Saarland University
Faculty of Mathematics and Computer Science
Field Computer Science

Navigation Modeled on Human Pathfinding to Run a Mobile Baxter

Bachelor Thesis
Chair Prof. Dr. rer. nat. Dr. h.c. mult. Wolfgang Wahlster

Jessica Lackas

9. April 2018

Advisor:
Christian Felix Bürckert, M.Sc.

Examiners:
Prof. Dr. rer. nat. Dr. h.c. mult. Wolfgang Wahlster
Dr. Tim Schwartz

Eidesstattliche Erklärung

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

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, ...

(Datum / Date) (Unterschrift/ Signature)

Abstract

This thesis aims to improve human-robot collaboration by modeling the robot's navigation on human pathfinding. As humans are very experienced with other humans' walking patterns, this may lead to a more natural and intuitive cooperation. The human collaboration partner can feel more in control of the situation, because the robot's movements are more predictable, even for an untrained observer. To achieve this, one first has to collect data about human pathfinding and extract some key findings. These findings can then be applied to a navigation algorithm resulting in a practical approach to generate human walking paths for the robot to navigate on. A lot of work on human trajectories is very theoretical in nature and not easily, if at all, applicable to actual practical scenarios. The approach proposed in this thesis promises an elegant solution designed to be applied in real contexts including dynamic environments, instead of controlled, experimental contexts. In the process of implementing this algorithm, a software framework is being designed to integrate a self written navigation algorithm easily on the *Baxter Mobility Base*.

Contents

1 Introduction

The number of robots working in industrial scenarios is rapidly growing. The 2017 World Robotics Report, as published by the International Federation of Robotics (IFR)[1], estimates, that by 2020 more than 1.7 million new industrial robots will be installed worldwide. The tasks taken on by these robots are usually either repetitive and tedious or they include work that cannot be done by humans in an ergonomical way. For example, robots aid in lifting heavy assemblies in the automotive industry[2], similar to what is shown in figure 1 and they palletize boxes of chocolate[3].

Figure 1: Robot aids in lifting heavy assembly[4]

A scenario, which is not very common yet, is real collaboration between a human and a robot, but since robots are becoming more and more technically advanced, the possibilities for collaboration increase. This is in particular due to improved sensor systems and sensitivity of robots, that allow humans to safely work alongside robots, as those can avoid collisions and therefore prevent injuries[5]. What is more, human-robot interaction is no longer focussed on just industrial scenarios, but robots are also increasingly seen in domestic context, for example in the form of vacuuming robots[6]. The spectrum of robots that exist to date is already very broad, however not a lot of people regularly interact with robots and there is a number of reasons for that. In industrial contexts, the most important reason, as mentioned above, is the human co-worker's safety. Like any technology, a robot can fail to execute its task properly, possibly endangering the human, that is working with it. Even if the robot is working correctly, human error or mishaps could also lead to injuries. Therefore, proper precautions have to be taken to ensure the human's safety. A reason that is often being neglected, but especially important in social contexts, is the human's attitude towards robots and the possibility of working in such a collaborative team [29, 17]. While many people are

[1] https://ifr.org/free-downloads/
[2] https://www.automobil-produktion.de/technik-produktion/fahrzeugtechnik/schutzzaunloser-roboter-entlastet-audi-mitarbeiter-bei-montage-123.html
[3] http://www.palettierroboter.com/applikationen/schokolade/
[4] https://automationspraxis.industrie.de/news/mensch-roboter-kollaboration-im-tagungsfokus/
[5] https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/industrieroboter/lbr-iiwa
[6] http://www.allonrobots.com/household-robots.html

generally open to the idea, the difficulty is not only to guarantee that they are safe, but also to comply with any social rules to induce a feeling of safety. This is obviously necessary for humans to comfortably live with a robot in their home or to work in such an environment. In order to achive this, the human needs to feel in control of the situation at any given time, which is hardly the case if the robot's actions cannot be understood by the human, because they are unpredictable [11]. This holds true for any kind of action, however this thesis focusses on the robot's navigation, more accurately on the trajectories taken by the robot.

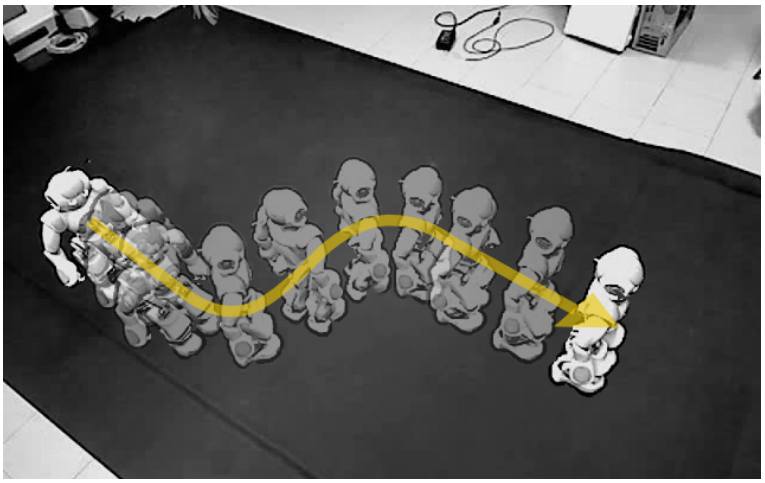


Figure 2: Sigmoidal walking trajectory of humanoid robot[7]

While in industrial scenarios it is common and mostly sufficient to have stationairy robots, many applications need mobile robots, as they are far more flexible. Mobile robots can aid in any form of logistics, they can switch between tasks at different locations and they can accompany a human almost anywhere. For all of these cases it is instrumental, if not necessary, for the robot to move automatically, meaning without any human assistance, as manually controlling it would be very unpractical and not collaborative. Therefrom results the need for autonomous mobile robots and suitable navigation algorithms. Simply letting the robot move on the shortest path is not only not that simple at all, because the path has to be planned and commands for the robot to follow this path have to be calculated, it can also result in unintuitive trajectories, one of which can be seen in figure 2. The s-curve the robot moves on seems unreasonable, because there is no obstacle the robot is avoiding. Unpredictable movements like this may surprise humans and make them feel uncomfortable. Additionally, a surprise moment like this may induce a human to have an inappropriate reaction with potentially dangerous consequences, such as stepping back and tripping over something. Thus it is important to make the robot's behavior predictable for a human.

[7] http://www.dis.uniroma1.it/labrob/research/HumanoidsTrajCtrl.html

1.1 Motivation

The movement pattern of the robot in figure 2 is most likely caused by the use of an ordinary A* algorithm [24] to plan the path on a grid that has been placed over the map of the environment. Two possible solutions of the A* algorithm in an abstract scenario can be seen in figure 3.

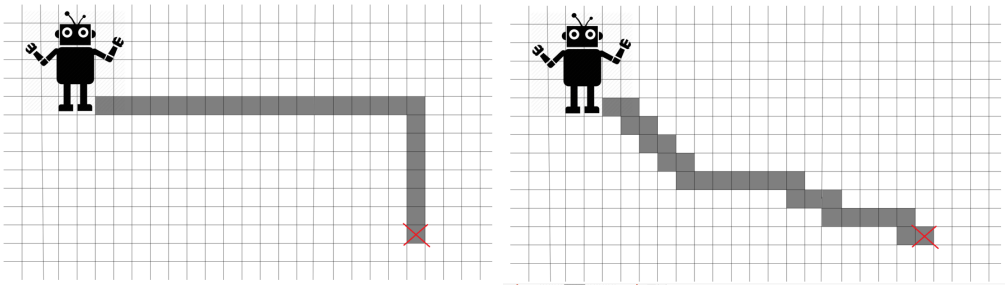


Figure 3: Possible solutions of the A* algorithm

As one can see, the use of the A* algorithm to plan a mobile robot's paths can result in unintuitive paths with sharp turns and zigzag motions. Particularly for a strong or heavy robot, such movements can make a human uneasy around the robot and unwilling to approach it. Moreover, it can be hard to predict the robot's destination, when it's moving on such a path, especially for an untrained observer. This can be discomforting and can also cause fear of collisions, because the human does not feel in control of the situation, when it is hard to predict the robots target point. A solution to this problem is to make the robot navigate in a more human way, as humans are very experienced with other human's walking patterns, which may lead to a more natural and intuitive cooperation in general. This approach also solves another issue with the A* algorithm. The shortest path to a target point around an obstacle often includes walking very close to the obstacle, which can be problematic in many cases. For example if the obstacle is a human, the robot may be entering the human's comfort zone, resulting in discomfort on the human's side or the obstacle can be a moving one, for which it would also be better to keep some distance to it to avoid collisions. These problems could be solved with a human-like navigation algorithm, as humans tend to walk around obstacles with some space between them and the obstacle, if this is possible. The replication of human walking is widely researched about in the field of humanoid robots [10, 27, 34] and the idea to replicate human walking paths is not new in itself, either. However, the work done so far on human trajectories is of a very theoretical nature and not easily, if at all, applicable to actual practical scenarios. The approach proposed in this thesis promises an elegant solution designed to be applied in real contexts including dynamic environments, instead of controlled, experimental contexts.

1.2 Robot

The subject of interest in this thesis is the ***Baxter Mobility Base*** (BMB), which can be seen in figure 4, once with the ***Baxter Robot*** mounted on top and once without it.

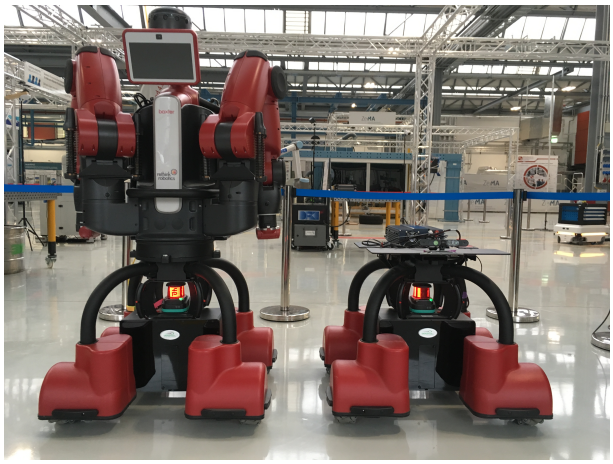


Figure 4: Baxter mobility base with and without Baxter robot

The ***Baxter Robot*** is available as an industrial or a research robot and in combination with its mobile base it has the potential to be an autonomous mobile robot. The BMB comes equipped with an on-board computer, that will run the navigation algorithm, an inertial measurement unit (IMU) that aids the robot's odometry and a 360 degree laser range finder, which can detect obstacles. The Mecanum wheel system enables the robot to move in any direction while independantly controlling its rotation and because of this, the wheels are often called omni-wheels. The capability to freely combine translation and rotation is quite useful for the navigation algorithm, as it enables more freedom, than an only forward moving and steering robot could provide.

1.3 Goals

The main goal of this thesis is the development of a human-like navigation algorithm to improve human-robot collaboration by making the robot's movements more predictable and more intuitive, which should lead to an overall more natural cooperation. This is realized, by ensuring the robot's trajectories resemble potential paths, humans would actually walk on.

In the process of implementing this algorithm, a software framework is being designed to integrate a self written navigation algorithm easily on the BMB. This framework includes the robot localization and a safety component, which prevents collisions with humans and other obstacles, as well as a skeletal structure for the actual navigation algorithm.

Not part of this thesis, on the other hand, are any other measures to estimate the robot's behaviour, like visual or auditive outputs. The next steps of the robot should

be predictable solely based on it's forward motion and the path taken. The maximum speed of the robot will be set to a moderate level to ensure this is possible, but there will not be a detailed accelation and deccelaration process to resemble human walking pace. While moving obstacles, such as humans, will be detected to prevent any collisions, their future movements will not be predicted. Therefore, humans are only avoided based on their current position.

2 Related Work

The replication of human walking is a long standing problem in robotics [10, 27, 34]. This also includes research on human walking trajectories [10]. This chapter assesses some of the previously pursued approaches and their shortfalls. Additionally, some approaches to autonomous navigation are being introduced and assessed. As the main goal of this thesis includes an effort to improve human-robot collaboration, this term is being defined for the scope of this thesis.

2.1 Human Walking Paths

The desire to replicate human walking paths is not new [10, 7]. However, most work done on this topic is either of a very theoretical nature and thus not easily deployable in actual scenarios or it requires a variety of sensors to extract human trajectories. This section introduces some important approaches and identifies useful findings.

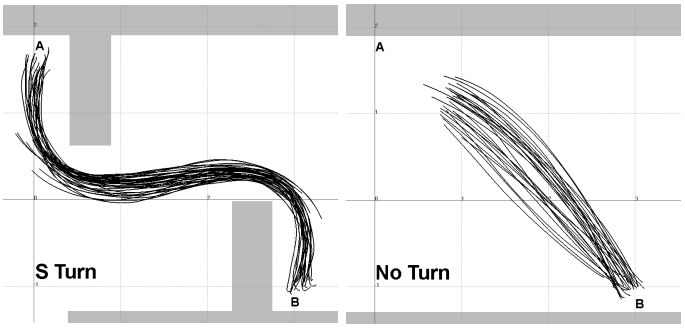
2.1.1 Generation of Human Walking Paths

A commonly used approach to generate biological movements such as human walking is the optimal control theory [10]. It formalizes the problem of trying to minimize a cost function in consideration of side conditions encoded in differential equations. In [5] and [28] the authors found evidence, that the human path planning process equates to an optimal control problem with an unknown cost function. In their work 'Generation of human walking paths' [30], the authors tried to find this cost function with the inverse optimal control problem, meaning that they used data of actual human walking paths and different models for the side conditions, in order to find the function, that may have been minimized. The goal of all these papers is to generate the exact geometric shape of human walking paths for a given start and end position and orientation. As they focused on these geometric details, they did not include the movement around obstacles, let alone dynamic ones.

2.1.2 Realistic Human Walking Paths

In the paper 'Realistic Human Walking Paths' [7], the authors' goal was to develop a realistic model of pedestrian navigation, meaning a model not only for suitable human walking paths, but for realistic paths humans would actually take. Although they developed this model for entertainment applications and different kinds of simulations, some of their finding are useful for this thesis. In order to gain insight on pedestrian navigation, an experiment was conducted, in which participants were observed in controlled conditions and salient features of pedestrian walking were identified. Their findings were then validated through comparison with pedestrians in a natural, non controlled setting. In the experiment, participants had to carry a piece of paper from one place to another and a video camera recorded it. The path taken by the participant was then digitalized by projecting the human's center of mass to the ground. The experiment was varied through different obstacle situations, including the need for an s-turn and no turn to get

to the target point. The images below (figure 5) show the observed trajectories in these two experimental conditions.



