data that is useful for the robot. A selection of useful sensors is provided and it is explained how these can be used to observe different aspects of the environment.

3.3.1 Selected sensors

After this overview of possible use cases, which already showed how some sensors can be used to get information about the environment, this section will cover how their measurements can be used to create data for maps. It provides an overview of sensors that can provide useful information about the environment. These sensors are explained in a short manner, before the next section covers in a little more detail for what kind of observation they can be used.

Laser range scanners (LIDAR) currently are the default sensors for mapping environments on many indoor robots, like the SCITOS G5 used in this scenario. They emit single laser beams that are reflected on objects. When the reflection is registered at the sensor, the distance to the object can be determined using the time the laser beam needed to travel to the object and back again.

Infrared distance sensors can be used in the same way as laser scanners.

Sonar distance sensors were used to build whole occupancy grid maps before laser scanners were affordable, so their range scans can be directly used to build maps [Mor88]. One thing to be on the lookout for is that these sensors don't emit single straight beams to measure distances, but rather cones. So a small object somewhere in this cone affects the whole area that is covered by this cone. This can be dealt with by using multiple sonar sensors with overlapping cones, which increases the accuracy when detecting objects.

Reed contacts can be used to detect open or closed doors in order to determine if a robot can use certain passages or not. The drawback of this sensor is that the state of the door can only be determined when the contact is closed. An open contact on the other hand carries no detailed information, so the exact state of the door is unknown.

Cameras can be used for all kinds of detections, when the two-dimensional shape of the objects are known. They can be used to detect and differentiate between different robots and people, and can be used to automatically register certain identifiers like bar codes and serial numbers when the line of sight is not obstructed. The goal of using cameras is to get the class of the object and its pose in global coordinates.

RGBD cameras like the Microsoft Kinect [Mic], the Xtion PRO Live [Asu], and the Intel RealSense [Int] can be used to get a three-dimensional representation of the environment. Using octrees the space can be partitioned to get a three-dimensional grid representation. This can also be used to get a two-dimensional map of the environment by setting points to "occupied" where the z-value exceeds a certain threshold that poses an obstacle for the robot. Using these types of cameras yields the additional benefit of being able to perform an object detection and distinguish between tables, chairs, shelves, different kinds of robots, and people. The goal of using these sensors is, similar to cameras, to get the class and pose of an object, or a three-dimensional mesh or a point cloud of an unknown object. Figure 3.1 shows an example for a point cloud.

Inertial navigation systems (INS) can be used to get idiothetic information (see 2.1.1) about the robot and to calculate its position using dead-reckoning.

Odometry can be used to get idiothetic information as well. By interpreting the values from rotary encoders or optical flow sensors, the travelled distance and rotations can be derived and using dead-reckoning the position of the robot can be computed.

RFID tags and wireless beacons attached to objects can be used to identify said objects. The information stored in the tags can be used to insert these objects into the map. On one hand this can simply mean that boxes from delivery services can be identified and placed into the map when it is known where the boxes were scanned. But even including more uncertain information is possible. For example, when doors can only be opened using RFID tags, the entity opening the door can be identified. Now rules can be derived from that information, like driving slower in a certain area where other robots or people have been registered, or completely avoiding areas where the floor is currently cleaned. As an example for wireless beacons, the iBeacon can be mentioned [Inc].

3.3.2 Possible use cases

This section provides some ideas of how different measurable factors of the environment can be measured by the sensors mentioned in the section before.

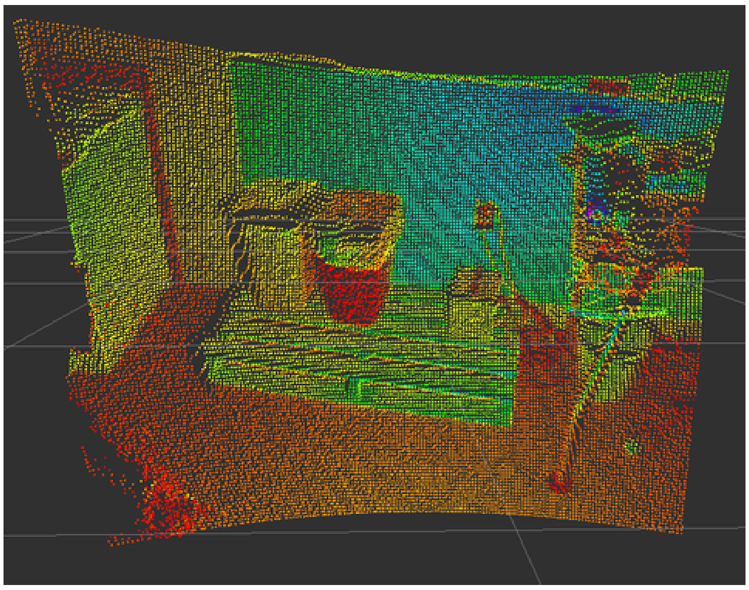

Figure 3.1: Example for a point cloud [Foo]

Some of these observations about the environment will go beyond the basic information of "occupied", "free", and "unknown" but rather serve as extensions to the occupancy grid information, or can be used for something completely different, like eliminating odometry drift.

Figure 3.2 shows a tree-like visualization of possible sources that provide position information about an object. This position information is divided into static and dynamic objects. In an office scenario, static objects include walls, tables and chairs, although chairs could also potentially be seen as dynamic objects. For these objects there exist different sources that can provide position information. The first and obvious source is an existing static map of the environment, which directly provides this information. Another source can be an RGDB sensor, which can generate a three-dimensional mesh of observed objects, or even run an object detection to identify the object. Either way, the sensor provides information about the location of the object when it was registered. Together with the shape information, the detected object can be added to the robot's map. As a last example the charging station of a robot shall be mentioned, which is probably installed somewhere at a fixed location. When the robot is able to detect the charging station, the position of it can be used to reset the robot's odometry and thereby eliminating any accumulated odometry drift.

For dynamic object, there also exist a plethora of possible information sources. One is, again, the RGBD sensor that can detect robots when the robot models are known and determine their position in the environment. Thus, they provide a useful source of information to correct the localization of the robots. This can go so far that each time such a sensor detects a robot, the localization can be reset and therefore eliminating accumulated drift errors.

Even a robot itself can be used as an information source. It is possible that a robot produces a heartbeat, which includes its position, which can be used to include the robot into the map when its dimensions are known. This information can be used by other robots to avoid collisions, but needs to be ignored by the robot providing this information, or else it will regard itself as an obstacle that always blocks the path.

For doors, not only the position, but also the information about the opening angle is important, so a robot is able to detect if a door blocks its path or not. Here, again, RGBD sensors can be used to get the needed information. By simply getting the point cloud and detecting the model of the door, the information can be derived. But again, there might be simpler and cheaper alternatives to this approach.

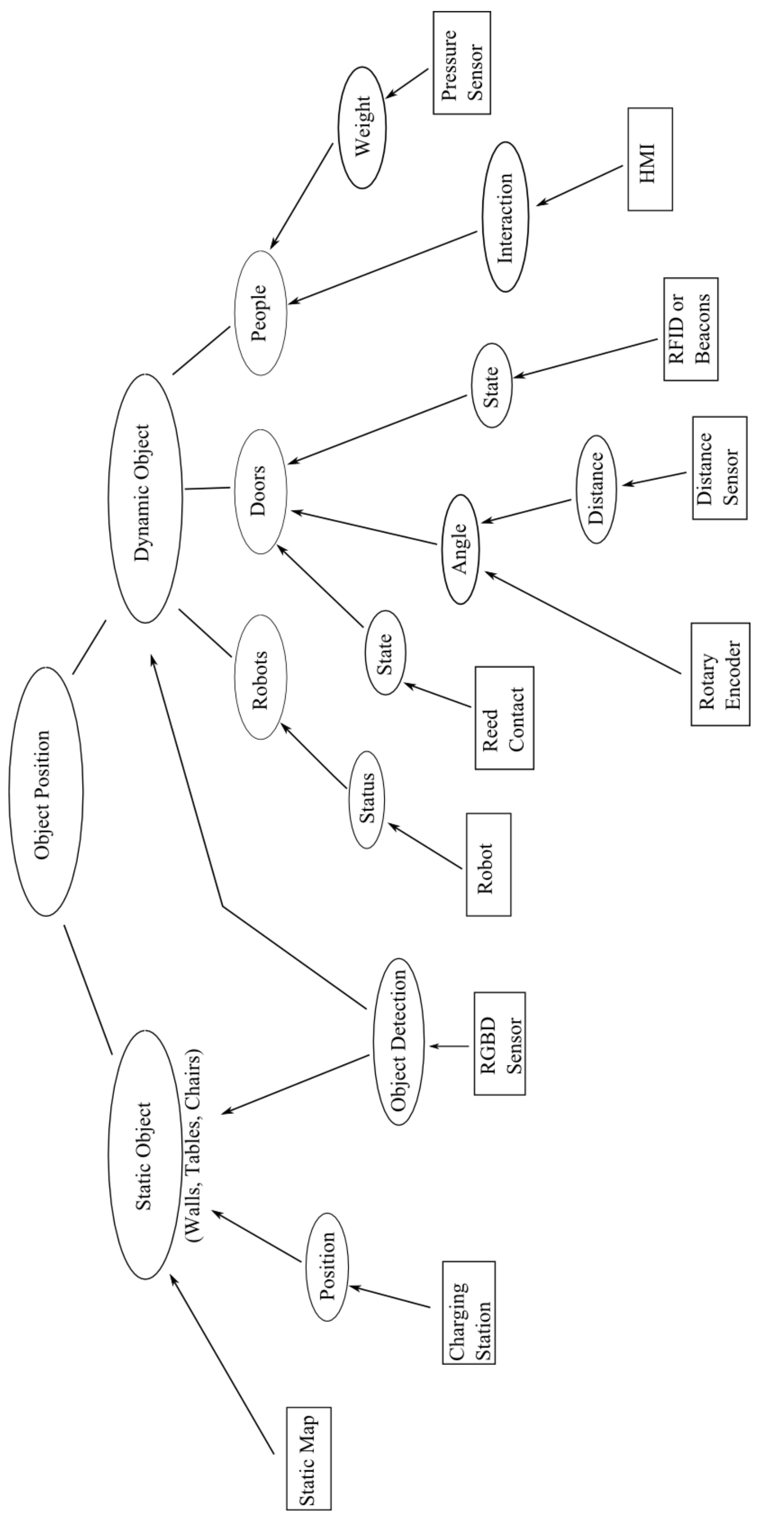

Figure 3.2: Possible information sources and transformations that provide object positions

One not so obvious possibility is the utilization of RFID sensors or wireless beacons on doors, which can provide the information about the state of the door. Either the door is closed, or it is being opened and it is known to remain in the open state for a certain time. This information can be provided to the participating robots, showing them a passage, that's either open or closed. Going one step further, it can be useful to know which door is RFID-activated when a robot can be equipped with a beacon, which could enable the robot to open these doors by itself.
Another example could be the automatic detection of objects via RFID tags. If a tag carries information about the associated object and informations about the object shape are known, this object can be added to the map automatically.

Reed contacts can provide similar information, although not as precise. The contact can either be triggered when the door is closed or open, providing exact information about the state of the door. While the RFID option still had some inaccuracy when the door is opening or closing, this sensor provides information only when one of the two states is reached. On the other hand, there is quite some uncertainty about the state of the door when the contact is not closed. In this case the door can be anywhere else, which can be still useful for robots that are able to push open a door that is on the latch.

Rotary encoders on door hinges can be used to get exact information about the opening angle of doors. This can be used as a more precise alternative to the option with reed contacts, where only one state per contact could be determined accurately. Using a rotary encoder, the opening angle of a door can be determined up to the point where it is possible to decide if a robot can fit through a half-opened door or if it has to use an alternative route.

A seconds option, that is not as obvious is the utilization of range sensors. These can be used to determine the opening angle of a door as well. By placing such a sensor in parallel with the wall of the door, the distance from the sensor to the door can be measured. A completely closed door will not be registered by the sensor, whereas the distance to the door decreases, the more the door is opened. When the position of the door, its length and the position sensor are known, the opening angle of the door can be computed.

As the last examples for dynamic "objects", the detection of people shall be mentioned. While this could be done, again, with RGBD sensors, two alternative methods shall be presented. If the robot is equipped with a Human-Machine-Interface (HMI), each registered interaction with the (for example) capacitive touch display can be interpreted as the presence of a person. Since the robot model includes the position of the HMI, the position of the person can be easily determined.

Detecting people can also be done by utilizing pressure sensors, that are installed in the floor of a room. With a high enough resolution, the position of individual people can be determined.

Figure 3.3 shows how some sensors can deliver information that can be used to include more than occupancy information into a map. For example, a speed map can be used to limit the robot's speed when it enters an area that includes speed limits.

These speed limits can be obtained in different ways. One way is to use the information from an HMI again. When an interaction is registered, it is reasonable to assume the presence of a person. Consequently, the robot should reduce its movement speed in this area in order to avoid hurting the person.

Another way to obtain informations about a reasonable speed limit is to use humidity sensors. These sensors can be used to detect increased humidity in a corridor after a floor was freshly cleaned, and can publish this information, which can be used to make the robot drive slower in these areas.

As a last example of meta information, NoGo maps shall be mentioned. The idea here is, that areas are marked as blocked, as long as too many people or objects are present. In such cases, robots could have difficulties navigating in the area, so they are forbidden to enter them until there is enough space again. To decide if an area is overcrowded, the people entering and leaving are counted. If the number of people exceeds a certain threshold, the area is blocked for the robot. To count the people entering and leaving, different sensors can be used. These counters can be implemented by arrays of range sensors or RGBD sensors, which can identify if an object moves out of a room or into it. Thereby it is possible to count the number of people that are in a room. This could also be implemented with different tiers, where only the highest tier blocks the room for the robot and lower tiers lower the robot's maximum speed.

Another possibility could be to use periodic events. These can be used to automatically provide information about blocked pathes when a reoccurring event makes it impossible to use certain passages. For example, if it is known, that every two hours a lecture ends, it can be deduced that it can be nearly impossible for the robot to use the hallway leading to the lecture hall for the next 5 to 10 minutes. Using this observation, timed events could be triggered automatically. These events carry occupancy information about the hallway, thus blocking it for the specified time frame and making the robot use alternative routes to its destination.

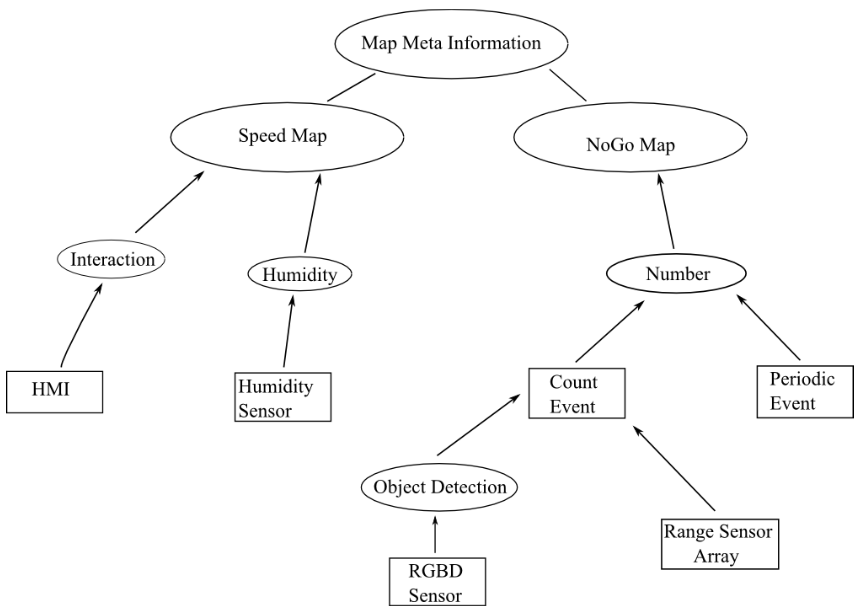

Figure 3.3: Possible information sources and transformations that provide map meta information

In conclusion, it can be said that various sensors can be utilized to get valuable information about the environment. Even more information can be gained when the sensor measurement can be brought into context with the given conditions of the environment. The mentioned use cases are not meant to be a complete or exhaustive. Nonetheless, they show how very different information coming from a network of heterogeneous sensors can provide useful information for mobile robots. The next section will describe how this wealth of information can be handled in a robotic scenario.

3.4 Sensor events and information filtering

The previous section covered suited sensors for information gathering and how they can be used to measure different aspects of the environment. This section describes how this information is structured internally, in order to make it available to the applications. When such a structure is provided, the application also has the possibility to choose what kind of information it needs at the moment. Therefore, it is possible to define filters for the provided information, so that only the currently needed bits are delivered to the application.

3.4.1 Information representation

When a sensor completed a measurement it creates a sensor event that carries, amongst others, the measured information. These events are then provided via a publish-subscribe mechanism.

Each event contains some basic information, regardless of what kind of sensor generated the event. This includes:

- the data type of the contained value,
- how this value is scaled,
- a time stamp of the measurement,
- the ID of the producing sensor,
- the position and rotation of the sensor in the global coordinate system $(x, y, z, \alpha, \beta, \gamma)$,
- the unit of the measured value,
- and the value itself

This can be extended if the situation demands it, so for example an event can also include the update frequency of the sensor, the time difference between the start of the measurement and the completion of building the event, the position of a detected object, noise characteristics and standard deviation of the sensor measurement, or the dimensions of the observed area. The combination of these bits of information defines the type of the event (detection, range, angle, etc.), which can be one criterion for applying filters later on.

The measurement information varies from sensor to sensor and can consist of either a single value like a distance, or multiple values like a vector of distances, a vector of three-dimensional points, or the ID of an object that was detected by the sensor. The next chapter will show in detail how these events are built and what they look like when they are generated.

3.4.2 Information filtering

Subscribing to every available sensor information can create huge traffic loads, depending on the size of the sensor network. This means that the use of multiple robots can quickly exhaust the capabilities of the communication medium. In order to avoid this, a robot can specify what kind of information it needs to complete its current task. As a simple example, let's assume that a robot has to bring an object from its current position to another room in the building. Using its old map data, the shortest path can be planned already. Now, knowing the path, the robot can use the information gathered by the sensor network to get current data of the environment. Because only a certain path is of interest for the robot, a filter can be added, so that only data up to a certain distance to the path is delivered. That could mean, that only data gets to the robot that covers areas that are at most 3 meters alongside the planned path. This should be enough to cover each hallway that the robot has to drive through and the doors of adjacent offices.

On top of that, a time interval could be used of which the upper bound is located in the future. That way, predictions of moving objects could be taken into consideration for the navigation. If a moving object (or even a group of people) is predicted to be crossing the path of the robot in the specified time interval, it might be more sensible to take an alternative route and avoid dealing with an unpredictable situation.
Adding time constraints to subscribed information could also prove to be useful when such information is not available for a certain path. This might be because of a sensor failure or communication problems. If no current information is available for a path, the robot could take that into consideration and decide to use an alternative path, for which current data is available. This path could be longer than the original one, but is safer to traverse because of available sensor information.