(a) S-turn condition (b) No turn condition

Figure 5: Observed trajectories under different turn conditions [7]

In the s-turn condition people kept a certain distance to the obstacles and walked on a rounded curve with no corners and in the no turn condition people did not always follow the straightest and thus shortest path, although it would have been possible.

2.1.3 Walking Characteristics Extraction and Behavior Patterns Estimation by Using Similarity with Human Motion Map

The authors' of 'Walking Characteristics Extraction and Behavior Patterns Estimation by Using Similarity with Human Motion Map' [33] goal was not to use the extracted walking characteristics to make a robot move in a human way, but to use this data to predict human motion, so a robot can move autonomously and prevent collisions with humans. However, their findings can be applied to get a more human robot navigation, as well. In order to collect data on human walking behavior, the authors conducted an experiment in which they let a robot observe humans in different environments, including a corridor, a corner and an elevator hall. They collected multiple independent data sets in the same areas, because human motion is highly dynamic and uncertain. They then used the human position displacement as walking characteristics in each environment of the observation experiment. The visualization of the results can be seen in figure 6.

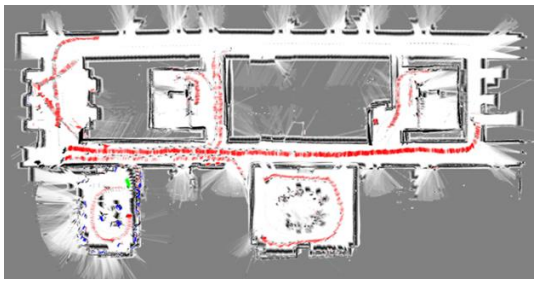
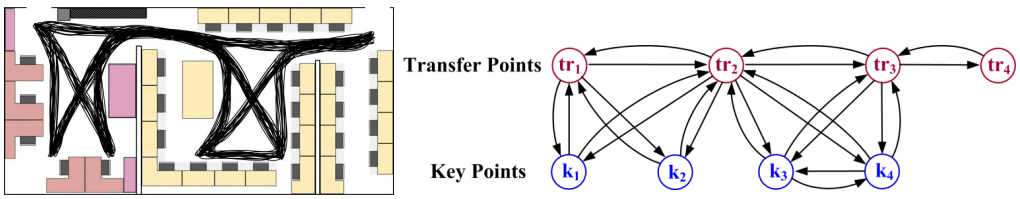


Figure 6: Observed human walking trajectories [33]

An obvious observation is, that human walking behavior depends on the environment. In the corridor, humans walked in a straight line in the middle of the corridor and in the corners people walked in a curved line. Near the elevator people were mostly just waiting or pacing, which is not important for this thesis, as waiting does not need to be implemented, because the robot should simply not move when it does not need to. With their approach, the authors were able to predict human motion only in the before observed areas or very similar ones, such as the same corridor on a different floor of the building, which seems quite unpractical, because the collection of data is non optional in order to use this system.

2.1.4 Mobile Robot Navigation Based on Human Walking Trajectory in Intelligent Space

A different approach was taken by the authors of 'Mobile Robot Navigation Based on Human Walking Trajectory in Intelligent Space' [13]. Their motivation were robots that are supposed to provide services in human populated environments and they thought the best navigation style for this was just like a human's. In order to achieve this, multiple vision sensors needed to be installed in the environment and the cameras needed to be calibrated to extract human walking trajectories. All trajectories are a sequence of data points and similar trajectories are grouped together and the averaged trajectories are smoothed based on method of least square [16]. From these trajectories key points are extracted, meaning points where humans stop often and transfer points, where humans enter or exit an environment. The stop points are found as points where a human's walking speed is below a threshold and then grouped together and the center is used as a node. The transfer points have to be set manually. From these points a topoligical map is created. An example of this can be seen in figures 7(a) and (b) below.



(a) Obtained human walking trajectories

(b) Topological map

Figure 7: Human walking trajectories and accordingly generated topological map [13]

The topological map represents the environment, with the key points as nodes and the extracted trajectories as edges, in both directions respectively. This map is then used in robot navigation by finding a global path with Dijkstra's algorithm based on the topological map and local person avoidance, while navigating on the trajectories. The paths taken by the robot should then reflect human activity patterns in daily life. In order to navigate on these trajectories the robot's global position needs to be known. This is done with a dead reckoning system and landmarks distributed on the ceiling and

an infrared ray, which is unpractical, as the environment needs to be prepared before the robot can move. Two further inconveniences are, the robot only being able to move to one of the key points and not to an arbitrary one. This is less of a problem, if the robot only needs to execute tasks at these points, but it is certainly unflexible. It is also not very human to move back to the predefined trajectory after avoiding a person, as humans constantly 'recalculate' their paths and adjust after leaving the initial trajectory.

2.2 Robot Navigation

In order to design a software framework to integrate navigation algorithms on a robot it is worth looking into similar systems first. As there are countless approaches to autonomous robot navigation, this section can only cover a small part of this field and is restricted to one very well known framework, which provides autonomous navigation software and one less known framework, which is being developped by DFKI GmbH.

2.2.1 ROS

The arguably most well known robotics software platform is called ROS, which is an acronym for robot operating system. As the BMB is ROS-compatible, this system is not only important as a related work, but also as a contributive component, because it will be used for the basic control of the robot. For more details on this refer to chapter 4. The official description of ROS is as follows:

"ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers." [3]

The following explainations are based on [2]. ROS as a framework is language-independent. At the present time, there are three main libraries, making it possible to program ROS in Python, Lisp or C++. There is also an experimental Java library. The basic concepts of ROS can better be understood when distinguishing between static concepts and concepts that are used by the running system. Statically all ROS resources are organised in a hierarchical structure of packages and stacks, called ROS File System. A *package*, as the most fundamental unit, is a directory, which contains resources, such as external libraries, data, configuration files and nodes, which are explained below. A collection of packages is called a *stack*. It usually offers a bundle of functionalities, such as navigation or localization.

The dynamic concepts of ROS are combined in the ROS Computation Graph. Any ROS program consists of different processes called *nodes*. A node is an instance of an executable and it may equate to anything from a sensor to a controlling algorithm. All running nodes are managed by a *master*, so they can find each other and exchange data. This exchange can be done in two different ways, asynchronously via a topic and synchronously via a service. A *topic* transports data based on a publish/subscribe system.

One or more nodes can publish and subscribe to the same topic, so it is essentially a many-to-many bus. The data published on a topic is structured as a typed message. A *message* is a data structure that can consist of primitve types such as strings, booleans and integers and of other messages recursively. If there is a need for synchronous communication between two nodes a *service* can be used. The functionality of a service is similar to that of a remote procedure call.

One of many functionalities provided by ROS is a navigation stack. It includes algorithms for all levels of the navigation process, meaning mapping, localization, path planning and obstacle avoidance. For this navigation stack to be applicable the robot needs to fulfill some requirements. Obviously the robot needs to be running ROS and publish sensor data using the correct ROS Message types. There needs to be a planner laser mounted on the robot, as this is used in the mapping and localization process. The robot's motion needs to be controlled by sending velocity commands in the form of a two dimensional translational vector and a one dimensional rotational vector. The robot should also have a square or circular shape, because the navigation stack was developed on such a robot and there are some problems generalizing the algorithms. For example large rectangular robots do not fit through doors, although they would simply need to be turned[8].

Since the BMB fullfills all of these requirements it is possible to use only ROS for autonomous navigation, however since ROS does not include a human-like navigation stack all of the problems mentioned in the introduction, such as moving close to obstacles, moving on zigzag paths and being unpredictable, are present. It would also be possible to integrate a self written ROS navigation package into the navigation stack, but for this thesis, it was decided to write the algorithm in Java instead.

2.2.2 ROCK

Another robotics framework is DFKI GmbH's ROCK (robot construction kit). It is the primary software framework running on most of DFKI GmbH location Bremen's own robot developments and it is designed to allow easy reuse of software components from different robotics projects. The general idea behind ROCK is very similar to that of ROS, but it is not nearly as wide spread, therefore there are not as many features, that are already implemented. The basis of ROCK's component model is formed by Orcos Real Time Toolkit.

Contrary to ROS, ROCK does not provide openly available, generally applicable autonomous navigation functionality, at the present time.

2.3 Human-Robot Collaboration

The term human-robot collaboration (HRC) means a scenario, in which a human and a robot work hand in hand, without separation and without safety fencing. So, rather than replacing, the robot assists the human with its skills, which are usually precision, strength and endurance. There being no seperation between automated and manual workstations

[8] http://wiki.ros.org/navigation

leads to all of the advantages of automation, such as efficiency, being present, without losing the flexibility and decision-making abilities of humans. More information on this topic can be found in the ISO/TS 15066 guideline[9]. HRC is considered to be one of the key factors in Industry 4.0[10].

2.4 Robot Localization

There are a number of ways to localize a robot, such as Simultanious Mapping and Localization (SLAM) [14] and landmarks [6], that will not be part of this thesis. The mobile robot localization method utilized in this thesis includes an extension of the Monte Carlo Localization (MCL) [12], called Adaptive Monte Carlo Localization (AMCL) [23]. This section explains both MCL and the extension to AMCL, based on 'Robot Localization in Dynamic Environments' [32].

2.4.1 Monte Carlo Localization

MCL uses a particle filter algorithm to localize a mobile robot on a given map. The particle filter estimates the pose of the robot, by using a weighted set S_t of n particles that represent the system's belief about the current state at time t:

$$S_t = \left\{ \left(s_t^{(i)}, w_t^{(i)} \right) | i = 1, ..., n \right\}$$

The state of the ith particle at time t is denoted by $s_t^{(i)}$ and its assigned importance weight is denoted by $w_t^{(i)}$. The sum of all weights, at any given time, is equal to one and they represent the probability for the particle's state to be the true state of the system.

In every iteration, the algorithm receives a map of the environment at time t, a motion control measurement, an observation of the surrounding area along with the sample set S_{t-1}, which represents the system's belief about the previous time step. The motion control measurement is usually taken from an internal measurement of the robot, for example the robot's odometry, calculated with data from motion sensors. The external observation can be taken from external sensors, such as a laser range finder. The previous set of particles and the control update is used to predict the new state of the particles. After this step, new importance weights are assigned to each particle according to the external observation. The weight depends on how well the particle's new state on the map fits the external observation. If a particle was in fact near the robot's true pose, its importance weight would be very high. The last step is then the sampling from this updated set, which is done according to the importance weights. This means that the probability of a particle to be sampled is proportional to its importance weight and particles that are more likely to represent the robot's true pose are more likely to be sampled. In reality, a particle with a very high importance rate can be sampled more than once, while a particle with a very low importance weight is likely to not be sampled, at all. An example of this process, can be seen in figure 8.

[9] https://www.beuth.de/de/vornorm/iso-ts-15066/250504228
[10] https://de.wikipedia.org/wiki/Industrie_4.0

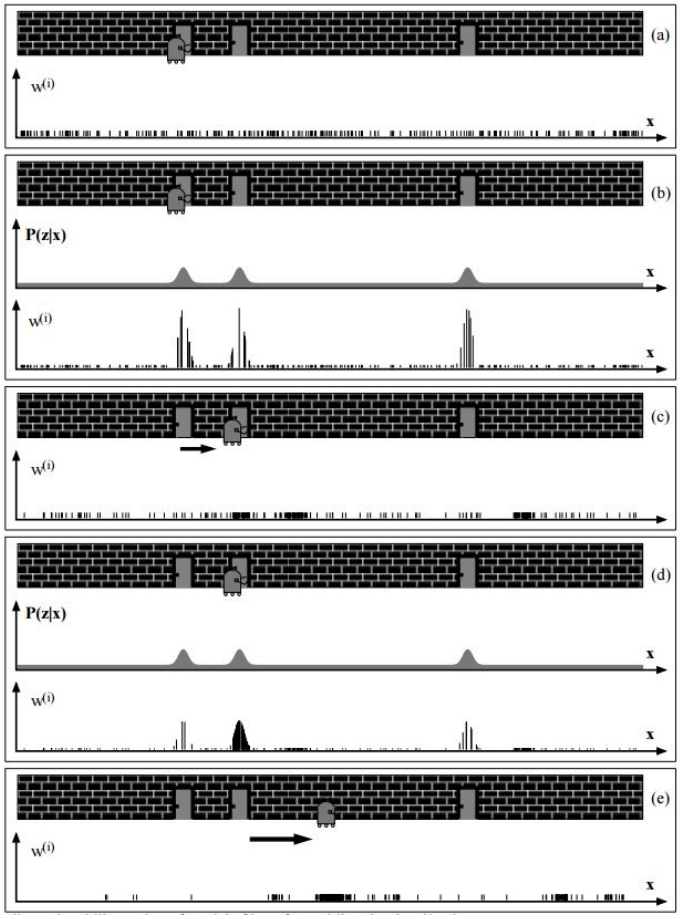


Figure 8: Example of MCL [32]

There is a robot in a one dimensional corridor with three doors. The robot has a sensor with which it can determine, wheter it is standing in front of a door and it counts its steps, in order to know its relative position. For this example, the same map is used in every step, but it would be just as possible to add or remove doors. In figure 8(a), the particles are uniformly distributed, because there is no knowledge about the robot's position in the corridor. In the next step it has received data from its door sensor, so the importance weights of the particles are being adjusted according to the information, that the robot is standing in front of a door. As there has been no resampling of the particles yet, the distribution is still the same. In figure 8(c), the sampling did already take place and the robot has moved forward, so the particles are being shifted according to the motion estimation by the robot's step counter. After this, there is a new measurement from the door sensor, and the importance weights are adjusted just like before. As one can see, only one of the clusters of particles gets high importance values, namely the one that best represents the robot's actual position. From this point onwards, as visible in figure 8(e), the particle filter accurately estimates the robot's true location.

2.4.2 Adaptive Monte Carlo Localization

AMCL aims to improve MCL, by adjusting the number of particles used in the localization process. When the robot is unsure of its position, it uses more particles, but when the filter converges, a small number of particles are sufficient to track the robot's location.

The algorithm uses the Kullback-Leibler divergence (KLD) [26], in order to adjust the particle set size. KLD measures the difference between probabilistic distributions and using it allows to adjust the number of particles according to the quality of the true distribution's approximation. The algorithm's inputs are the same as in the MCL version and additionally there are two statistical error bounds ϵ and δ. The error bounds are used to determine the particle set size, by ensuring that the error between the sample based approximation and the true distribution is less than ϵ with a probability of 1 - δ. The resampling process is different to that in the MCL. In each step of the algorithm, particles are generated until the requirement mentioned above is fulfilled. To determine whether this is the case, the algorithm uses a histogram over the entire state space with all bins being initialized as empty in each iteration. Then, exactly as in MCL, particles are sampled from the previous set according to their importance weights and their states are predicted using the control update. Also as in MCL, the weights are then updated according to how well the measurements fit the map at the particles' states. After this, each particle is placed in a bin on the histogram and whenever a particle is placed in an empty bin a counter c is incremented. This counter c and the statistical bounds ϵ and δ mentioned before, determine the number of particles needed for an efficient representation of the state space. If c is high, it means that the sampled particles are spread around the state space and the filter has not converged, so a lot of different bins have been filled. Therefore the number of particles has to be high. If the particles are focused in small clusters, a lot of them will fall into the same bins and c will be low, so the number of particles needed is also low.