3.5 Transforming sensor readings to grid maps

Now that there is a collection of different sensors that provide useful information, there needs to be a way to include this information into the map. This is not always as simple as saying that there exists an object at a certain distance, so the area at that distance can be marked as occupied. When, for example, a humidity sensor is used, the provided information cannot be directly included into the map. Because this type of sensor information, including detection of objects and people, can provide additional information about the environment that cannot directly represented in an occupancy grid, there has to be a way to include them into the map. Here, this is done using "transformations", that take a sensor measurement as input data which they can translate into occupancy grids (or a similar representation that can expand these grids) using certain rules. There exist different ways to define how such a transformation can look like, depending on how flexible this approach needs to be. The following sections will show some examples for transformations.

3.5.1 Transformations without additional knowledge

Here, it is assumed that only the measurement and no additional information is available. In this case, in order to get useful information, additional knowledge has to be derived from the map itself.

For example, when only a distance measurement is provided, this information alone is not very useful, because in the worst case it only means that one cell is occupied. But if the sensor is close to a door and oriented towards it, it might be sensible to assume that it observes the state of the door. So instead of marking a very narrow area as occupied, the information can be transformed into the angle of the door, which then can be drawn into the map as mentioned before. In combination with a reed contact (that may be known to be attached to a door), this could improve the confidence in such a presumption that the range scanner is observing a door.

As an example for a measurement that does not need to derive information from the map in order to be useful, point cloud data shall be mentioned. Everything in the point cloud above a certain height can be interpreted as an obstacle, so only the dimensions of the object have to be determined and it can be placed into the map using the information about the position of the sensor and where the object was detected. The next section will, amongst other things, show how these sensors can provide more useful information when they are used for object detection.

3.5.2 Transformations in combination with human knowledge

For this type of transformation, additional knowledge about the environment, the sensors, or some objects in the environment can be utilized. If a certain type of information is known to be connected to a certain object or situation, this knowledge can be used to derive additional information from it. It can either be used to add the connected object to the map, or somehow represent the situation or a changed condition that could be derived from the sensor data.

For example, the measurement for humidity sensors alone are not very useful, but by using the knowledge that increased humidity can cause slippery underground it can become very useful. The transformation for this is to get the area that this sensor covers and to create a speed map for this certain area. Such a map, of course, is an extension to the traditional occupancy grid, but one that is already included in the MIRA framework and thus mentioned here. Depending on how severe the effect of the humidity is, there can be different levels that limit the speed of the robot in increments.

As the next example, consider the sensor measurement of a rotary encoder. Again, this measurement alone does not provide any usable information about the environment, so it has to be interpreted in combination with an object in order to be useful. This can either be in combination with a robot, giving information about the travelled distance or the turned angle, or it can be in combination with a door hinge in which case the sensor provides detailed information about the state of a door. Using the measured angle, the position and the length of the door, this can be translated into an occupancy grid showing the current state of the door.

This angle information can be obtained using different sensors, as mentioned before (see 3.3.2). So, using a camera, a laser or infrared sensor to get angle information would still result in this transformation to include the door into the map.

Another example could be the heartbeat of a robot. Here, it is assumed that the heartbeat also carries the information about the position of the robot. On its own this information cannot be included into the map, but when the robot model is known - or even just the rough dimensions - the robot can be added to the map. This can look like a two-dimensional representation of the robot or simply like a circle or rectangle which approximately represent the robot's appearance.

As a last example, the object detection shall be considered. As previously stated, the point cloud data from RGBD sensors can be used to include generic geometric objects into the map. When models for certain objects are known, this information can be used to include more than just a geometric representation of the object.

By detecting people, for example, a person can be represented as a circle in the map and in addition to that, a speed map for the area around this person can be generated. A speed map is an extension for occupancy grids that is used by MIRA to define certain areas in which robots are only allowed to drive at a certain speed. Such a map can be used to reduce the risk of accidental collisions in areas where the sensors of the robot cannot register the person.

As a second example for object detection the detection of robots shall be mentioned. If additional information is known about the type of the detected robot (e.g. the maximum speed), then not only can the robot be added to the map but also a prediction could be made to tell if the detected robot is still an obstacle when the planning robot enters the area. There might exist the possibility that the detected robot will already be gone by the time the planning robot is able to reach this position. In this case the planning robot would not have to use an alternative path to its destination.

3.5.3 Transformation chains

This section shows how the aforementioned transformations can be combined in order to convert one type of information to another type, although no direct transformation between those two was defined.

For example, through object detection a person is found to be present in a room. This could lead, as mentioned, to two transformations already:

1. transformation into an occupancy grid for the area that this person occupies
2. transformation into a speed map for the general area where the person is in

In addition to these two, the detection could also trigger a third transformation that produces a count event or a NoGo map as the result. Such a NoGo map is, like the speed map, an extension included in MIRA that prevents robots from entering certain areas. An extension for these count events could be that not only the amount of people or objects is being counted, but also the occupied area of these objects is being monitored. In the case that a certain percentage of the room is occupied by these detected objects, a NoGo map is provided for this room, preventing the robot to enter it until there is enough space available again.

So in a case where a robot specifically subscribes to count events, the system could check for transformation chains that provide the wanted event, even though it could not be generated directly. For example, a simple count event could be generated by an array of range scanners that detect when an object enters or leaves a room. The same event could be generated by RGBD sensors. By detecting that there is some object in the room, it can be added to the map and its occupied area can be computed. Using this information, a count event for this room can be generated and provided to the robot.

These transformations are currently stored in a list and only simple transformations are possible and they can only be found by pattern matching. In the software, the transformations are defined for specified pairs of publishers and subscribers. So, in order to create transformation chains, every single transformation in the chain has to be defined by pairs of publishers and subscribers. When one information type in such a pair is not available, the transformation can not be executed. It is planned to store the information types and their transformations in a graph structure, which allows to automatically find all possible sources and transformations for a needed information type. Using this structure, transformation chains would not have to be hard coded, but could be constructed dynamically when all necessary information types are available.

3.6 Summary

This chapter outlined a concept for combining sensor networks with robots in order to expand the robots' perception to areas that they cannot observe themselves.
For this, a selection of suited sensors was presented and how they can be used to get different information about the environment by using transformations. It was explained how these transformations could be used to convert certain types of information into others and how these can be stringed together to derive completely different types of information.
Furthermore, this chapter described that the sensor measurements are distributed as sensor events. These structure of these events was described and that each unique set of values in an event describes its event type.
In the last section the transformation framework was described, that it currently only works when the transformations are explicitly defined. Furthermore, it was mentioned that transformation chains are possible, but have the same constraint of needing explicitly defined pairs of input and output types. It is planned to improve this in the future. The chapter was concluded by describing the planned improvement for storing event types and their possible transformations.

The next chapter describes implementation details of a scenario that is based on this concept, which includes the use of different robotic frameworks and how they interact with each other.

4 Implementation

This chapter shows the implementation of a robotic scenario based on the concept mentioned in the chapter before. The scenario consists of a robot, a sensor network, and three different types of information that are transformed into occupancy grids. The robot is provided with these occupancy grids and includes them into its static map of the environment. With this map the robot is able to navigate using current information.

The goal is to implement a scenario with a robot that gets current information about the environment from a sensor network. This scenario can be divided into some main components that needed the most effort to implement. These components are the sensor network where all the data about the environment originates from, the application side where this data is used, and the communication between those two components. In this case, the sensor network is using the ASEIA framework to handle communication between sensors and applications. ASEIA is using the publish-subscribe mechanism of the ROS framework to provide the information gathered by the sensors. The information, in order to be of use for the robot, has to be transformed into occupancy grids. After this transformation is done, the occupancy grids have to be published from one framework (ROS) to the other (MIRA), which us used by the application. In this case, the application is a robot using the MIRA framework for navigation. The robot is provided with a static map of the environment at the beginning of the scenario and uses the information from the sensor network to update this static map with current information from the sensors.

One important detail of this implementation is the use of two Middlewares for Robotics. For the communication between the sensor side and the map merger the “Robot Operating System” (ROS) was used and further processing of the sensor data was done with the “Middleware for Robotic Applications” (MIRA). The next section will give an overview of both frameworks and why the decision was made to use both.

4.1 Robotic frameworks

In chapter 2 some basic tasks were mentioned that most mobile robots have to carry out in their work environment. Since each robot is built differently, acts and reacts differently, the basic algorithms (e.g. for navigation, etc.) have to be configured with some parameters in order to work correctly with this robot. The underlying algorithms for path planning and navigation, for example, stay the same but need information about the robot in order to work correctly in the environment. One example for this could be just the dimensions of the robot. The planner has to know them in order to plan around narrow passages that one robot could drive through, but another one cannot. Another example would be to take the mobility into account. It makes a big difference to the planner if a robot can turn 180 degrees on the spot or if it has a bigger turning radius and thus preventing it to turn in narrow spaces.

One possibility to handle this situation could be to rewrite the algorithms for each robot configuration, but that would lead to much duplicated code that is hard to maintain, because one has to know everything about every configuration. Introducing new algorithms would lead to a huge overhead for integration for the different configurations. So this approach would only be manageable in smaller projects, and even then it would not be optimal.

One alternative to this is implemented by robotic frameworks like ROS and MIRA for example, which will be explained in this section. These frameworks come with a multitude of algorithms that can be used on many different robots. Here, the basic algorithm is not changed, but rather configured using files that contain information of the robot that is currently using this functionality. This approach enables the developer to easily introduce new software, update existing one, and do so without changing the software for each robot. The software is in one place and updates are only applied there, making it easier to focus on development of new functionality, rather than maintaining old one.