2.5 Summary

This section summarizes the useful findings of the related work mentioned above and compares it to the approach, that is proposed in this thesis.

While the optimal control theory provides very detailed information on the geometric shape of human walking trajectories, it is not designed to be used to calculate paths for a robot to move on. The same holds true for the pedestrian model, however some key characteristics, such as rounded curves and distance to obstacles can be adopted. The human motion prediction can be used to estimate human walking paths, but as humans are highly complex, it is very hard to develop a system, which is generic enough to be used universally. A robot navigation system based on the concept of intelligent space works well, but is highly impractical and inflexible. It would be possible to implement a human-like navigation algorithm purely on the basis of ROS, but this requires knowledge about ROS specific details, both for the deployment and maintanance of the system.

An overview of the capabilities and limitations of the methods developped in the

related papers in comparison to the proposed method can be seen in table 1.

	Suitable for human-robot collaboration	Works in dynamic environment	Needs observation data	Obstacle avoidance	Learning system
Optimal control approaches 2.1.1	No, because it is not actually deployable	No	Yes	No	Not an actual system
Pedestrian navigation model 2.1.2	No, because it is build for simulation purposes	No	Yes	Yes	No
Walking characteristics extraction 2.1.3	No, because it is only designed to estimate human motion	No	Yes	No	No
Human walking trajectory in intelligent space 2.1.4	Yes	No	Yes	Yes	No
ROS navigation stack 2.2.1	Not specifically designed for this purpose, but possible	Yes	No	Yes	No
Proposed approach	Yes	Yes	No	Yes	Yes

Table 1: Comparison of related work

3 Concept

This chapter concretizes the methods of resolution for the problem of human-like robot navigation. Some key aspects, which make a path appear human-like, are described, possible transfers to robot navigation are explained and implications for the BMB navigation framework (BMBNF) are inferred.

3.1 Human Walking Paths

One of the main challenges of making a robot navigate in a human way, is that there is no accurate definition of what human trajectories or even just human-like paths are. However some aspects found in traditional robot navigation can clearly be classified as non-human, such as sharp turns and zigzag motions, so there is a general understanding about non-human walking characteristics. This is because we, as humans, are very familiar with the way humas walk, because we observe it every day. Combining these characteristics with the observation of people in everyday life and the findings explained in the related work chapter, the following key aspects of human path finding have been deduced:

- Humans prefer to walk through familiar areas, so they will probably take paths they have already taken many times, rather than possible shorter ones. This is because it would require more attention to walk through an unknown area and still get to one's destination. The indicated can be assumed, as driving through unfamiliar areas has a similar effect [8].
- Resulting from the former point, the length of a path has only a secondary role, but it is still considered for efficency reasons.
- Wider areas are generally preferred over areas too narrow to comfortably walk through. This is because it would require additional attention to not knock any body parts on any obstacles.
- Similarly, humans tend to keep some distance to obstacles, if possible. This can, for instance, be seen in the s-turn condition in the experiment from section 2.1.2, where people kept a distance to the obstacles at all times.
- When walking through a narrow segment, humans tend to walk right in the middle of the obstacles. For example, in the experiment from section 2.1.3 people walked right in the middle of the corridor.
- Paths taken by humans are rounded with smooth curves and there are generally no sharp turns or zigzag, because it would require additional effort.
- Humans naturally walk forwards and neither sideways nor backwards, unless absolutely necessary, simply because this is what we are biologically built to do.

Of course these characteristics are not quantifiable, but since there are substantial differences between individuals anyways, following these rules should result in paths, that resemble actual human ones.

3.2 Human-Like Navigation

As robot navigation is a multilayered problem, including the handling of sensor information, the global localization of the robot, path planning and path execution, the algorithm can be compartmentalized. For clarity's sake, the individual steps of the algorithm, as depicted in the diagram below (figure 9), will be looked at individually and the realization of each of the human-like path characteristics from above is explained in the relevant step. While this thesis focusses on the BMB, all of these concepts are general and thus also applicable to other robots. However, there may have to be adjustments in the actual implementation, for example for robots without omni-wheels, because the translational and rotational velocity cannot necessarily be set independantly, like is the case with the BMB. Therefore, the path execution algorithm would need to be modified.

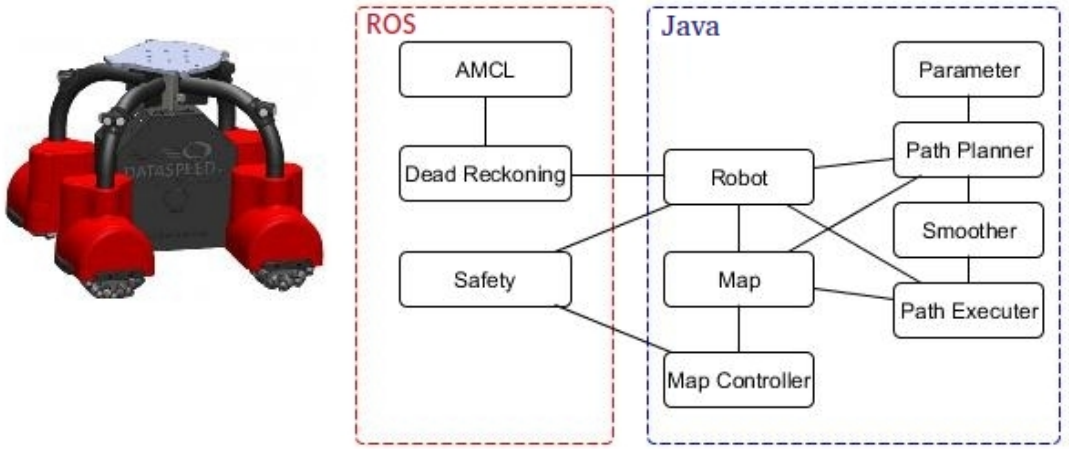


Figure 9: Steps of the navigation algorithm

Step 1. Map

In order to use graph search algorithms in the path planning process, one needs to discretize the continuous environment the robot is supossed to navigate in. One popular approach to this problem is cell decomposition, where the two dimensional space is decomposed into cells of some kind, often circles or convex polygons [4]. The only regular polygons, that can be used to tessellate continuous 2D environments are triangles, squares and hexagons [4] with square grids arguably being the most popular ones. Using a square grid, however, makes for the unnatural movements usually seen with the A* algorithm, as it leads to straight lines and sharp turns, as depicted in figure 10 on the

left side, rather than smooth curves or to zigzag motions, as depicted in figure 10 on the right side.

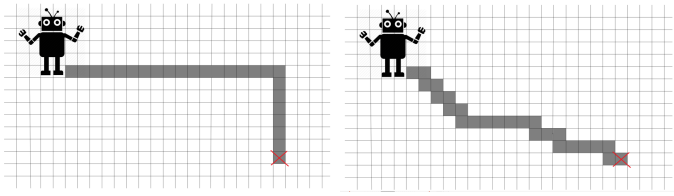


Figure 10: Possible solutions of the A* algorithm

An elegant way to deal with this issue is to use a hexagon grid instead, as this allows for steps in six different directions including four diagonal ones, as can be seen in figure 11 below. Since hexagons align with each other the problem of distance differences, that would occur with diagonal steps on a square grid does not occur.

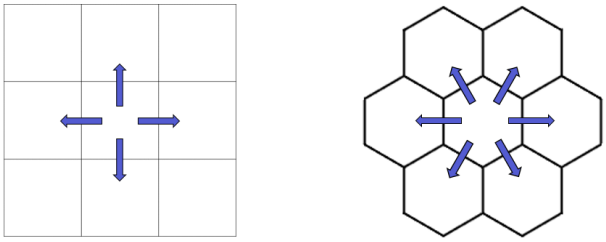


Figure 11: Neighbours on square grid and hexagon grid

While it would be possible to use squares small enough, that the difference in length of a diagonal and non-diagonal step are neglectable, this would result in a substantially smaller grid and thus a bigger search graph, which would make the pathfinding unnecessarily time-consuming. The size chosen for the hexagons should depend on the robot's measurements, because if they are too small, the pathfinding can be too slow and if they are too large the position of obstacles is inaccurate and paths may appear blocked, although they are not. An example of this can be seen in figure 12, where a 20 cm hexagon grid has been laid over a 1m wide cordidor. While in reality, a 80 cm wide robot would fit through this coridor, this does not seem to be the case on the hexagon map. Therefore, the size of the hexagons will be part of the adjustable parameter set.

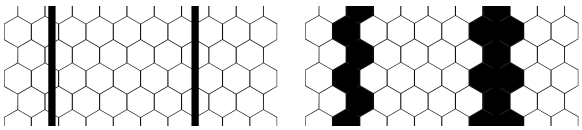


Figure 12: Coridor on a hexagon grid

One way to handle this map, is to connect it to the TEDURU server [15], which is a

dual reality framework for HRC scenarios. It can be used to manage the hexagon grid by providing information, such as the size and orientation of the hexagons. TEDURU can then provide collision information about every hexagon that can be used to update the map with information on obstacles. The detection of obstacles, however, is still a challenge, as the sensors of most robots cannot cover the entire surrounding of the robot. For example a table is seen as four small obstacles by the BMB, as, because of the LRF's height, only the table's legs are detected, but not the tabletop. This is not as problematic if the robot is small enough to fit under the table, but it could make a human co-worker uncomfortable if the BMB emerges from underneath a table and if the Baxter robot is mounted on top of the BMB, it would result in a collision. Therefore it would be helpful to introduce some sort of external obstacle detection, for example in the form of cameras that cover the entire space. If this is successful the map should consist of the free moving space to plan on, as well as all the obstacles, possibly with information such as their height. In figure 13 below, the P4PLab (Power4Production) can be seen, as an example of such a map. This data can be used to either plan around the obstacles or if the only possible way is to move under a table, depending on the robot, this path could also be considered, although it is certainly not the most human way to move under a table. Another point to note is, that even small obstacles, that could easily be stepped over by a human, have to be registered, as a robot can be more limited in its movements.

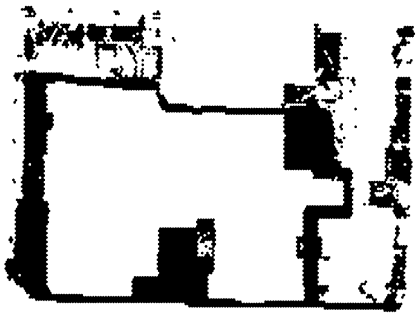


Figure 13: Hexagon map of the P4PLab

Step 2. Map Controller

In pursuance of human-like trajectories, the map should be evaluated to allow the graph search algorithm to find suitable solutions. The first key aspect of human pathfinding that can be taken into account, is the keeping of a certain distance to obstacles, if possible, for example when walking around a table. Not only is this human, it is also important for safety reasons, because surrounding humans are also detected as obstacles and the robot should not get too close to them. To achieve this, the obstacles on the map are processed and their outer edges are diffused, resulting in a fluent transition from the free moving space to the obstacle. A graphical representation of the result of diffusing two small obstacles can be seen in figure 14. Part of the transition space can

still be entered by the robot, if all other passage ways are blocked, or the path around the obstacle is unreasonably long.

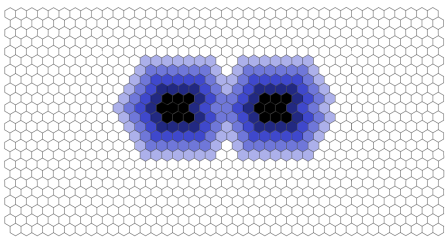


Figure 14: Two obstacles with diffused edges

This diffusion process can be realized by assigning double values v_t between 0 and 1 to every map tile t, with a 1 standing for 'blocked' and a 0 for 'free' and then transferring part of that value to the neighbor tiles $n_i(t)$ (with i = 1,..., 6), that are not part of the obstacle. The obstacle value can be calculated as follows:

$$v(t) = max\left\{\sum_{i=1}^{6}\frac{1}{6}\cdot v(n_i)\cdot flow\_value,\ threshold\right\}$$

It is then added to the cost of entering said map tile, meaning the transfer takes place, by taking a certain percentage, alias the $flow\_value$, from the average value of all neighbor tiles and assigning the result to the tile in question, as long as it exceeds a certain threshold. This $flow\_value$ is part of the adjustable parameter set, as it controlls how close the robot can move to obstacles and the threshold is necessary, because an obstacle should only affect nearby areas and not the entire map. In order to get smooth values, the transferring is done in an iterative manner until no value changes any longer. This also leads to a fluent change of the values when an obstacle is moved, because they are not simply set to 0 when the distance to the obstacles increases, instead they decrease in each iteration step. Furthermore, this results in the best values being right in the middle between two obstacles that are close, but not too close to each other, so the robot can take a path right through the middle, what is also a key aspect of human pathfinding. The images below (figure 15) show the map from the last step with the diffusion of the obstacle values after one, five, ten and fifty full iterations represented in blue.

Figure 15: Map iteration process after one, five, ten and fifty iterations

Similarly to the obstacle values, another factor that can be taken into account is the familarity of an area. As humans prefer to take paths they already know, every hexagon

the robot already passed can get a small increase in its familarity value which will be subtracted from its cost. This value can then be transfered to the neighbor tiles just like in the blurring of obstacles and the robot will thus prefer familiar paths and areas, just like a human. As an outlook to the evaluation, figure 16 shows the familiarity values after a simple transport task represented in green.

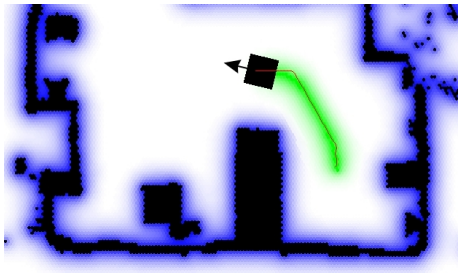


Figure 16: Familiar area marked in green

Step 3. Path Planning

The blurred edges and familiar areas together with the free space can then be used for the actual path planning, that works similar to the A* algorithm, but takes the calculated costs of each tile into account. Figure 17 shows a possible solution for a path around the two obstacles from above if there are no familiar areas yet. The algorithm chose to plan around the obstacles instead of through the middle of them, because the space in between is too narrow to walk through comfortably.

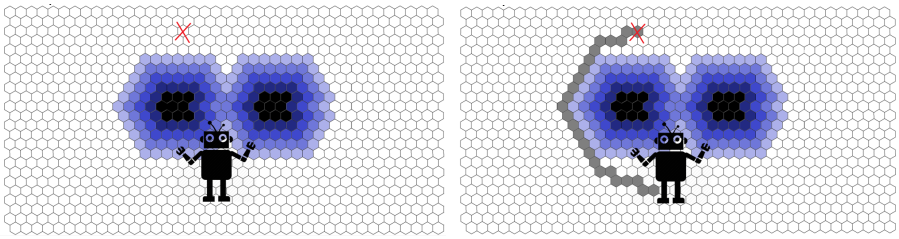


Figure 17: Planned path around obstacles

A challenge in the path planning process is the dynamic nature of the environment, because the map has to be updated constantly, possibly even during the path planning, as an obstacle could be moved to block the calculated path. This can be quite problematic, as it could lead to the robot never actually moving, because the best path is constantly changing. The human way to deal with this problem is to start walking the initially planned path and avoid obstacles locally, as humans do not have a global overview. Some details about the local obstacle avoidance will be given below in the safety component section, as this also avoids any collisions. In addition to solving the dynamic environment problem this also helps to make the robot more predictable. If a human co-worker is used

to the robot always moving in the same direction when executing a task, it would be very confusing if the robot started moving in a different direction because of an obstacle far away the human does not know about. This could lead to the human believing the robot is faulty or executing the wrong task. Another aspect worth taking into account are the areas, where the robot has to replan its path because of moving obstacles. If there is an area where obstacles are constantly changing and the robot needs to replan every time it enters this area it might be better to avoid it altogether, if possible.

Step 4. Smoothing

The smoothing of the planned path is the first step in converting the discrete path result from the path planner into commands that can be executed by a real robot, which moves in a continuous environment. Since another key aspect of human locomotion is the walking of rounded curves, rather than sharp turns, as well as straight lines, rather than zigzag, it would not be reasonable to simply connect the hexagons' midpoints to a continuous curve. On the planned path in figure 17, there are still some zigzag parts, caused by the usage of a grid, as well as a sharp turn at the corner of the blurred out edge of the obstacle. There are two main kinds of zigzag patterns possible on a hexagon grid. The first one happens due to the quirk of the hexagon map that it is not possible to go straight up or straight down and the second possibility occurs, if there is a diagonal part of the path that does not align with the natural angle of the hexagons. Those zigzag parts can be smoothed by shifting the reference points for the continuous movement plan. In the graphics below three possible segments of the result of the path planner can be seen. In figure 18(a) the midpoints of the hexagons are marked and in figure 18(b) their midpoints are marked. This shift results in the reference points being on the edges of the hexagons. In the third image, once again the latter points' midpoints are marked and in the forth image the continuous curve that results from connecting these new reference points is shown. This way of shifting the reference points removes the zigzag pattern and converts it into a straight line.

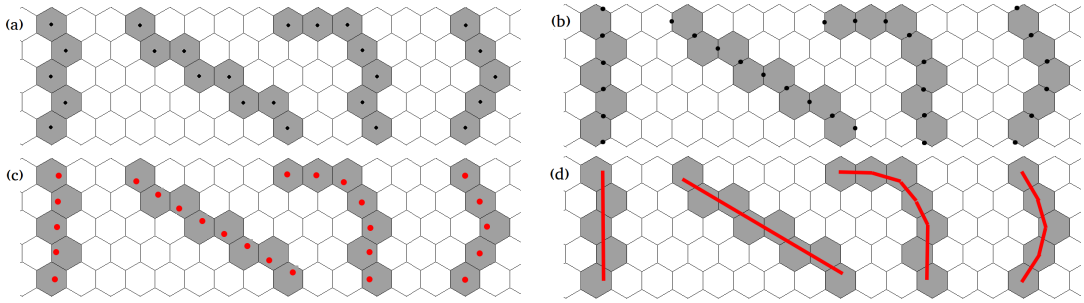


Figure 18: Smoothing process via midpoints

What can also be seen in the images above is the smoothing of any sharp turn, because there are also two kinds of turns, either switching from one diagonal to another diagonal or switching from a horizontal line to a diagonal one or the other way around. Both of these turns can be replaced by a round curve that is approximated with the new

reference points. An advantage of this very simple smoothing approach is, that the curve resulting from connection the reference points does not leave the hexagons. Therefore, the smoothed path does not collide with any obstacles. With traditional smoothing approaches, such as spline interpolation[11] or Bézier curves[12] the path can be altered quite heavily and there needs to be an assessment, whether the new path violates any constraints, such as getting too close to obstacles or even colliding with any. What is more, using such a method in a reasonable way could result in losing the advantages from the hexagon grid, because the diagonal steps could be smoothed out.

Step 5. Path Execution

The state of the robot in the navigation process can be modeled with a state machine, as is shown in figure 19 below.

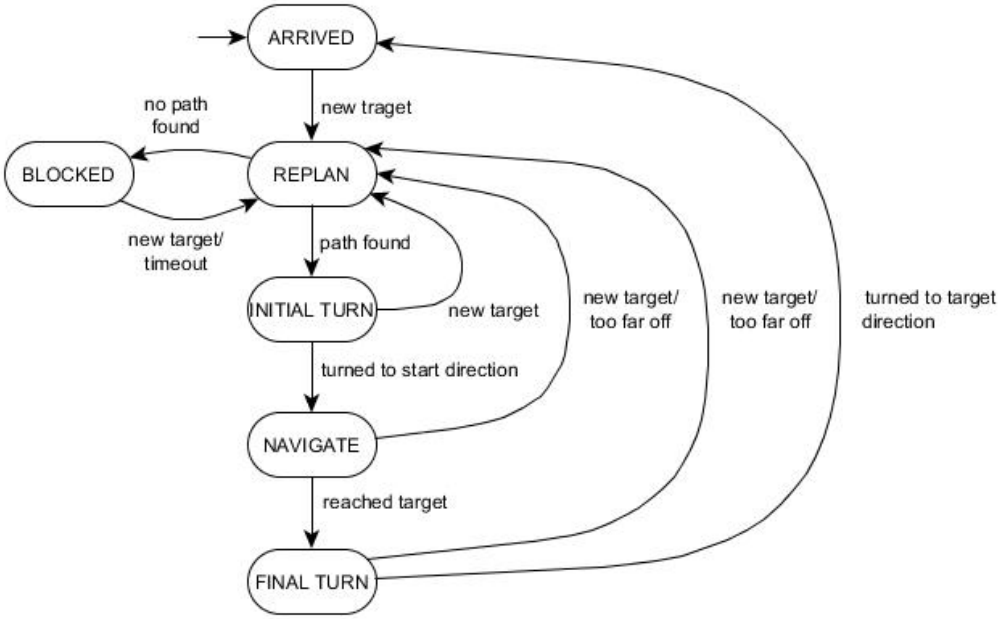


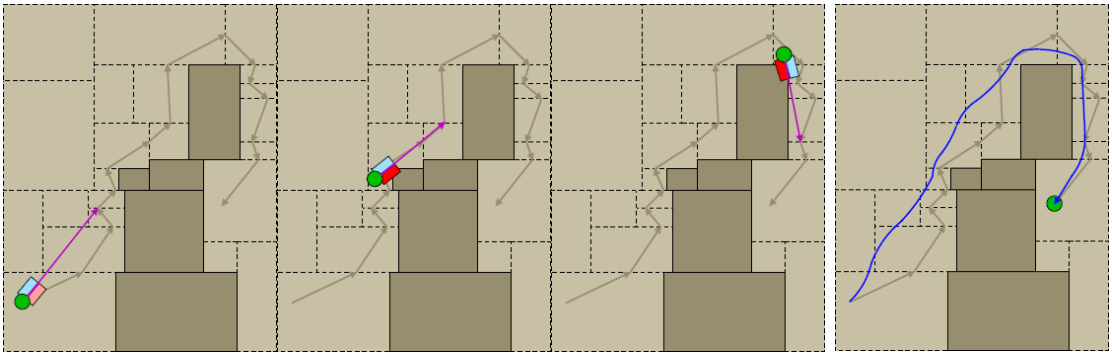
Figure 19: State machine modelling the navigation process

The idle state is called ARRIVED, because whenever the robot has reached its destination it has to wait until a new target is set and the new path can be planned, hence the REPLAN state. For the purpose of making the robot move forwards, the robot first has to be turned in the direction the path starts in, which is done in the INITIAL TURN state. Similarly the robot can be required to turn in a specific direction at its target, thus the FINAL TURN state. The key state is the NAVIGATE state, as this is where the robots velocity is calculated. This process will be explained in more detail below.

[11] https://en.wikipedia.org/wiki/Spline_interpolation
[12] https://en.wikipedia.org/wiki/Bezier_curve

From all of these states it can be necessary to return to the REPLAN state if the robot went too far off the planned path, if there is a new or moved obstacle near the robot or if a new target is set before the old one has been reached. If a new path can be found after this, the sequence will be repeated, however if no path can be found the BLOCKED state is entered and after a timeout a new attempt will be made.

Since the result of the smoother in the previous step is still a discrete path, these reference points are being used to calculate commands that can be executed by the robot and result in a fluid motion in the continuous environment. Simply sending the robot from one reference point to another would result in jerky motions, because the path would be a series of line segments, and not a rounded curve. The approach to solving this problem is based on the reactive path following algorithm described in the AI systems of the video game Left 4 Dead[13]. The developers of this game utilize a navigation mesh and calculate paths by connecting the transition points between different sectors, as can be seen in the images below. Just like on a regular grid, this results in a rough path with zigzag patterns and following this path would result in unrealistic walking behaviour. In the game, this is avoided, by letting the AI walk towards a point further ahead on the path, as is shown in figure 20(a). This look-ahead point is changed, when the AI is getting within a certain distance of this point. The resulting walking trajectory looks a lot more fluid, as can be seen in figure 20(b). A disadvantage of this approach is, that it can cause the AI to move too far away from the calculated path, possibly into areas that were not part of the initial path, because they should be avoided.



(a) Walking direction of AI (b) Walking trajectory

Figure 20: Walking trajectory of AI in video game left4dead[14]

A solution to this problem is to add a slight motion towards the initially planned path, while still keeping the look-ahead. Since the reference points for the robot's path have already been smoothed in the last step, this approach is especially expediant, because the zigzag has already been removed and following the reference points a little more closely than the game AI does is desirable. A graphical representation of the calculation of the robot's velocity can be seen in figure 21 below. One component is the vector from

[13] http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf
[14] http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf

the robot's current position to the closest reference point of the path and the second component is the vector towards the look-ahead reference point. Exactly how far ahead the look-ahead should be depends on the hexagon size in compairison to the robot's size and is to be evaluated. If the hexagons are relatively small compared to the robot it should be further ahead in order to actually be a look-ahead or the robot's body might cover the reference point already. On the other hand, if the hexagons are relatively large in comparison to the robot the look-ahead should be less advanced, or the path would be altered too significantly, because the robot would cut the path's curves.

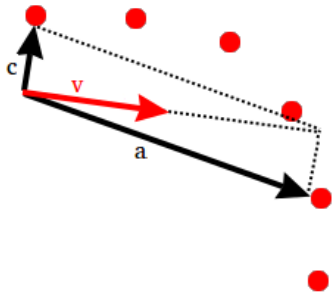


Figure 21: Velocity calculated from closest point and lookahead

The velocity $\vec{v}$ can be calculated by normalizing the sum of the before mentioned components and scaling it with the desired speed s the robot should move with. The $\vec{c}$ stands for the vector from the robot's position to the closest points of the path and $\vec{a}$ for the vector to the look-ahead point.

$$\vec{v} = speed \cdot \frac{1}{||\vec{c}|| + ||\vec{a}||} \cdot (\vec{c} + \vec{a})$$

In addition to the robot's velocity, its orientation has to be set, as because of the omni-wheels, those two components are completly independant. Obviously the human way to move is forwards, so the robot should do this as well. However because of the physical constraints of the real world, there should be a certain tolerance, since it is not possible for the robot to be turned to an exact orientation. Moreover slight sideways motion is actually helpful to make the robot appear more human like, because if the robot would be oriented exactly forwards, it would constantly be turning to adjust to its velocity.

Step 6. Robot Localization

For the robot to be able to navigate it needs to know its position on the global map, which, as mentioned before, is a non trivial problem. The first attempt to solve this problem was done with an ultrasound based indoor navigation system called marvelmind[15], which consists of stationary and mobile beacons and a timer. Four stationairy ultrasonic beacons were installed on the traverse at the labratory and the mobile beacon

[15]https://marvelmind.com/

was installed on the BMB. Its location is then calculated based on the propagation delay of the ultrasonic signal and trilateration. While this system is marketed to provide two centimeter precise location data of moving objects, including autonomous robots, the actual results were not promising. When standing still the robot could be located, but when moving the calculated position was jumping around up to one meter off the actual location of the robot, which might have been due to the noisiness of the mecanum wheels on the labratory's floor. However these noises are unavoidable and a system that is supposed to be applicable in different scenarios, including industrial ones, should not rely on silence. Therefore localization with the marvelmind system was not feasible in this context.

The second attempt at the localization problem was using a position calculation method called dead reckoning [20]. Traditionally used in marine and air navigation, it describes the process of localizing oneself by using a previous position, speed and course and trigonometrically calculating the new position. Based on the robots velocity v and the robots rotation r, the postion and orientation displacement can be calculated as follows:

$$\begin{aligned} \Delta v &= \Delta t \cdot \begin{pmatrix} cos(r) & -sin(r) \\ sin(r) & cos(r) \end{pmatrix} \cdot v \\ \Delta r &= \Delta t \cdot r \end{aligned}$$

The position obtained by this approach is subjected to accumulating errors, because both course and speed cannot be measured accurately by the robot's sensors. Traditionally this was counteracted by regularly adjusting the position at known points, such as specific landmarks. While this would also be possible in this context, it would make the localization dependant on the objects, that are used as landmarks, which would not only limit the environmental dynamic the algorithm can cope with, it would also require manually setting these landmarks.

A more elegant way to deal with this problem is to pair the dead reckoning system with the AMCL algorithm explained in the previous chapter. The order in which these two systems are applied is not arbitrary, as only the dead reckoning is smooth, while the AMCL is jerky. The position of the robot on the global map is calculated by applying transformations to its lastly calculated position. The first transformation that should be applied is the one resulting from the AMCL algorithm in order to prevent dead reckoning errors. The dead reckoning is then applied as a second transformation to obtain a smooth positioning. This combination then provides a mostly fluid positioning system which readjusts by itself.