On a more abstract level, these frameworks provide the user with lots of functionality that is needed for many robotic scenarios. This includes software for all kinds of problems like mapping, localization, map merging, path planning, and many more. In order to have this framework and its software running on different robots, it provides drivers for lots of hardware, like motors, sensors, and other periphery. In addition to that, the framework handles the communication between the different parts of the robot, including both software and hardware on one or multiple robots.

This section mentions two robotic frameworks, namely ROS and MIRA, that both handle the communication differently, yet with the same underlying messaging pattern. Both frameworks were used in the course of this thesis for different tasks of the scenario. ROS is part of the communication system that was used in the sensor network for data aggregation, whereas MIRA is the framework that is used by the robot, which was used to navigate the environment, merge maps, and overall control the robot. Although other robotic frameworks exist [Hoc13], only these two were used in the context of this thesis. Both systems were already integrated in some part of the project, so choosing any different framework would have resulted in additional integration overhead just for the cause of working around systems that were already in place. So the following sections focus on describing the frameworks in more detail.

4.1.1 Robot Operating System (ROS)

One of the most used frameworks for software in robotics is the **Robot Operating System** [QCG+09]. As mentioned, these frameworks were created with specific design goals in mind, that the developers deemed most important. For ROS these design goals encompass the following:

- Peer-to-peer
- Multi-lingual
- Tools-based
- Thin
- Free and Open-Source

These design goals will be described in short in the following paragraphs. A more detailed description about these design goals can be found in [QCG+09].

The *peer-to-peer* approach for communication handling in the framework was used to avoid traffic overhead. This topology requires a lookup mechanism, so processes can find each other at runtime [QCG+09]. In ROS this mechanism is called *master*.

The goal to support *multiple programming languages* was introduced to make full use of each language and its characteristics. So ROS is, if possible and sensible, implemented natively in each language. As the main languages, C++, Python, Octave, and LISP are mentioned [QCG+09]. To support multiple languages, ROS makes use of technologies that have an implementation in most current languages. So, for example, the configuration and negotiation of connections happens in XML-RPC. The usage of a language-neutral interface definition language further promotes cross-language development. It describes the messages that are being sent between the modules in a short text file containing the fields of the messages.

The *tool-based* approach of ROS is shown by the fact that a microkernel approach was chosen when ROS was designed. Functionality is then added by building small tools for ROS. Each tool has a very specific task to fulfil and the collaboration between these tools leads to more complex functionality. The lost efficiency using this approach is believed to be overshadowed by the gained stability and complexity management [QCG+09].

The design goal to make ROS *thin* means, in short, that the development of drivers and algorithms in stand-alone libraries is encouraged. These libraries, which don't need any dependencies on ROS, are easier to be reused, debugged, and tested than software that is part of a big software system. Each library can be debugged and tested separately, or at least without the need to recompile and restart the whole software. Using ROS, it is easy to build the library and implement changes while the remaining modules are left running. This yields easy and fast debugging and testing, and such fast prototyping of new functionality [QCG+09].

Thanks to the fact that ROS is *free and Open-Source* the code is publicly available, so all algorithms and modules can be tested, debugged, and extended by the community. It also makes it very easy to contribute new software and further expand the project. ROS is distributed under the BSD license and the module-based architecture makes it possible, to use different licenses for each module, since they don't need to be compiled into a single executable.

ROS will be referenced later in this thesis. To put the notions into context, the last paragraph will explain in short the fundamental concepts of ROS [QCG+09]:

Nodes denote the processes that run certain programs. The notion is interchangeable with "software module". The whole system at runtime normally consists of multiple nodes, where each node normally fulfils one task or runs one algorithm.

Messages are used to communicate between nodes. Messages are data structures that are passed between nodes at runtime. Messages can be nested into other messages, making it possible to create more an more complex data structures.

Topics are used by nodes to send and receive messages. Topics themselves are simply strings like "map" or "sonar" that, in the best case, describes the type of message that is communicated in this topic. Nodes that are interested in the data can simply subscribe to the topics and receive new data when it is available. There can be multiple publishers and subscribers for each topic. Nodes can publish to and subscribe to data from multiple topics.

Services are defined by a string name and a pair of strictly typed messages: one for requests and the other one for responses. Contrary to topics, services can only be advertised by one node at a time with a certain name (e.g. there can only be one service "local_map").

4.1.2 Middleware for Robotic Applications (MIRA)

MIRA is the framework for the SCITOS robots, that are built and sold by the company "metralabs" [ELS+12]. Like ROS, it covers drivers for hardware, algorithms for robotic scenarios, and communication between the software modules. The focus of MIRA is a little different from ROS. It was used because it is well integrated in the SCITOS G5, the robot that was used for navigation. The software stack for navigation, localization, and mapping works well out of the box and could therefore be used without greater integration overhead.
Since the design goals of MIRA differ from ROS, a short overview shall be given here [ELS+12]:

High performance and low latency: the communication techniques must have a low CPU footprint to be able to use the middleware on robots with low computational power and to leave enough capacity for the actual algorithms.

Easy to learn and use: for this reason, all functionality should be realized by using features of the designated programming language (C++, Java, etc.) only. In contrast to ROS, no additional interface description meta-language is necessary.

Small number of different concepts that are used for a large number of different features to assure ease of use[...]

High usability: the features of the middleware must be intuitive and accessible with little code and configuration overhead.

Foolproof: programming mistakes like type mismatches when using the communication techniques should be reported by the compiler or at run-time instead of leading to unexpected results.

Robust and reliable: the MIRA software system must not contain a central component that - in case of failure - might stop the entire system from working. Moreover, a high software quality is assured by strict software reviews and by many automated test cases.

One problem of using two different frameworks is that even basic mechanisms can be implemented differently. For example, both frameworks use the publish-subscribe method for their communication. However, the implementation was done differently, preventing a direct communication with them. The next section will describe how this problem can be handled.

4.2 Communication between frameworks

Both frameworks are needed in the scenario, one for the communication in the sensor network, and the other for navigation and control of the robot. As mentioned, the different implementation of these frameworks prevents a direct communication between them. In order to provide communication between those frameworks, a program has been implemented that translates messages from ROS to MIRA and vice versa. This adapter is basically a program that is both a ROS node and a MIRA unit. Here, “node” and “unit” both describe a software module for their respective framework. So when a message from ROS is received, a callback function is executed that takes the content from this message and builds a new MIRA message with it. This message is then forwarded to MIRA where it can be processed further.

Two details are important here, when it comes to certain information in these messages. The first is that ROS and MIRA use different coordinate systems, so before a map can be used in MIRA, a transformation into the coordinate system of MIRA has to be done. The second thing to be aware of is that occupancy grids are represented differently in ROS and MIRA in ROS values from -1 to 100 are stored, where -1 means that no information is known about this cell. However, in MIRA values from 0 to 255 represent the occupancy and no explicit value is used, when no information for a certain cell exists. So these values have to be altered, too, in order to be correctly interpreted in the receiving framework. The information to be shared between those frameworks has to be known beforehand because the translation between messages from each ROS topic to a MIRA channel (and vice versa) has to be done manually.

Now that the communication between frameworks was covered, the next section will describe the information gathering in the sensor network.

4.3 Sensor network

In short, the sensor network delivers measurements from various sensors to applications that are interested in this information. By using ASEIA as the foundation for communication, applications don't subscribe to specific ROS topics. Instead, subscribers can subscribe to a certain type of information that can be gathered by different sensors. The sensor data is delivered in the form of events, where each event has a specific set of values that defines its type. Applications can subscribe to such a type and when the sensor network can provide this type of information, a channel is established between the information source and the subscriber. In case the requested type of event is currently not available, it is checked if there exist transformations from any other event type to the requested one. For example, if a publisher requests the angle of a door, but no publisher offers this kind of information, ASEIA can look for transformations that have the angle as output. When, for instance, a distance sensor exists that looks in the direction of a door, the distance value from the sensor can be transformed into an opening angle of the door and be provided to the subscriber.
In ASEIA, finding these matches is currently done by looking up pairs of information sources and subscribers. If a matching pair is found, the event from the source is transformed into the event requested by the subscriber. As an example for such an event, let's consider a distance event. The actual sensor measurement is contained in the value "Distance", which is stored as a

16 bit integer, the unit is meter with a scale of 1/100 which means that the distance is stored in centimeters. The additional information in the event is used to identify the sensor, its position, the age of the message, and scale its contained data. All values together form the signature of the event which is used to dynamically find matching publishers and subscribers.

One feature of ASEIA that is important for this scenario is the ability to publish the data to “normal” ROS topics, meaning topics that are not part of the ASEIA network. This is used in this scenario to get data from the sensor network to the robot running MIRA. The adapter between ROS and MIRA is subscribed to certain ROS topics that get their data from ASEIA. Thus, it can get information about the environment without being a native part of the ASEIA network.

Having covered the sensor network and the message forwarding from the network to the MIRA application, it is now time to cover the application side of the scenario.

4.4 Application side

On the application side there can be any software module that is able to subscribe to data from the ASEIA network, meaning any ROS node or software that can communicate with ROS. In the scenario of this thesis, on the application side there is a robot that subscribes to occupancy grid data. While in theory there could be many possible subscribers, in this scenario there is only one, a SCITOS G5 running MIRA.

It subscribes to two different map channels, one containing a static map of the environment, and the other containing the maps that were generated using the sensor information from the sensor network. These two channels are then used to produce a merged map containing the building with all object that were detected by sensors.
The merging was done on the application side to separate any application-specific functionality from the part of the system that should only aggregate and transform sensor data to the application itself.