Step 7. Safety

The importance of collsion avoidance has already been mentioned and as delays in the navigation algorithm are not readily completly avoidable the safety component should be a seperate module. This module has two functionalities both of which require knowledge of the closest obstacle. This information can be obtained by processing the data provided

by the robot's LRF. Firstly, all move commands are processed and depending on the distance to the closest obstacle the speed is reduced up to the point of no movement if an obstacle is in the safety critical zone. For the purpose of avoiding abrupt deceleration of the robot, this is done by decreasing the robot's speed proportional to the distance of the closest obstacle in a predefined interval, like follows with $d(c)$ being the distance to the closest obstacle, r the distance to the closest obstacle from which the robot's speed is reduced and s being the distance to the closest obstacle from which the robot is stopped.

$$\vec{v}_{safe} = \begin{cases} \vec{v} & , \ if \ d(c) > r \\ \vec{0} & , \ if \ d(c) < s \\ \frac{s-d(c)}{s-r} \cdot \vec{v} & , \ else \end{cases}$$

Secondly the position of these obstacles is send back to the navigation algorithm, so that the map can be updated. This step is particularly important if there are no other measures taken to update the map, such as cameras that detect new or moved obstacles.

As an example, if a human being comes within the first safety layer, the robot's speed will be reduced and a new path around the human will be calculated and executed. If the human clears the way, this is also detected and the robot will continue moving with its usual speed. If a human comes within the second safety layer, the robot will stop altogether and wait for the human to move away.

3.3 BMB Navigation Framework

Different steps of the navigation process explained above are specific to the human-like navigation algorithm, while others are more universal. A navigation algorithm can be designed in many different ways, for example to aid HRC, to avoid certain areas or to optimize path length or energy efficency. However, all of these different navigation styles have some common basic functionality, such as robot localization and collision avoidance. Quite possibly, there is no universally best navigation algorithm, but only algorithms that suit a specific purpose. For example, a robot programmed as a museum's guide should keep a certain distance to obstacles, so that a human can follow comfortably, while a robot working in logistics needs to move close to an obstacle in order to load. Although there are already countless approaches to robot navigation [18, 31, 25, 21], it is still a very open and actively researched topic [19, 22, 9]. As the BMB is also a research robot, it is instrumental to be able to run a navigation algorithm on it, which can result in substantial overhead of getting to know its specific controlling details. Therefore, a framework that allows for easily interchangeable navigation algorithms to be runable on the BMB would be valuable, both in a research context and also for general application.

The different layers of the navigation process, as stated in the Human-Like Navigation Section 3.2 are also represented in the architecture of the BMBNF, an overview of which can be seen in the image below. There are two main parts, the ready-made ROS part and the java part, which consists of interfaces, abstract classes and readily implemented classes.

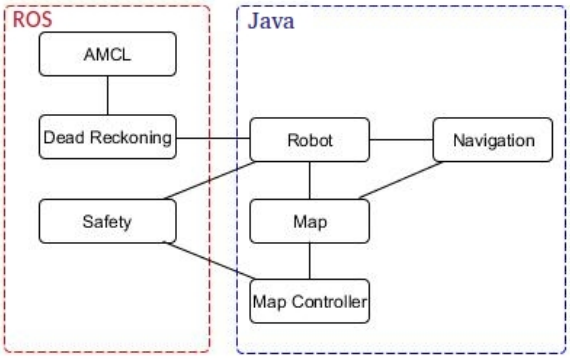


Figure 22: BMBNF overview

The ROS part of the framework does not need to be altered, as it provides the basic functionality needed for the navigation, such as the global localization of the robot and a safety component.

The Java part of the BMBNF provides the general structure needed for a navigation algorithm, such as the map, a map controller that keeps it up to date and the actual navigation algorithm. More details are given in the following chapter.

4 Implementation

In the context of this thesis, a framework has been developed to integrate self written navigation algorithms easily on the BMB. This chapter describes the implementation of the BMB navigation framework (BMBNF) and the realization of the ideas formed in the concept chapter.

4.1 BMBNF

It was decided to implement the extendable part of the BMBNF in Java, because this allows its usage without knowing any BMB specific controlling details. However, since on the BMB the controlling commands are executed via ROS, there is also part of the framework, which is written in C++. The two parts are connected via rosbridge, which provides a JSON API to ROS functionality, such as publishing on topics and subscribing to topics.

4.1.1 ROS

As has been described in chapter 2.2.1 ROS programms consist of nodes and asynchronous communication is implemented via a publish/subscribe structure and topics. A graphical representation of said structure of the BMBNF is shown in figure 23.

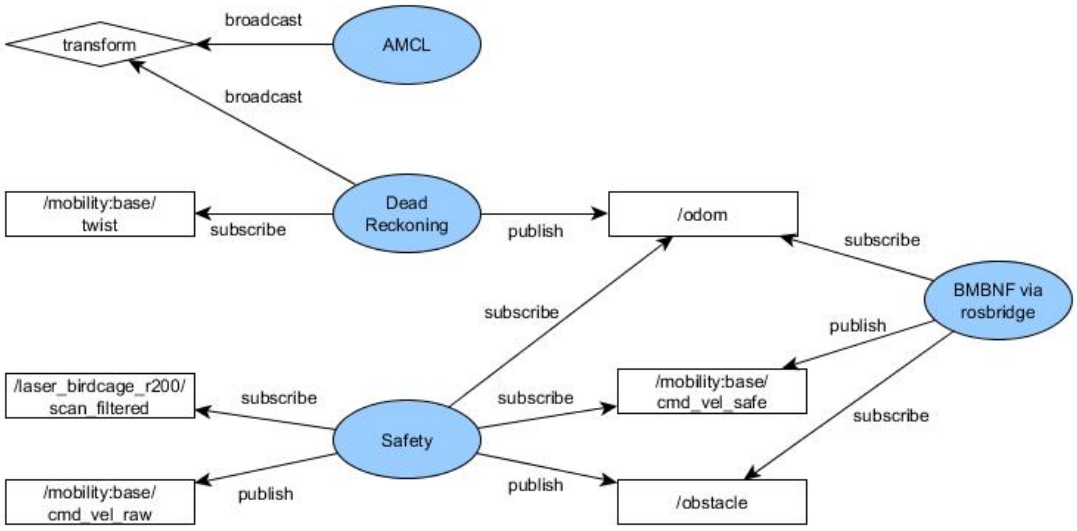


Figure 23: ROS nodes of the BMBNF

Both the AMCL node and the Dead Reckoning node broadcast transform data. The Dead Reckoning node is subscribed to the twist topic, which contains information about the robot's velocity, that is used to calculate the position. It publishes this position on the odometry topic. While odometry usually means the calculation of a robot's position

based on information from rotary encoders and the circumference of its wheels, ROS uses the term more generally. In this context, odometry simply means the estimation of a robot's position relative to an initial location, no matter where this information stems from. More detail on the localization process is given in the localization section below. The safety node is subscribed to the laser scan topic, as it needs this information to calculate the position of new obstacles. This position is then published on the obstacle topic which is subscribed by the Java part of the BMBNF. The laser scan information is also needed to reduce the robot's speed by piping the move commands from the Java part of the BMBNF through the safety node. Therefore the safety node is subscribed to the safe command topic and publishes on the command topic. Commands published on this topic are then executed by the BMB. Detailed information about this process is given in the safety section.

Localization

In ROS, transforms are a way to keep track of multiple coordinate frames and their relationship towards each other. This information is stored in a so called transform tree. The BMB has its own coordinate frame, called *base_frame*, with its center as the origin and the global map has a coordinate frame, called *map_frame*. The odometry frame transforms the *base_frame* to the *map_frame* in order to get the robot's global position. In figure 24 a comparison of localization via dead reckoning and localization via AMCL in addition to dead reckoning can be seen.

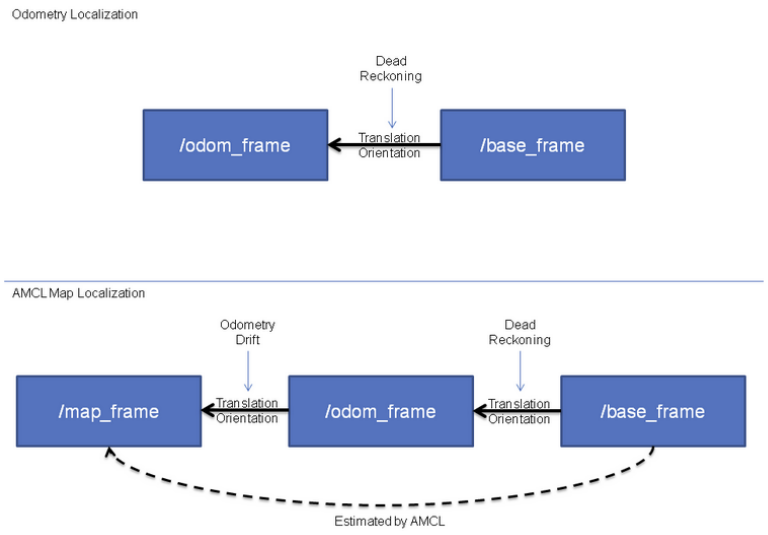


Figure 24: Robot localization via dead reckoning in compairison to AMCL[16]

The AMCL node transforms sensed laser data to the odometry frame and in doing so estimates the transformation from the *base_frame* to the *map_frame*. However it

[16]http://wiki.ros.org/amcl

only broadcasts the transformation between the *map_frame* and the odometry frame to account for the drift of the dead reckoning. If this step is ommitted, the errors of the dead reckoning will accumulate, leading to an increasingly inaccurate position.

Safety

The main task of the safety component of the BMBNF is the avoidance of any collisions and also to send newly detected obstacles to the map. As both of these tasks require knowledge of the closest obstacle, the safety node is subscribed to the laser scan topic and calculates the position and distance to the closest obstacle, which is shown in algorithm 1. It would also be possible to consider all obstacles near by, however this is not necessary, because the closest one would lead to a collision first.

Algorithm 1 Calculate position of and distance to closest obstacle

Require: laserscan
Ensure: P = position of closest obstacle, d = distance to closest obstacle
  $d \leftarrow 10$
  for i = 0,...,scan.ranges.size() **do**
    if $scan.ranges[i] < d$ **then**
      $d \leftarrow scan.ranges[i]$
      $obs\_r \leftarrow scan.angle\_increment \cdot i$
    end if
  end for
  if $d < send\_to\_map\_distance$ **then**
    $r \leftarrow pos_r + obs_r$
    $delta_x \leftarrow -sin(r) \cdot d$
    $delta_y \leftarrow cos(r) \cdot d$
    $P_x \leftarrow pos_x + delta_x$
    $P_y \leftarrow pos_y + delta_y$
  end if

Only obstacles, that are being detected closer than 10 m away from the robot need consideration, because the robot is not moving fast enough for there to be any safety concerns with obstacles further away than 10 m. Thus the minimal distance to any obstacle is initialized with 10 m. Then all data from the laser scan (scan.ranges) is iterated over and the closest detected obstacle's data is being stored. The laser range finder detects the distance to any obstacles and depending on the data's position in the ranges array, the angle in which the robot stands to the obstacle can be calculated using the angle increment between each laser detection point. The distance from which obstacles are being send to the map is customizable, as for many applications 10 m is too rough and maybe only obstacles within a radius of 2 m to the robot need to be considered. The position of the obstacle can be calculated with the data about its angle and distance and using trigonometrical functions.

The velocity commands are published on the safe velocity command topic, which the

safety node is subscribed to. This velocity is then scaled according to the distance measured by the laser scanner and published on the actual velocity command topic.

4.1.2 Java

The Java part of the framework handels any communication to the ROS components, so there is no need to become acquainted with them, in order to use the BMBNF. The published obstacles are handeled by the MapController and the move commands are handeled by the BMB class. The Map is implemented as an interface, so the MapController can work with it, but any new environment needs to be configurated by letting the software know, where the map can be found, so this part can be extended to integrate a self written map for the navigation algorithm.

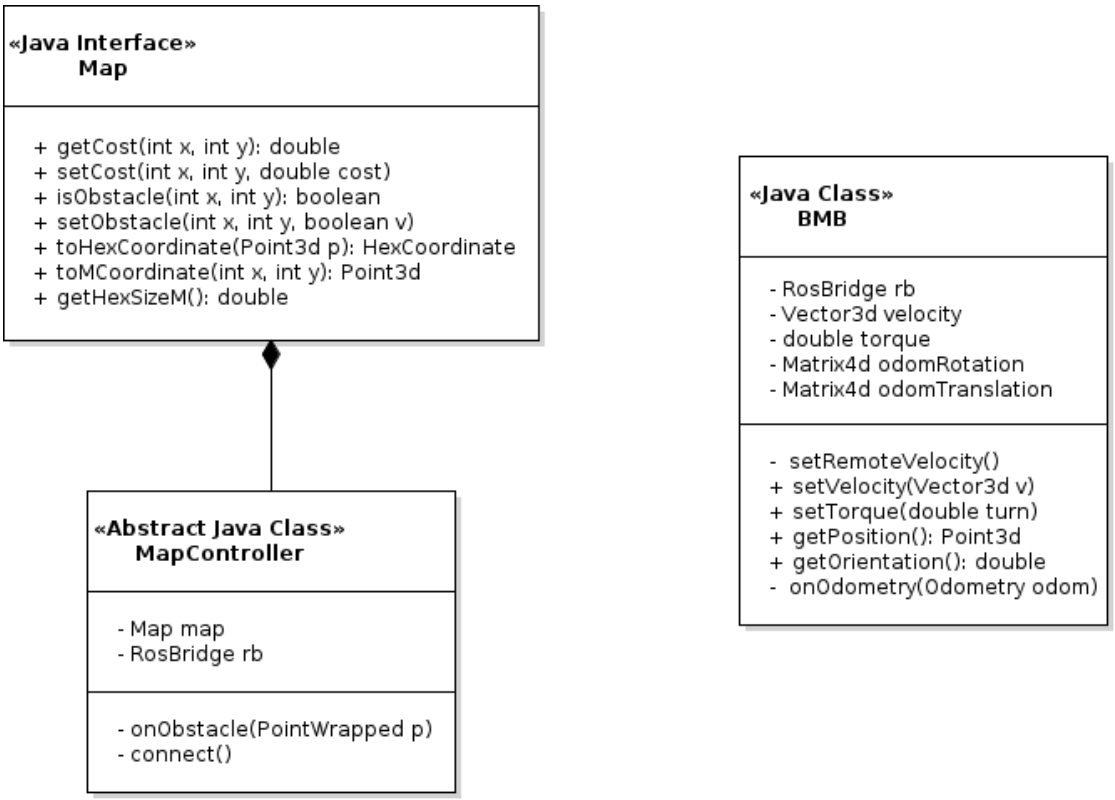


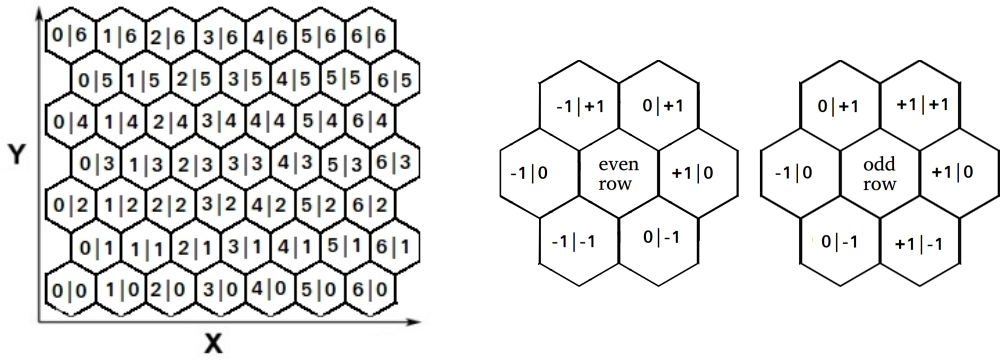
Figure 25: UML of BMBNF's Java classes

The UML diagram in figure 25 shows these classes in more detail. The Map interface includes the methods needed for a path planning algorithm, meaning the managing of costs and obstacles, as well as the conversion between hexagon coordinates and world coordinates. The MapController class requires access to such a map, in order to handle new obstacles, that are communicated via rosbridge. The BMB classes rosbridge has two tasks. Firstly, the robot's odometry is communicated via ROS and stored in the

BMB class to be accessed by the getPosition and getOrientation methods. Secondly, it publishes the move commands set by the setTorque and setVelocity methods.

Map

There are different ways to handle a hexagon map, including different ways to arrange the hexagons and different coordinate systems. For more details refer to [1]. The method used in this thesis, is to offset every other row, but it would be just as possible to offset every other column instead. With this arrangment offset coordinates have been used, because they match the standard cartesian coordinates used with square grids, as can be seen below, in figure 26(a). Therefore, this option is more intuitive than cube coordinates or axial coordinates. As the odd rows are offset, the calculation of the neighbour coordinates differs for even and odd rows, as can be seen in figure 26(b).



(a) Offset coordinates (b) Neighbour coordinates

Figure 26: Offset coordinates on hexagon grid

Using the cartesian coordinates, the cost of each map tile could be stored in a two dimensional array. For performance reasons, however, a one dimensional double array should be used, as the coordinates can easily be converted as follows:

$$2D \rightarrow 1D : x' = width \cdot x + y$$
$$1D \rightarrow 2D : x = x' div\ width, y = x' - x' div\ width$$

BMB

The BMB class is responsible for the communication between the Java and the ROS components of the framework. It holds the velocity and torque the real robot is supposed to move with and sends these values as commands via rosbridge. This class also holds data about the robot's position, that is obtained by subscribing to the odometry topic.

4.2 Human-Like Navigation

The following section describes how the BMBNF can be used to implement a human like navigation algorithm, by realizing the ideas of the concept chapter.

Step 1. Map Controller

The flow procedure explained earlier can be implemented by continuously iterating over the map and adjusting any values, that differ from the value a tile is supposed to have, according to its neighbour's values. On a large map, however, this is highly inefficient, especially if there are only few obstacles or only a few familiar areas. A more efficient approach is to store any changes to the map and only iterate over the immediate environment of the changed tile, which can lead to more changes, that can also be stored and reevaluated. This allows for parts of the map that have not been changed to simply be skipped, which obviously saves time.

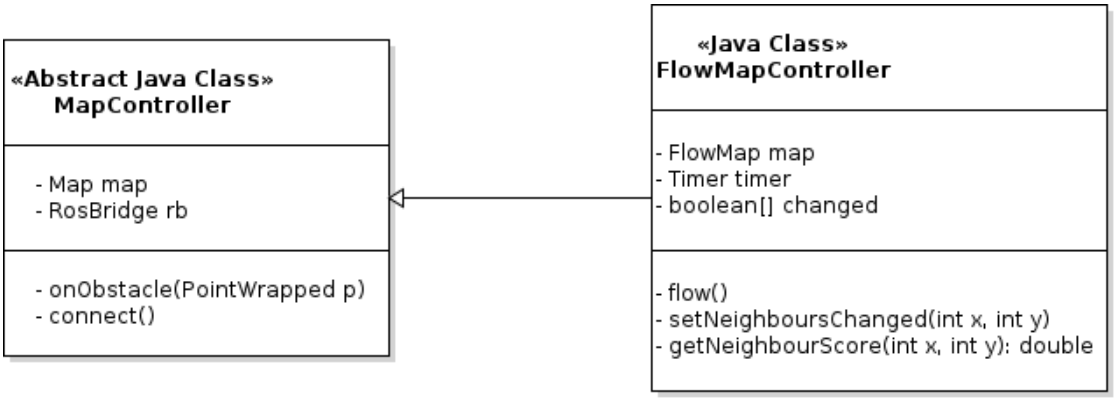


Figure 27: UML MapController

The UML diagram 27 above shows the abstract MapController class from the BMBNF, as well as the FlowMapController realizing this class for the human-like navigation algorithm. Since the FlowMap offers additional functionality compared to the abstract Map class, it is one of the attributes. A timer is needed, to continuously check for changes of the map and these changes are stored in the changed array, so they can be evaluated. This evaluation is done in the flow method, which uses the setNeighbourChanged method to store its changes in the changed array and the getNeighbourScore method to calculate the obstacle and familiarity values according to the neighbours' values.

Step 2. Path Planning

Using the obstacle and familiarity values that have been calculated by the MapController, the path planning procedure can be implemented as a standard A* algorithm. The cost function g includes the number of hexagons traveled so far, including the current one, an added cost c for every change of direction, as well as the obstacle values o and negative

familiarity values f of the hexagon tiles that would have been covered by the robot, so far.

$$g(n) = g(pre(n)) + 1 + c + o - f$$

The hexagons traveled so far is the standard value for the A* algorithm. The cost for changing directions is added to minimize zigzag patterns in the planned path. The obstacle values are added to ensure that there is sufficient space between any obstacles and the planned path and the familiarity values are subtracted to make sure familiar paths are preferred.

The heuristics function h ist simply the distance in hexagons to the desired target t.

$$h(n) = d(n, t)$$

Step 3. Smoothing

The implementation of the smoothing process via determining the midpoints in two iterations is very straight forward. Firstly, the hexagon coordinates from the path planning step have to be converted into two dimensional points on the continuous map. This is done by calculating the midpoint of every hexagon like shown in algorithm 2.

Algorithm 2 Calculate coordinates of hexagon midpoint

Require: $x,\ y$ cartesian coordinates of hexagon
Ensure: P is midpoint of hexagon (x, y)
  if $y\ mod\ 2 = 0$ **then**
    $xShift \leftarrow x \cdot HEX\_WIDTH$
    $yShift \leftarrow y \cdot HEX\_Y\_OFFSET$
  else
    $xShift \leftarrow HEX\_X\_MIDDLE + x \cdot HEX\_WIDTH$
    $yShift \leftarrow y \cdot HEX\_Y\_OFFSET$
  end if
  $P_x \leftarrow ((HEX\_X\_MIDDLE + xShift)/HEX\_HEIGHT) \cdot getHexSizeM()$
  $P_y \leftarrow ((HEX\_Y\_MIDDLE + yShift)/HEX\_HEIGHT) \cdot getHexSizeM()$

The algorithm needs the values *HEX_WIDTH, HEX_Y_OFFSET, HEX_X_MIDDLE* and *HEX_HEIGHT*, whose meaning can be seen in figure 28. They can all be calculated from the initially defined hexagon size, as shown below. The calculation of the midpoint of a hexagon obviously depends on, whether this hexagon is part of an even or an odd row, but other than that, it is pretty straight forward.

$$HEX\_HEIGHT = HEX\_SIZE$$
$$HEX\_WIDTH = \frac{\sqrt{3}}{2} \cdot HEX\_HEIGHT$$
$$HEX\_X\_MIDDLE = \frac{1}{1} HEX\_HEIGHT$$

$$HEX\_Y\_MIDDLE = \frac{1}{2} HEX\_WIDTH$$

$$HEX\_Y\_OFFSET = \frac{3\sqrt{3}}{8} \cdot HEX\_HEIGHT$$

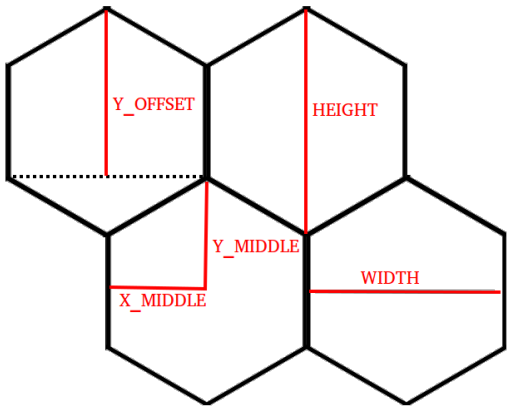


Figure 28: Hexagon constants

The resulting list of 2D points has to be iterated over twice. The midpoint M of two adjacent points P and Q is calculated in a typical manner given the coordinates of two points, which results in a smoothed over list of two dimensional points.

$$M = \left(\frac{P_x + Q_x}{2}, \frac{P_y + Q_y}{2} \right)$$

Step 4. Path Execution

The BMB is controlled by two three dimensional vectors, the translational vector, called linear, and the rotational vector, called angular. In linear, only the x and y coordinates are needed, as the robot cannot fly and in angular, only the z coordinate for yaw rotation is needed, as the BMB can perform neither a pitch nor a roll rotation. In algorithm 3, a scheme of the calculation of these vectors is illustrated.

This calculation needs to take place, as long as the robot has not reached its target, which is checked with some tolerance in mind, because it is not possible to navigate the BMB to a millimeter exact position. In every iteration, there is a check whether the path needs to be recalculated, which would be the case, if an obstacle has been moved to block the path ahead or the robot has moved too far off the calculated path. The algorithm calculates the point on the path which is closest to the robot and the vector from the robot to this point, as well as the look ahead point on the path and the vector to that point. The actual velocity is then calculated from these two vectors like explained in the concept chapter and if the deviation of the robot's orientation to the direction of the velocity exceeds a tolerance, the robot's rotational componend is set according to this

Algorithm 3 Calculate commands to execute path

Require: path
  while ¬ hasReachedTarget() **do**
    if needs replanning **then**
      replan();
    end if
    $velocity \leftarrow calcVelocity(closestPoint, lookaheadPoint)$
    $orientation \leftarrow vectorsOrientation(velocity)$
    if absolute(orientation - robot.getOrientation()) > TOLERANCE_DEGREE **then**
      $robot.setTorque \leftarrow getTorque(orientation)$
    end if
    $robot.setVelocity \leftarrow transformVelocity(velocity, orientation)$
  end while

velocity's direction. The velocity is then transformed to match the direction the robot is currently facing and then the robot's translational component is set, as well.

4.3 Use and Interplay

The navigation algorithm's main thread is the Navigator class, an overview of which can be seen in the UML diagramm in figure 29. Its run method implements the state machine from figure 19 in the concept chapter, hence the state attribute. This method waits for the setTarget method to be called, to provide a new target and orientation to move to. This method can either be called via an RPC client or a web service. The commands needed to reach this target are then calculated as explained in the steps above, which is why the BMB, FlowMap, Smoother and AStar attributes are needed.

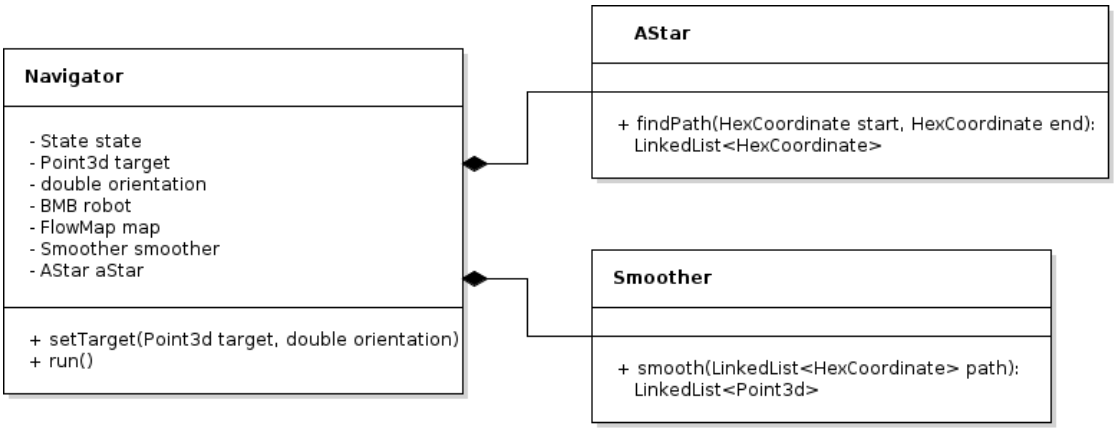


Figure 29: UML Navigator

5 Evaluation

Prior to letting the BMB navigate in a real scenario, some of the adjustable parameters are being evaluated with a simulated robot.

5.1 Simulations

The parameters, which are being evaluated include the size of the hexagon grid, the flow value that specifies how far the outer edges of obstacles are being blurred, as well as, the lookahead value that determines the point on the planned path, that is used to calculate the robot's velocity. There is also a simulation, which shows the preference of familiar areas.

5.1.1 Hexagon Grid Size

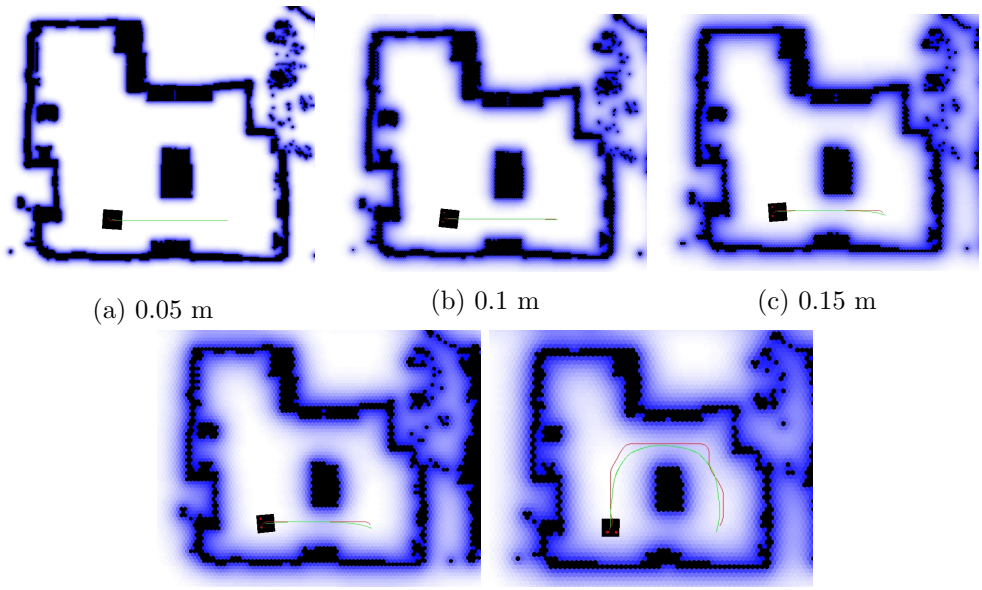
For the first simulation, the map of the P4PLab, which can be seen in figure 30, is being transformed to a hexagon grid of different sizes. On each of these maps, a path is being planned from the same start to the same target position. The resulting paths on the hexagon maps are shown in figure 31 and the computation time of the path planning procedure is shown in table 2.