Of course, other applications than MIRA robots are possible, but at the time of this thesis there were no other suited robots available or functional for this scenario. A simple application would be a robot running ROS, where a translating adapter would be no longer required. Of course, the map merging would have to be done in ROS now, but there exist packages that provide this functionality, like “map_merger” [And] for example.

4.5 Transformations

In this paragraph, a selection of three transformations is described that are currently used to display different kinds of information in the occupancy grid of the robot.

The first transformation converts angle information to an occupancy grid. For this, it is assumed that angle events are associated with doors, so angles will be transformed into occupancy grids of doors. The source for angles can be a rotary encoder, that is able to directly provide the information, or a transformation that takes another measurement and converts it into an angle. The position of the measurement is known since it is part of the event data. It is assumed to be the position of the door hinge. Using this position and the angle it is possible to generate an occupancy grid with a line representing the door. Here, the Bresenham algorithm for drawing lines [Bre65, bre] was used.

The second transformation converts a distance to an angle. This is an example for a source of angle information without the sensor being able to directly provide angle measurements. If a distance sensor is near a door and detects the distance to said door, the opening angle can be computed if the position of the door hinge is known. Thus, a distance can be used to generate an occupancy grid of a door when two transformations are performed.

As the third and last example the transformation from a position to a grid shall be mentioned. Here, it is assumed that a heartbeat containing a position is sent by a robot. This heartbeat does not have to carry any additional information, just the existence is proof that the robot has not lost it’s connection to the network. Since every event has a position in it, this information can be used to generate an occupancy grid containing a circle that represents the robot. If different robots are in the scenario, the heartbeat could be extended by an ID with which the robot and its dimensions can be retrieved from a lookup table. Here, the position is used as the center of the circle and with the radius of the robot, the circle can be drawn using the equation of a circle.

4.6 Summary

This chapter provided information about decisions in the implementation, including why two robotic frameworks were used instead of only one. The different parts of the scenario, namely the sensor network, the application side and the communication between frameworks were described and how they interact with each other. The last section covered how sensor measurements are transformed into occupancy grids for the robot. Using this scenario, an evaluation of the approach has to be done to see if everything works as expected or if the robot is not able to use this information efficiently. The source code for the implementation of ASEIA will be made public at `https://github.com/steup/ASEIA` and `https://github.com/steup/ASEIA-ROS` by Christoph Steup in the course of his dissertation. Further, the source code for the implementation of the translation module between ROS and MIRA will be made available at `https://github.com/dirksteindorf/master-thesis`.

In the following chapter an evaluation of this approach is described and documented.

5 Evaluation

This chapter describes experiments that are aimed to assess the concept developed in this thesis. A simple scenario is constructed where a robot gets three basic types of information about its environment. When new information arrives that influences the current navigation, the robot has to act accordingly. If the robot gets the information that its current path is blocked, the robot is supposed to react by planning an alternative route. If such a behaviour can be observed, the concept is working as intended. Otherwise, further investigation has to be conducted to find a way to provoke the intended behaviour.
The first part of this chapter covers the adaptation to dynamic changes in the environment, and the second part covers performance and network traffic measurements that were conducted after some unforeseen errors occurred.

5.1 Adaptation to changes in the environment

In theory, the three transformations mentioned in the last chapter can be used to generate occupancy grids, which can then be merged with the static map of the robot. The robot should then be able to avoid newly emerged obstacles and plan a new path to its destination.

To test this theory, an this scenario was conducted using the lecture hall as the environment. It provides two doors which can be used to block the path that the robot would normally use. Using events that tell the robot that one door is now blocking its current path, the robot can be forced to use the other door to reach its destination. When everything works as intended, the robot should discard its initial path and use one that leads through the open door. By trying different start positions and destinations, it can be observed if the robot shows the expected behaviour and how long it takes to react to new data. In this scenario all sensor information was published by simulated sensors in order to save preparation overhead that would have included setting up the environment and configuring the sensors. Instead, these simulated sensors were used to allow quick adaptations of the scenario. Figures 5.1 and 5.2 show this intended behaviour using the lecture hall example.

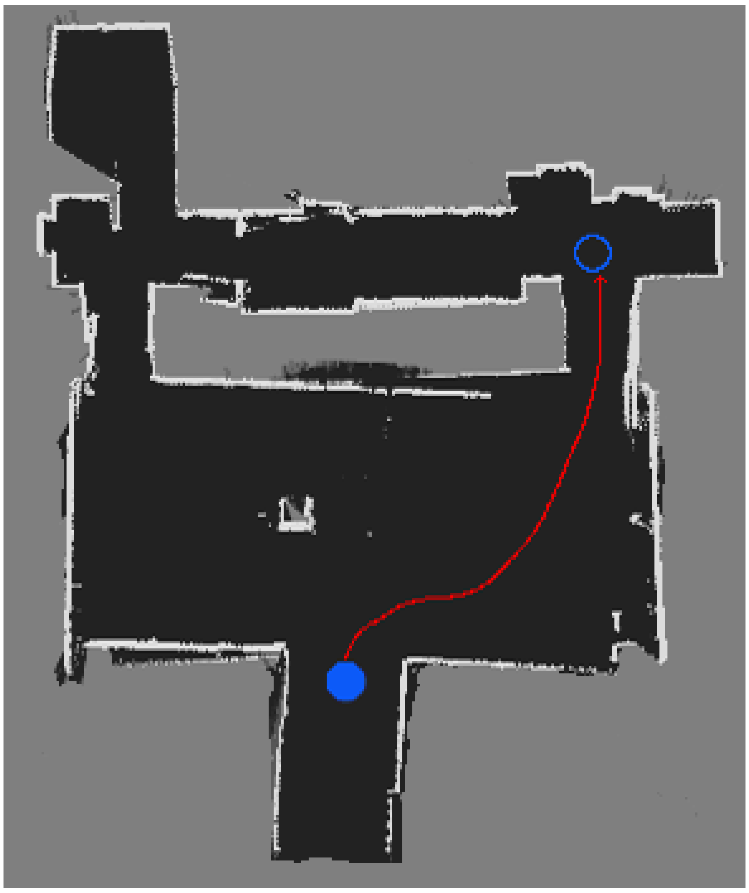

Figure 5.1: The initially planned, short path

Sadly, after recording a map of the environment and preparing the required publishers, MIRA was unable to reliably use the newly generated map for navigation. When a path to the destination could be planned, the robot would only move haltingly to this position, often never reaching it and aborting the navigation. In some cases, MIRA was not able to plan a path even when the destination was one meter in front of the robot with no obstacles in between. On one hand, the observation could be made that a new path was planned when the previous one was suddenly blocked by the door. On the other hand, it was not possible to observe the intended resulting behaviour of the robot

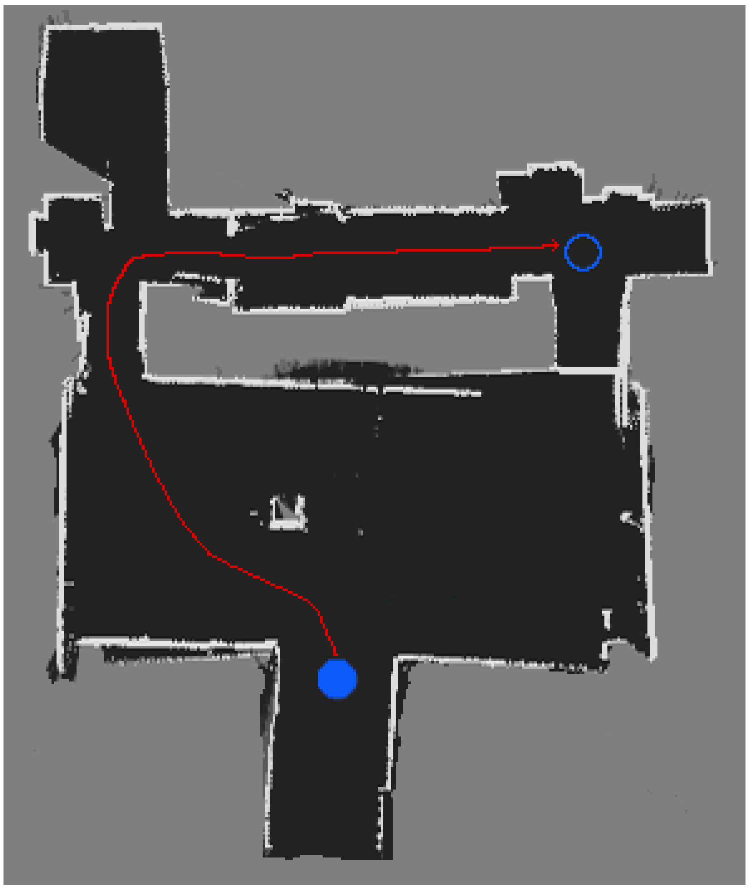

Figure 5.2: The alternative path, planned as a reaction to the closed door

because MIRA discarded this path within one second. This discarding behaviour could be replicated, but the robot was not able to follow this path to its end. The problems here were that no error message was displayed and that the navigation stack from MIRA is closed source, so troubleshooting was not an option.

After this unsatisfactory first evaluation attempt, a second one was prepared. In this attempt, the robot running MIRA did not do any computations on its own. in fact, the robot was not even provided a map for navigation. The whole navigation computation was done in ROS using “hector_slam” [Mey]. For this, the robot provided only its laser scanner and odometry data, so that mapping and navigation could be done on the ROS side. Then, it was supposed to receive driving commands from ROS so that it could follow the planned path. Sadly, this attempt failed too, this time with an error message from hector_slam: “Transform failed during publishing of map_odom transform: Lookup would require extrapolation into the future.” This error could not be fixed, and even a similar question asked in the ROS community was never answered [jac]. But seeing that this seemed to be a timing problem, there was reason to suspect that the transformations were taking too long to generate a map in time. In order to investigate this assumption a last evaluation attempt was made, in which the used CPU time of the transformations and the network traffic generated by publishing the events were measured.