Figure 30: P4PLab

Figure 31 shows that with increasing hexagon size, the position of obstacles becomes less refined and on the last map, the obstacles are too distorted to allow the path to be planned in the same way as before and a detour is planned instead, altough the actual distance between the obstacles did not change. For this reason, the hexagon size is set to 10 cm for the following scenarios, because this way obstacles are not too distoreted, while the computation time is reasonable.

Hexagon size in cm	Computation time in Milliseconds
5	670
10	620
15	147
20	63
25	51

Table 2: Path planning time subject to hexagon size



(a) 0.05 m (b) 0.1 m (c) 0.15 m

(d) 0.2 m (e) 0.25 m

Figure 31: Evaluation of different hexagon sizes

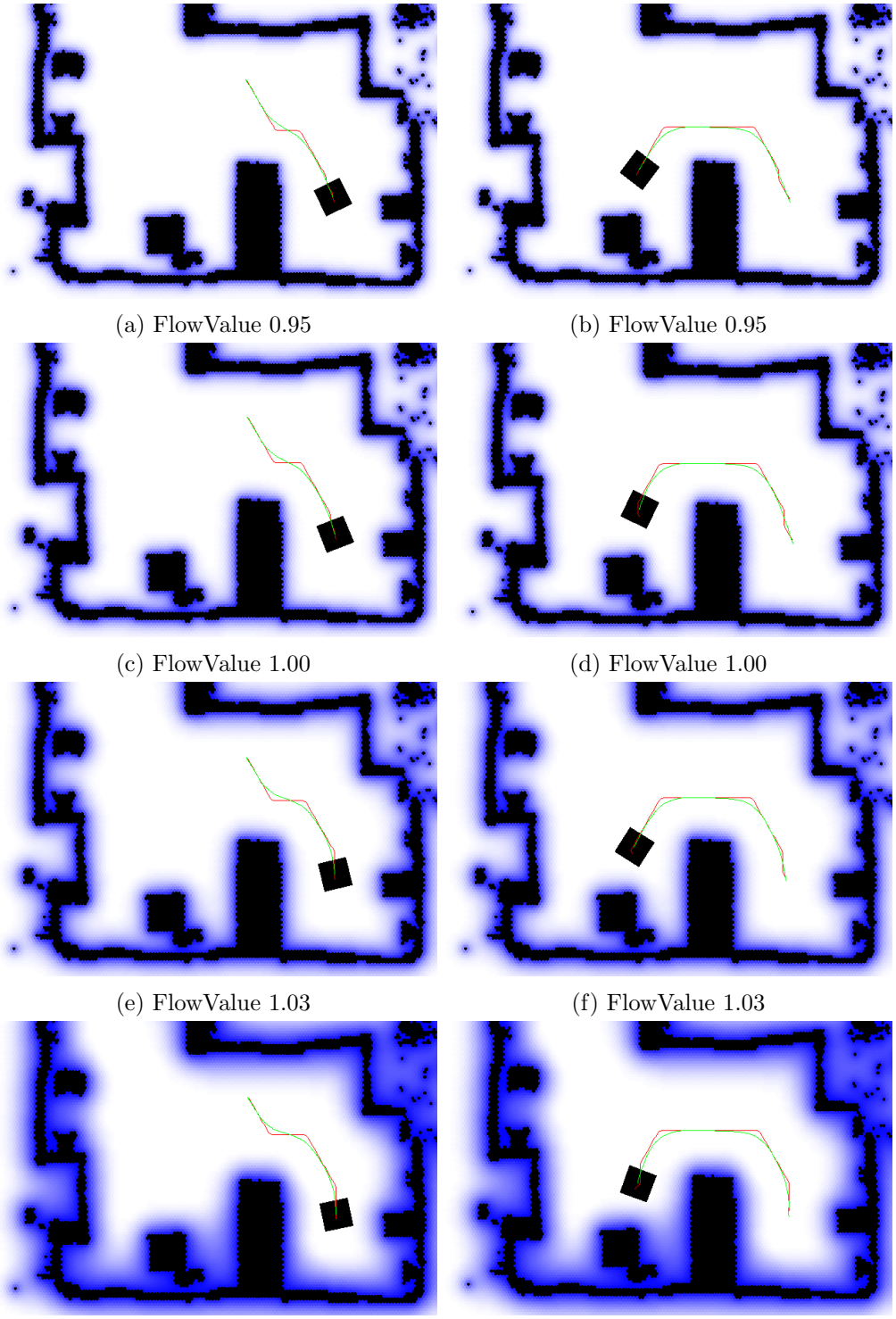
5.1.2 Flow Value Evaluation

In the second simulation set, the flow value is being evaluated in two fixed contexts, meaning that the start and target position is the same respectively and all other adjustable parameters are fixed. The results of these simulations can be seen in figure 32. The red line represents the planned path and the green line represents the robot's actual trajectory. In order to analyse these simulations, the average distance from the robot to the work bench has been calculated, while the robot was driving around it. The results for each flow value are the following:

FlowValue	Distance from planned path to workbench in hexagons	Distance from robot to workbench in cm
0.95	13	90
1.00	15	110
1.03	16	120
1.05	18	140

Table 3: Resulting distance with different flow values

Which one of these flow values is best suited in a specific context depends on the scenario. Considering that the workbench is a static obstacle, which is part of the map and therefore does not need to be detected by the robot, the flow value 1.03 has been chosen to be used in the following scenarios. This decision was made to ensure the prevention of any collisions with moving obstacles, that have to be detected and processed.



(a) FlowValue 0.95 (b) FlowValue 0.95

(c) FlowValue 1.00 (d) FlowValue 1.00

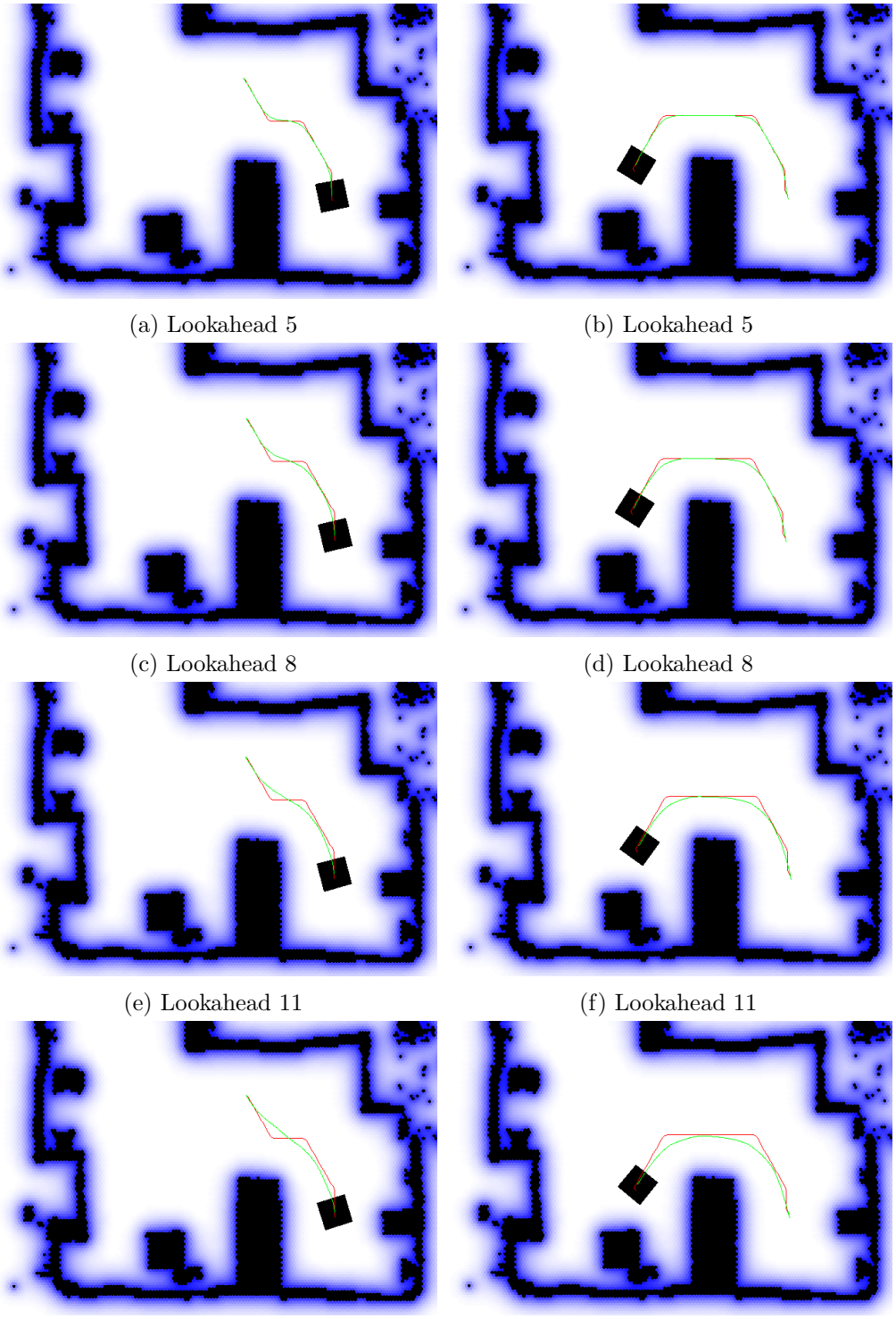
(e) FlowValue 1.03 (f) FlowValue 1.03

(g) FlowValue 1.05 (h) FlowValue 1.05

Figure 32: Simulations with different flow values.

5.1.3 Lookahead Evaluation

In the third simulation set, the same contexts are used, except for the flow value, which is set to 1.03 and the lookahead of the navigation process is being adjusted. The results can be seen in figure 33. One can observe, that the lower the lookahead value is set, the closer the actual trajectory is to the planned path, which is exactly the expected outcome. The simulations with the highest lookahead value seem to be the most human-like, however there is quite a substantial deviation from the planned path. For this reason, the lookahead 8 is chosen for the following scenarios, because the resulting trajectory has the desired smoothness and roundedness of a human trajectory, whilst still being close enough to the planned path, that with the flow value 1.03, the robot will not try to navigate too close to an obstacle.



(a) Lookahead 5 (b) Lookahead 5

(c) Lookahead 8 (d) Lookahead 8

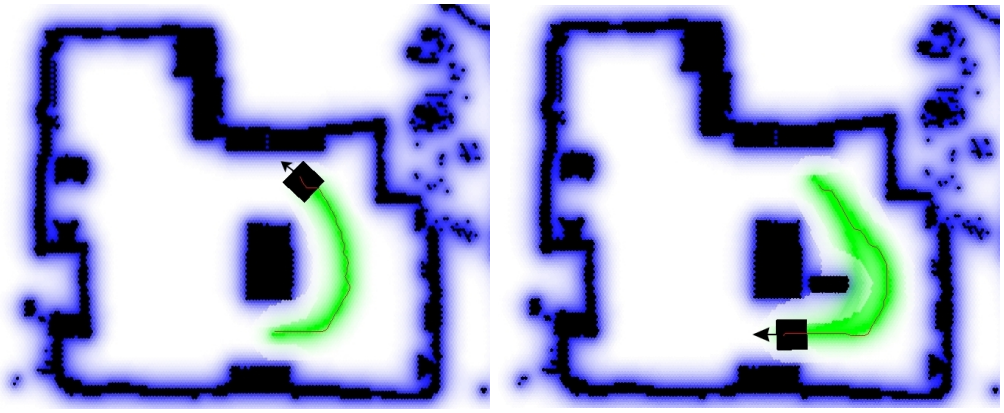
(e) Lookahead 11 (f) Lookahead 11

(g) Lookahead 14 (h) Lookahead 14

Figure 33: Simulations with different lookaheads values.

5.1.4 Familiarity Value Evaluation

In the fourth simulation, the preference of familiar areas is being evaluated. For this purpose, the simulated robot first navigates around a workbench to establish a familiar area. Driving around the workbench on the left side would take roughly 6.8 m, while driving around the right side takes 6 m. Therefore, the algorithm plans a path on the right side of the workbench. The resulting familiar area can be seen in figure 34(a). After this, an obstacle is added on the right side of the workbench, for example a chair that has been moved there. The algorithm still plans around the right side, although this path is roughly 8 m long and thus longer than the path around the left side. This helps to predict the robot's behavior, because if the robot has the task to get an object and come back to its original position, it would be unexpected if the robot chose an entirely different path after getting the object. The resulting path is shown in figure 34(b). How much longer the familiar path has to be to cause the planning algorithm to plan around the left side depends on how much weight the familiarity values are given in the path planning process.



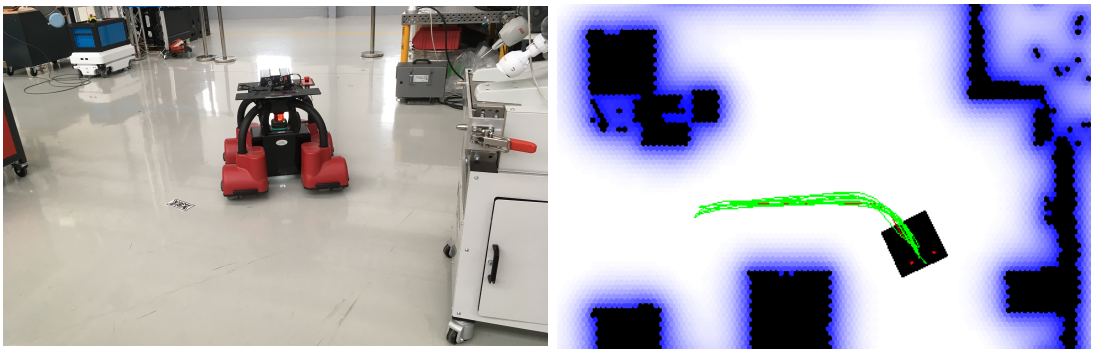
(a) Establishing of familiar area (b) Utilization of familiar area

Figure 34: Path planning based on familiarity values

5.2 Application

For the following scenarios, a fixed set of parameters is being used. According to the previous evaluations, the hexagon size is set to 10 cm, the flow value is set to 1.03 and the lookahead is set to 8.