5.2 Measuring CPU time and network traffic

For the following measurements the following hardware and software was used:

- Lenovo X220 Laptop
  - Intel Core i5-2520M Processor (2.5GHz, 3MB Cache)
  - 8GB RAM (DDR3 - 1333MHz)
  - 320GB Hard Disk Drive, 5400rpm
- wireshark 1.10.6 (v1.10.6 from master-1.10) [wir]
- perf version 3.13.11-ckt36 [per]
- ROS indigo
- Ubuntu 14.04

5.2.1 CPU time measurements

The CPU time was measured to investigate if the computation of transformations takes too long, so that the aforementioned timing error occurs. Using the linux tool “perf”, the overall CPU time spent on computations of the manager was measured. The manager module, which is part of the ASEIA network, is the part of the sensor network where the transformations are performed and the communication channels are established.

To get a measurement of how much time the CPU spent on computations (in the userspace) for the manager, the following command was used.

Iteration	CPU time (1 e/s)	runtime	CPU time (10 e/s)	runtime
1	2919.14 ms	599.63 s	13040.02 ms	599.80 s
2	2843.21 ms	599.65 s	12571.84 ms	599.65 s
3	3007.38 ms	599.63 s	9565.02 ms	599.63 s
4	2797.25 ms	599.60 s	11497.51 ms	599.72 s
5	2641.03 ms	599.64 s	10087.44 ms	599.73 s
average	2841.60 ms	599.63 s	11352.36 ms	599.70 s

Table 5.1: CPU time used by the manager

```
perf stat -e cpu-clock:u
```

These measurements were conducted for two publishing rates of the publishers, namely one event per second (1 e/s) and ten events per second (10 e/s). For each rate, the measurement was conducted over a period of ten minutes and was repeated five times. Table 5.1 shows the results of the measurements.

As can be seen, when the publishers create one event per second, the manager uses on average 2841,60 ms from a total runtime of 599,53 s, which evaluates to 0.47% of the total available CPU time. When ten events per second are published, the manager on average uses 11352,36 ms from a total runtime of 599,70 s, which evaluates to $1,89\%$ of the total available CPU time.

In order to get information about the individual time the CPU spent on a transformation, another measurement was conducted. Here, it was observed how much time the CPU spent in the userspace when the manager was executed. The following command was used to generate a recording of the time the CPU spent on the transformations.

```
perf record -g -e cpu-clock:u
```

As before, this was observed for two publishing rates (1 event per second and 10 events per second), the measurement was conducted over a period of ten minutes and was repeated five times for each rate.

An examination of these recordings showed the percentage of computation time the manager was occupied by the transformations. Using the following command, a recording could be observed:

```
perf report
```

Iteration	share of used time (1 e/s)	share of used time (10 e/s)
1	6.45%	6.84%
2	12.50%	8.82%
3	7.89%	6.76%
4	9.38%	5.68%
5	7.41%	8.09%
average	8.72%	7.23%

Table 5.2: Percentage of computation time for converting angles to occupancy grids

Iteration	share of used time (1 e/s)	share of used time (10 e/s)
1	3.23%	7.82%
2	18.75%	7.52%
3	13.16%	6.20%
4	3.12%	6.94%
5	11.11%	6.65%
average	9.87%	7.02%

Table 5.3: Percentage of computation time for converting distances to angles

Table 5.2 shows how much CPU time the manager spent on computing transformations from angles to occupancy grids. When the publishers produced one event per second, the manager spent on average 8.72% of its time on this transformation. This evaluates to $247.78ms$ of CPU time, which equals to 0.0413% of the total available CPU time. In the case the publishers produced ten events per second, the manager spent on average 7.23% on this transformation. This evaluates to $820,77ms$ of CPU time, which equals to 0.1368% of the total available CPU time.

Table 5.3 shows how much CPU time the manager spent on computing transformations from distances to angles. When the publishers produced one event per second, the manager spent on average 9.87% of its time on this transformation. This evaluates to $280.46ms$ of CPU time, which equals to 0.0467% of the total available CPU time. In the case the publishers produced ten events per second, the manager spent on average 7.02% on this transformation. This evaluates to $796.93ms$ of CPU time, which equals to 0.1328% of the total available CPU time.

Iteration	share of used time (1 e/s)	share of used time (10 e/s)
1	19.35%	4.56%
2	3.12%	6.86%
3	7.89%	6.20%
4	9.38%	5.36%
5	7.41%	5.78%
average	9.43%	5.77%

Table 5.4: Percentage of computation time for converting positions to occupancy grids

	min (1 e/s)	max (1 e/s)	min (10 e/s)	max (10 e/s)
manager	2641.03 ms	3007.38 ms	9565.02 ms	13040.02 ms
angle to grid	183.28 ms	355.20 ms	644.81 ms	1001.27 ms
distance to angle	91.78 ms	532.80 ms	703.84 ms	887.75 ms
position to grid	88.65 ms	549.84 ms	517.66 ms	778.77 ms

Table 5.5: Summary of CPU time used by the different computations

Table 5.4 shows how much CPU time the manager spent on computing transformations from positions to occupancy grids. When the publishers produced one event per second, the manager spent on average 9.43% of its time on this transformation. This evaluates to 267.96ms of CPU time, which equals to 0.0446% of the total available CPU time. In the case the publishers produced ten events per second, the manager spent on average 5.77% on this transformation. This evaluates to 655.03ms of CPU time, which equals to 0.1092% of the total available CPU time.

In table 5.5 the minimum and maximum used CPU time is presented for each computation.

It can be seen that the manager uses only a small fraction of the total CPU time, namely 1.89%. Each transformation that is computed, takes only a fraction of the manager's computation time. In this case, each transformation uses less than 0.5% of the total available CPU time. So, in conclusion, the transformations can be excluded as the cause for the timing problem in the previous evaluation attempt.
As a last evaluation step, the network traffic is measured in order to see if the

amount of data is so big that it takes a lot of time to be transmitted between the manager and the hector_slam package.

5.2.2 Network traffic measurements

The last step in this evaluation is to measure the traffic generated in the scenario in order to see if the size of the events was so big that the timing problem from the previous attempt could be explained by huge network traffic. Specifically, the traffic generated by angle events, door grid events, and robot grid events was measured. The measurement was done using "Wireshark" [wir] which provides a tool to visualize the average generated traffic (IO Graph). For each event type, the traffic was measured over a period of 10 minutes, once with a publishing rate of one event per second and once with a publishing rate of 10 events per second. For angle events there were two publishers present, one that directly provided angle events and one that took distance events and transformed them into angle events. For both robot and door grid events there was only one publisher present respectively.

The figures 5.3, 5.4, and 5.5 show the average traffic generated by the mentioned events when the publishers generate one event per second. In order to fit the whole measurement on the screen, the axes were scaled to show the traffic in bytes per 10 seconds.

The figures 5.6, 5.7, and 5.8 show the average traffic generated by the mentioned events when the publishers generate ten events per second.

As expected, when the events are published ten times per second, the generated traffic is tenfold, too. The maximum amount of generated traffic comes from the door grid events, which generate 7620 bytes per second when ten events per second are published. The sum of all events combined is 15240 bytes per second. This amount of data is not enough to generate delays, since all the software modules were on the robot and thus, the timing problem of the previous attempt cannot be explained with the traffic, either.

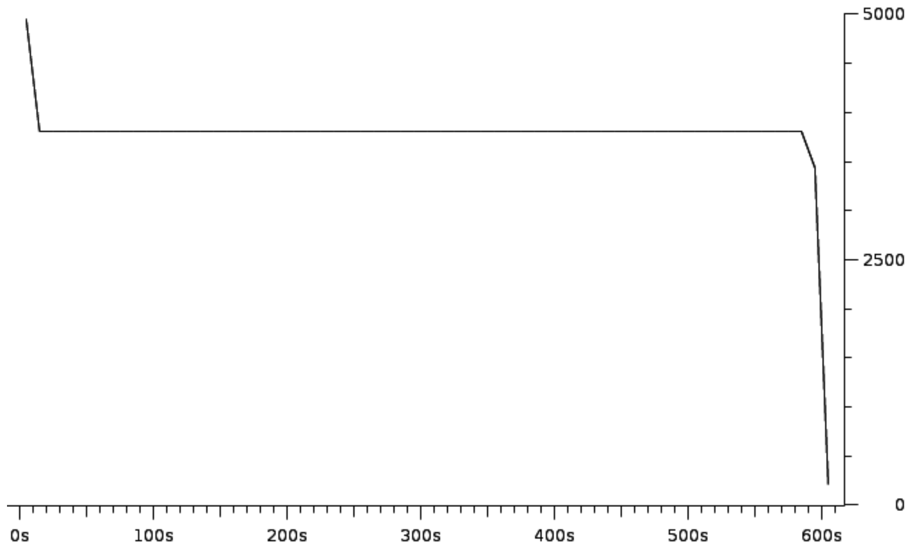

Figure 5.3: Angle events generate traffic of approximately 380 bytes per second

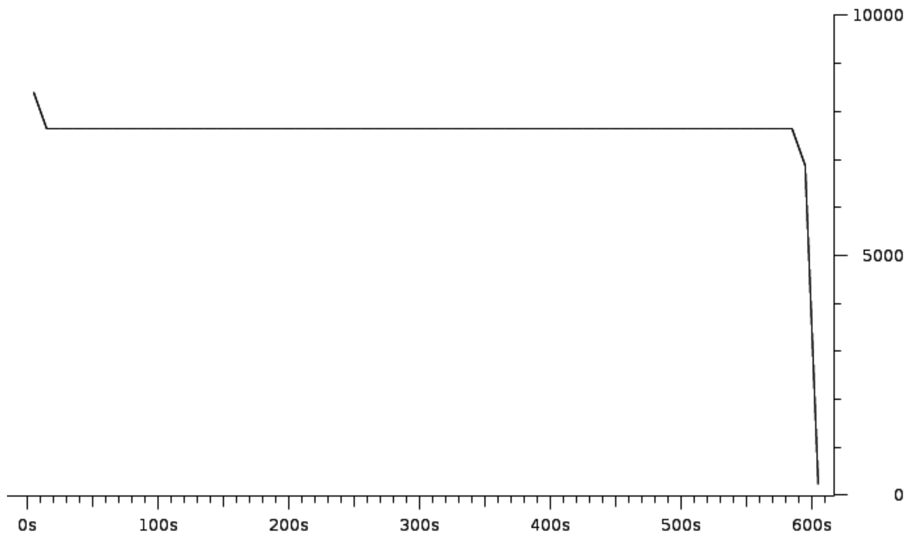

Figure 5.4: Door grid events generate traffic of approximately 762 bytes per second

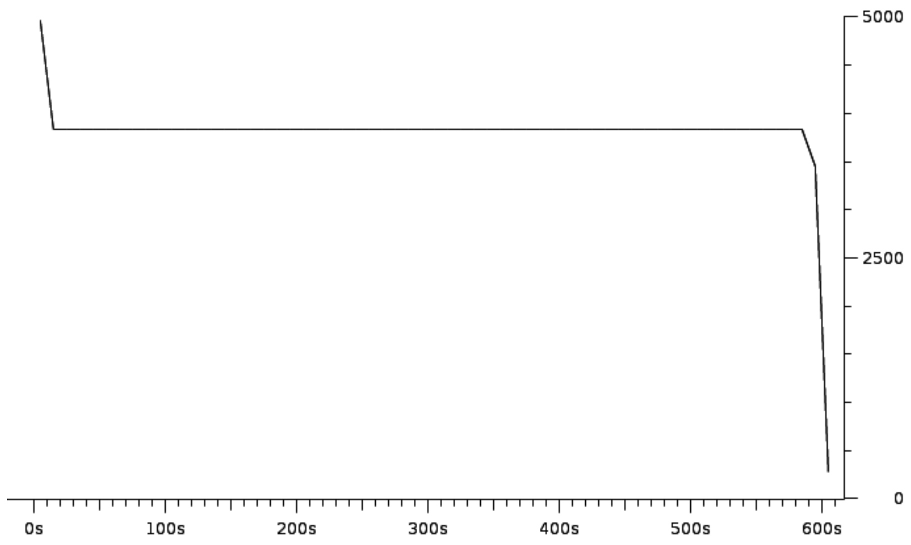

Figure 5.5: Robot grid events generate traffic of approximately 382 bytes per second

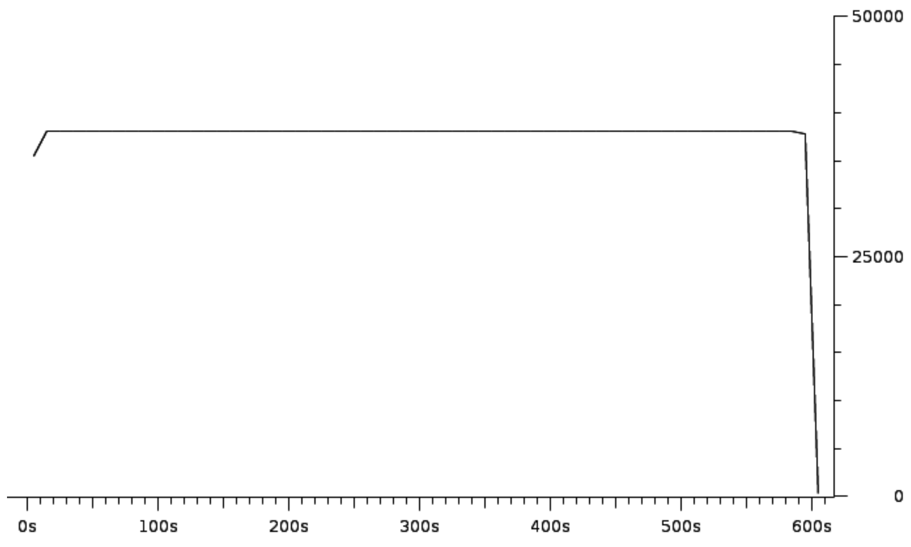

Figure 5.6: Angle events generate traffic of approximately 3800 bytes per second

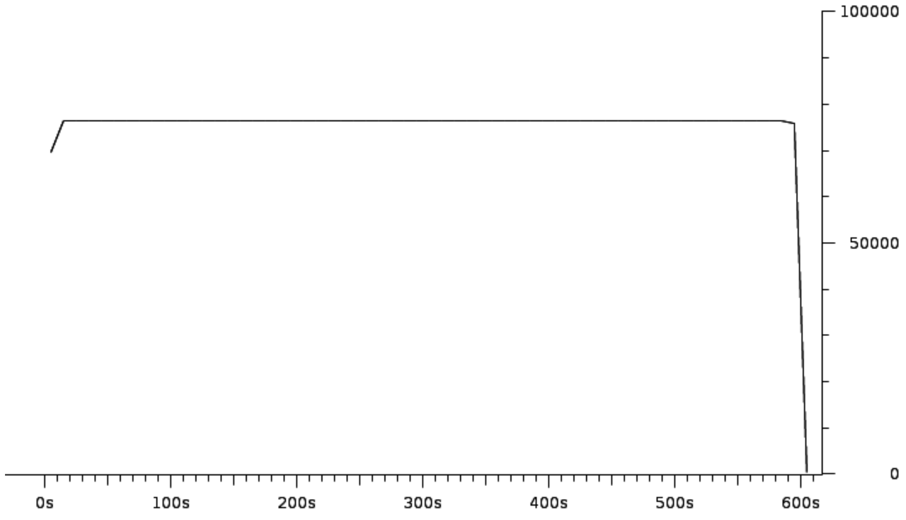

Figure 5.7: Door grid events generate traffic of approximately 7620 bytes per second

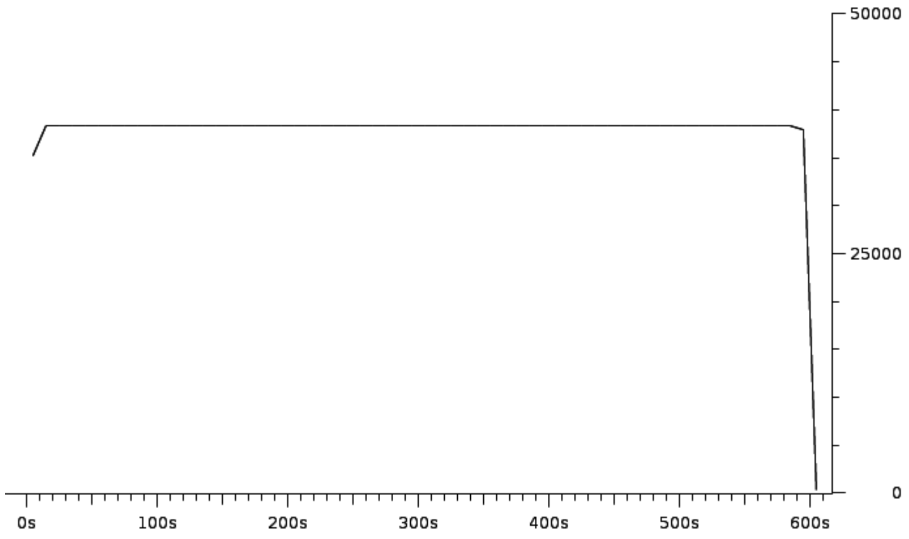

Figure 5.8: Robot grid events generate traffic of approximately 3820 bytes per second

5.3 Summary

In this chapter, three evaluation attempts were presented. Two of them had the goal to examine if the robot acts as envisioned when new map data is available that invalidates the current path. Both of them had to be aborted unsuccessfully because of an errors in the robotic frameworks that prevented the robot's navigation. The third evaluation attempt had the goal to examine if the problem might be caused by the transformation framework. Performance and network traffic measurements were conducted that both show that the transformation network can not be the cause for the errors, leaving only internal problems of the framework as the origin of these problems. The data collected for this evaluation will be made available at `https://github.com/dirksteindorf/master-thesis`.

In the next chapter a short summary of this thesis, and an impression of future work is given. Possible next steps will be mentioned, that can be taken to build upon this first implementation of this concept.

6 Outlook

In the course of this thesis, a new approach for dealing with dynamic environments in robotic scenarios was presented. After a brief introduction to the topic of utilizing robots in many different situations in everyday life, the technical background for the new approach was provided.
The chapter covering this technical background gave an overview of the basic navigation tasks that a mobile robot has to carry out in indoor scenarios. In this context, this chapter included an overview of how a robot can gather information about itself and the environment and how this information can be processed. This processing of information and many other common tasks are part of frameworks that provide this functionality for reuse in different robots and scenarios.
After an overview of different map representations, some advanced navigation tasks were covered, which showed how navigation in dynamic environments can be implemented. These existing approaches still have some weaknesses that make some of them very complex or difficult to implement. The approach of this thesis takes some of the ideas of the existing methods and combines them in a way, so that these weaknesses are not adopted.

The concept chapter then described how this approach should look like and how it extends some of the mentioned methods. A description of information gathering and various applications for sensors was given. Following this, it was explained how the information is shared between the sensors and the robots in order to to get all the needed information to each robot. Furthermore, a filtering mechanism was introduced that enables the robot to get only the data that is currently of interest. The chapter was concluded by detailing a transformation framework that allows conversions of one information type into others.

Afterwards, the implementation details of the approach were provided which, amongst others, covered the utilization of different robotic frameworks and their communication amongst them. For this scenario, a few sensor informations were selected and possible transformations between them were implemented. The informations were chosen to show different capabilities of the system. This includes basic functionality like information sharing in the sensor network, and certain transformations from sensor measurements to occupancy grids.

In the last chapter, this scenario was evaluated. The evaluation started with two attempts that could not brought to a conclusion due to problems with the robotic frameworks that could not be fixed. Then, a third attempt was conducted, which was aimed to further investigate if the problems could have been caused by the implementation of the transformations and their publication. This last attempt was able to show that this implementation could not have been the cause, due to the low complexity of the transformations. Although the initial evaluation attempt could not be concluded, it was possible to show that the approach is suited to be used with bigger sensor networks in dynamic environments. This is possible due to the low computational complexity and a resulting low network traffic, which allows for the utilization of much more complex transformations and the gathering of more sensor information.

The next step from here could be to repeat the originally planned evaluation with more time for troubleshooting. Using another robot running ROS could also be beneficial, since this allows to remove the communication module that translates between frameworks. When this evaluation can be brought to a conclusion, the implementation of transformation chains could be an interesting next step to show how complex information conversion can be done using multiple simple conversions. In combination with clever filtering, this approach could quickly become a powerful, yet simple way to have robots navigate in dynamic environments.

Bibliography

[AG92] Mongi A Abidi and Rafael C Gonzalez. *Data fusion in robotics and machine intelligence*. Academic Press Professional, Inc., 1992.

[AHM02] Daniel Arbuckle, Andrew Howard, and Maja Matarić. Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 409–414. IEEE, 2002.

[And] Peter Kohout, Torsten Andre. map_merger package for ROS. `http://wiki.ros.org/map_merger`. Accessed: 2016-05-6.

[Asu] Asus. Xtion PRO LIVE. `https://www.asus.com/de/3D-Sensor/Xtion_PRO_LIVE/`. Accessed: 2016-05-6.

[Aug12] Marcus Augustine. Design of Biologically Inspired Topological Maps for Robot Navigation, 2012.

[BD$^+$05] Peter Biber, Tom Duckett, et al. Dynamic maps for long-term operation of mobile service robots. In *Robotics: science and systems*, pages 17–24, 2005.

[Ber13] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[Bor95] Johann Borenstein. Internal correction of dead-reckoning errors with a dual-drive compliant linkage mobil robot. *Journal of Robotic Systems*, 12(4):257–273, 1995.

[bre] Implementations of Bresenhamś Line Algorithm. `http://www.roguebasin.com/index.php?title=Bresenham%27s_Line_Algorithm`. Accessed: 2016-05-6.

[Bre65] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[BSH04] Maxim A Batalin, Gaurav S Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 636–641. IEEE, 2004.

[Dyn] Boston Dynamics. BigDog - The Most Advanced Rough-Terrain Robot on Earth. `http://www.bostondynamics.com/robot_bigdog.html`. Accessed: 2016-05-6.

[ELS$^+$12] Erik Einhorn, Tim Langner, Ronny Stricker, Christian Martin, and Horst-Michael Gross. Mira-middleware for robotic applications. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2591–2598. IEEE, 2012.

[EWK90] Ramez Elmasri, Gene TJ Wuu, and Yeong-Joon Kim. The time index: An access structure for temporal data. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 1–12. Morgan Kaufmann Publishers Inc., 1990.

[Foo] Tully Foote. ROS Driver for the IFM Efector O3D303. `http://www.ros.org/news/2015/07/ros-driver-for-the-ifm-efector-o3d303.html`. Accessed: 2016-05-6.

[Ger] Brian P. Gerkey. amcl package for ROS. `http://wiki.ros.org/amcl`. Accessed: 2016-05-6.

[Gri] Cyrill Stachniss, Udo Frese, Giorgio Grisetti. OpenSLAM. `http://openslam.org/`. Accessed: 2016-04-27.

[Hea] Nick Heath. Amazon, robots and the near-future rise of the automated warehouse. http://www.techrepublic.com/article/amazon-robots-and-the-near-future-rise-of-the-automated-warehouse/. Accessed: 2016-05-6.

[Her] William Herkewitz. Robot Performs Soft-Tissue Surgery By Itself. `http://www.popularmechanics.com/science/health/a20718/first-autonomous-soft-tissue-surgery/`. Accessed: 2016-05-6.

[Hoc13] Sebastian Zug, Thomas Poltrock, Felix Penzlin, Christoph Walter, Nico Hochgeschwender. Analyse und Vergleich von Frameworks für die Implementierung von Robotikanwendungen. Technical report, EOS, EUK, Fraunhofer IFF, Hochschule Bonn-Rhein-Sieg, 2013.

[Inc] Apple Inc. iBeacon Website. `http://www.ibeacon.com/`. Accessed: 2016-05-6.

[Int] Intel. Intel RealSense. `http://www.intel.eu/content/www/eu/en/architecture-and-technology/realsense-overview.html`. Accessed: 2016-05-6.

[iRo] iRobot. Roomba cleaning robot. `http://www.irobot.com/For-the-Home/Vacuum-Cleaning/Roomba.aspx`. Accessed: 2016-05-6.

[jac] jacobsolid. User question about Hector SLAM error message in ROS community. `http://answers.ros.org/question/210645/problem-with-hector_slam-on-ground-robot/`. Accessed: 2016-05-6.

[JY08] Margaret E Jefferies and Wai-Kiang Yeap. *Robotics and cognitive approaches to spatial mapping*. Springer, 2008.

[Mat15] Matthias Kohlmaier. Dieser Roboter soll Flüchtlingskindern Deutsch beibringen, December 2015. http://www.sueddeutsche.de/bildung/forschungsprojekt-roboter-sollen-fluechtlingskindern-deutsch-beibringen-1.2793491.

[Mey] Stefan Kohlbrecher, Johannes Meyer. hector_slam package for ROS. `http://wiki.ros.org/hector_slam`. Accessed: 2016-05-6.

[MF03] Jean-Arcady Meyer and David Filliat. Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies. *Cognitive Systems Research*, 4(4):283–317, 2003.

[Mic] Microsoft. Kinect for Windows. `https://developer.microsoft.com/en-us/windows/kinect`. Accessed: 2016-05-6.

[MM96] Martin C Martin and Hans P Moravec. Robot evidence grids. Technical report, DTIC Document, 1996.

[Mor88] Hans P Moravec. Sensor fusion in certainty grids for mobile robots. *AI magazine*, 9(2):61, 1988.

[MT08] Nikos C Mitsou and Costas S Tzafestas. *An introduction to the problem of mapping in dynamic environments*. INTECH Open Access Publisher, 2008.

[per] Perf - Linux profiling tool. `https://perf.wiki.kernel.org/index.php/Main_Page`. Accessed: 2016-05-6.

[QCG$^+$09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[Rob] Mobile Robotics. Mobile Robotics Lab - Mapping. `http://cse17-iiith.virtual-labs.ac.in/mapping/index.php`. Accessed: 2016-05-6.

[SGV$^+$06] Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, Carlos Ferraz, and Judith Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, 10(1):37–44, 2006.

[SK08] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer Science & Business Media, 2008.

[Sta] RT Staff. Japan to Create More User-Friendly Elderly Care Robots. `http://www.roboticstrends.com/article/japan_to_create_more_user_friendly_elderly_care_robots/medical`. Accessed: 2016-05-6.

[Sta09] Cyrill Stachniss. Mapping and localization in non-static environments. In *Robotic Mapping and Exploration*, volume 55 of *Springer Tracts in Advanced Robotics*, pages 161–175. Springer Berlin Heidelberg, 2009.

[Thr00] Sebastian Thrun. Probabilistic algorithms in robotics. *Ai Magazine*, 21(4):93, 2000.

[Thr02] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[TK06] Kanji Tanaka and Eiji Kondo. Towards real-time global localization in dynamic unstructured environments. *JSME International Journal Series C*, 49(3):905–911, 2006.

[TN12] Xiaoyu Tong and Edith C-H Ngai. A ubiquitous publish/subscribe platform for wireless sensor networks with mobile mules. In *Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on*, pages 99–108. IEEE, 2012.

[WB99] Ian Q Whishaw and Brian L Brooks. Calibrating space: exploration is important for allothetic and idiothetic navigation. *Hippocampus*, 9(6):659–667, 1999.

[wir] Website of Wireshark. `https://www.wireshark.org/`. Accessed: 2016-05-6.

[WS04] Denis Wolf and Gaurav S Sukhatme. Online simultaneous localization and mapping in dynamic environments. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1301–1307. IEEE, 2004.

[WTT03] Chieh-Chih Wang, Charles Thorpe, and Sebastian Thrun. Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 842–849. IEEE, 2003.

I hereby declare that I wrote this thesis on my own and without the use of any other than the cited sources and tools and all explanations that I copied directly or in their sense are marked as such, as well as that the thesis has not yet been handed in neither in this nor in equal form at any other official commission.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als den angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Diese Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Magdeburg, den 9. Mai 2016 B. Sc. Dirk Steindorf