The purpose of the first scenario is to see how stable the navigation algorithm works on the real robot. For this, the robot navigates between two fixed points ten times. The taken trajectories can be seen in figure 35(b) with the corresponding picture of the real robot in figure 35(a). While the trajectories are not perfectly aligned, they are similar enough, that a human bystander would not be able to tell a difference, which gives the algorithm the desired stable appearance.



(a) BMB at target position

(b) Trajectories taken by the BMB

Figure 35: Scenario 1

The second scenario evaluates, how a static obstacle, which is not part of the map, is being avoided. For this purpose, a cardboard box is placed in front of the robot before the new target is set. This setup can be seen in figure 37.

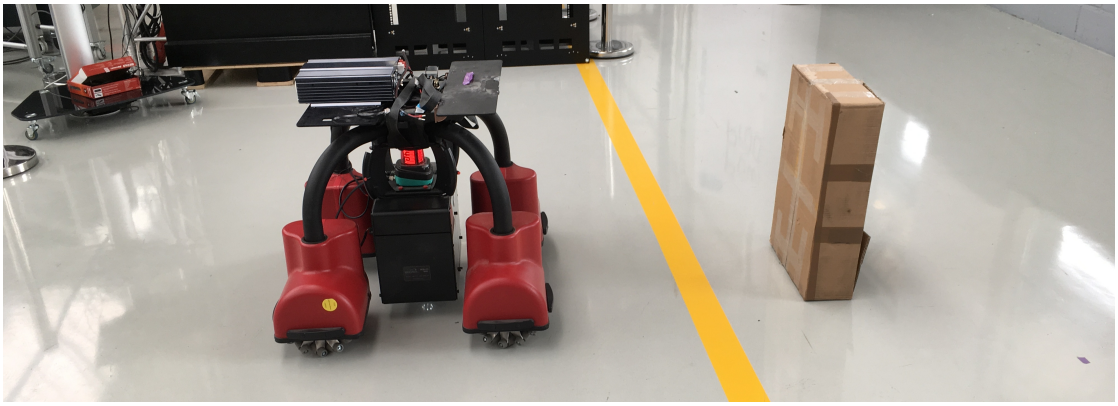


Figure 36: Setup of scenario 2

As can be seen in figure 37, the box is being detected and the path is being planned around it, just as if the box was part of the initial map and the actual trajectory is

smooth and rounded. As the box was detected prior to setting the new target, there was no recalculation of the planned path.

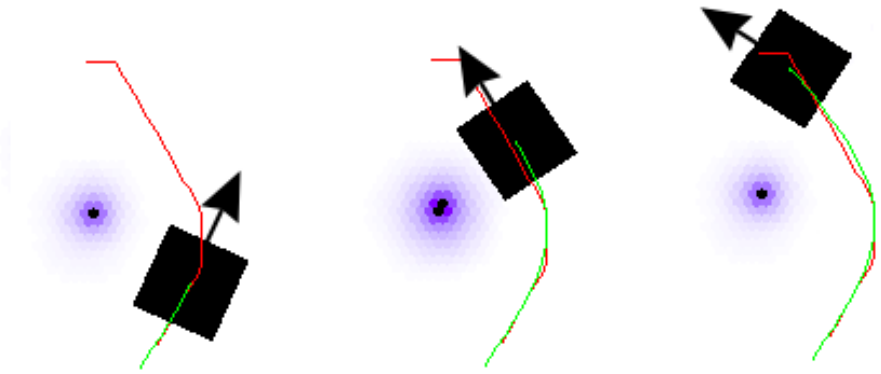


Figure 37: Planned path and trajectory of scenario 2

In the third scenario, the box is placed in front of the robot at a distance of roughly 1m, after the target has been set and the robot is already moving, to evaluate the dynamic obstacles avoidance. This setup can be seen in figure 38(a).

(a) Box has been placed (b) Collision avoidance

Figure 38: Scenario 3

The resulting planned path and trajectory is shown in figure 39. The robot first follows the initially planned path. After the box is placed, it is detected and the robot stops and recalculates its path, which is then followed. As can be seen in figure 38(b), the robot keeps a sufficient distance to the obstacle. In this scenario, the minimum distance between the robot and the box was roughly 50 cm at the point where the robot was replanning its path, which is still safe, as the robot only moves with moderate speed. When the box was placed closer to the robot, it stopped moving and could not reach its target, which is the desired behavior to prevent any collisions. The robot would only

continue moving, when the box was moved further away.

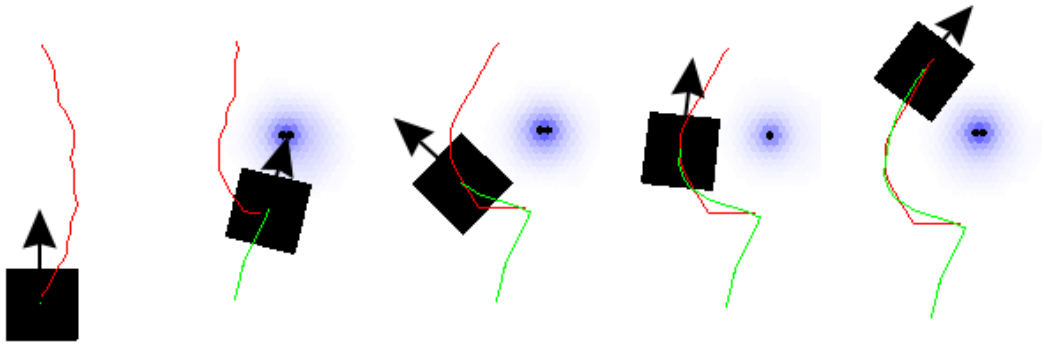


Figure 39: Experiment with different lookaheads values.

5.3 Human-like characteristics

Lastly, the human-like navigation characteristics are being evaluated, based on the list in the concept chapter 3.

- Humans prefer to walk through familiar areas: This characteristic is met, as has been evaluated in section 5.1.4.
- The length of a path has only a secondary role, but it is still considered: Since the path planning algorithm is based on an A* algorithm, the length of the path is being considered, but the distance to obstacles and familiar areas are more important.
- Wider areas are generally preferred over areas, that are too narrow to comfortably walk through: The preference of wider areas is realized through the obstcale values and the keeping of a certain distance to obstacles, as can be seen in the evaluation scenarios above.
- Humans tend to keep some distance to obstacles, if possible: Both the simulations and the actual applications show, that this characteristic is being fulfilled.
- When walking through a narrow segment, humans tend to walk right in the middle of the obstacles: This characteristic is also being satisfied by the obstacle values.
- Paths taken by humans are rounded with smooth curves and there are generally no sharp turns or zigzag: The navigation algorithm is conform to this point, because all paths are being smoothed after planning and they are executed in a way, that leads to an overall smooth locomotion, as can be seen, for example, in the application section 5.2.

- Humans naturally walk forwards and neither sideways nor backwards: The path execution technique makes sure this point is met, which can also be seen in the application section 5.2.

6 Future Work

This chapter gives an outlook on possible future work building on top of this thesis.

One of the goals of the navigation algorithm introduced in this thesis was the avoidance of any obstacles to prevent collisions, which was achieved by keeping a set distance to obstacles at all times. Depending on the scenario, however, it may be necessary to approach an obstacle. If the Baxter robot is supposed to take a load out of a shelf, for instance, this cannot be done whilst standing half a meter away from said shelf. Therefore it would be useful to include a mode which allows the approaching of specific obstacles if this was specifies within the task, in order to ensure the distance only falls below the safety limit in these contexts. Furthermore, it may be necessary to step away from an obstacle backwards, for example when taking something out of a shelf, because turning without stepping back first may cause a collision with said shelf. This is also not included in the current version of the navigation algorithm, because the approaching of obstacles is not supported yet. These two enhancements should go hand in hand.

There are two elements, which could be elaborated to further improve the human-likeness of the proposed navigation algorithm. Firstly, as mentioned in the goals section, this work did not include any research on the acceleration and deceleration process in human walking, although it may be beneficial to include these details in the path execution part of the navigation algorithm. Secondly, the target orientation of the robot was not considered in the path finding algorithm and the robot is simply turned at the target, if necessary. It would be more human-like to adjust the planned path, so that the robot is facing the desired direction without the need to turn.

In furtherance of the applicability of the BMB with its new navigation algorithm, it would be useful to connect it to the TEDURU server [15] mentioned in the concept chapter. This way, the robot could not only be used in a self-contained context, but also be included in any HRC scenario managed by TEDURU.

Lastly, the ideas formed in the concept chapter are not BMB specific. Therefore, they could also be extended to many different robots, to make them more predictable and more intuitive to work with, as well.

7 Conclusion

In this thesis, a navigation algorithm for human-like path planning has been developped, that can easily be applied in practical scenarios, which is a key piece still missing in human-robot collaboration. To achieve this, a hexagon grid is introduced, to solve the distance issue of diagonal steps on a square grid. The actual path planning is done on this hexagon map in consideration of some of the key aspects of human pathfinding, such as keeping a distance to obstacles, which is done by diffusing their outer edges and preferring familiar areas. To overcome the issues connected to planning on a grid, such as zigzag patterns, the path is then smoothed out by an optimizer. This path is executed with a focus on a point further ahead on the path to account for another key aspect of human pathfinding, namely the roundedness of walking trajectories. All this is done to make the robot's movements more intuitive and easier to predict, in favor of better human-robot collaboration. In order to achieve this, it is not necessary to imitate real human trajectories to the millimeter exact, as there are individual differences between people anyways and humans are generally used to being around very different people. Thus, the work introduced in this thesis is a simple yet effective approach to the human-like path finding problem. As has been evaluated, the algorithm is able to cope with dynamic environments, which makes it applicable in real HRC scenarios.

In the course of the implementation of the human-like navigation algorithm, a software framework has been developped, which enables the easy deployment of a self written navigation algorithm on the BMB. This framework allowed to simply implement the human-like navigation algorithm as a Java project, without dealing with BMB specific controlling details and it would be easy to develop different navigation algorithms, building on top of this framework.

Acknowledgements

I would like to express my deep gratitude to my supervisor, Christian Bürckert, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I also want to thank Prof. Dr. Wahlster and the DFKI GmbH who provided the required resources for this research. Finally, I wish to thank my family and friends for their support and encouragement throughout this work.

Saarbrücken, 9th April 2018

Jessica Lackas

8 Appendix

In the following sections, an installation guide for deploying the human-like navigation algorithm on a BMB is provided, along with a startup guide and a guide for calling the algorithm via a web interface.

8.1 Installation guide

This guide explaines, how the navigation system can be deployed on a BMB.

1. Install a current version of ROS on the BMB. The algorithm was developped and tested with ROS Indigo Igloo[17].
2. Set up the catkin build system[18].
3. Download the mobility_base package[19].
4. Download the navigation stack[20]. The amcl package and the map_server package will be used.
5. Provide the map_server with a map of the environment.
6. Download the odometry and the safety package from the C++ folder at https://github.com/JessicaLac
7. Invoke catkin_make to build the odometry and safety nodes.
8. Download the Java project from https://github.com/JessicaLackas/HumanLikeNavigation.git. All required dependencies and executions are specified in the pom.xml.
9. Build the executable .jar file.

8.2 Startup guide

This guide explaines, how the navigation system can be started on the BMB.

1. Run the mobility_base_bringup to start the ROS master.
2. Run the odometry node. It is beneficial to provide a rough estimate of the initial position.
3. Run the map_server with the previously generated map file.
4. Run the amcl node.
5. Run the safety node.
6. Run the previously build .jar file.

[17] http://wiki.ros.org/indigo
[18] http://wiki.ros.org/catkin
[19] http://wiki.ros.org/mobility_base
[20] http://wiki.ros.org/navigation

8.3 Webservice guide

This guide explaines, how the navigation system can be called via a web service.

1. Download the Java project from https://github.com/JessicaLackas/HumanLikeNavigation.git. All required dependencies and executions are specified in the pom.xml.
2. Start the SpringBoot application.
3. Open the bmbsettarget.html file in any browser.
4. Set the desired target.

References

[1] Hexagon grids. https://www.redblobgames.com/grids/hexagons/. Accessed: 2018-03-30.

[2] ROS in a nutshell. https://www.generationrobots.com/blog/en/2016/03/ros-robot-operating-system-2/. Accessed: 2018-03-29.

[3] ROS wiki. http://wiki.ros.org/ROS/Introduction. Accessed: 2018-03-15.

[4] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. Int. J. Comput. Games Technol., 2015:7:7–7:7, January 2015.

[5] G. Arechavaleta, J. p. Laumond, H. Hicheur, and A. Berthoz. Optimizing principles underlying the shape of trajectories in goal oriented locomotion for humans. In 2006 6th IEEE-RAS International Conference on Humanoid Robots, pages 131–136, Dec 2006.

[6] M. Betke and L. Gurvits. Mobile robot localization using landmarks. IEEE Transactions on Robotics and Automation, 13(2):251–263, Apr 1997.

[7] D. C. Brogan and N. L. Johnson. Realistic human walking paths. In Proceedings 11th IEEE International Workshop on Program Comprehension, pages 94–101, May 2003.

[8] Starkey N. J Charlton, S. G. Driving on familiar roads: Automaticity and inattention blindness. Transportation Research Part F: Traffic Psychology and Behaviour, 19:121–133, 2013.

[9] Ching-Yu Chou and Chia-Feng Juang. Navigation of an autonomous wheeled robot in unknown environments based on evolutionary fuzzy control. Inventions, 3(1), 2018.

[10] Debora Clever and Katja D. Mombaur. An inverse optimal control approach for the transfer of human walking motions in constrained environment to humanoid robots. In Robotics: Science and Systems, 2016.

[11] Carnegie Mellon Debra Bernstein, Kevin Crowley. Working with a robot: Exploring relationship potential in human–robot systems. Psychological Benchmarks of Human–Robot Interaction, page 465–482, October 2007.

[12] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), volume 2, pages 1322–1328 vol.2, 1999.

[13] Peng Duan, Guohui Tian, and Hao Wu. Mobile robot navigation based on human walking trajectory in intelligent space. 7:249–258, 11 2014.

[14] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. IEEE Robotics Automation Magazine, 13(2):99–110, June 2006.

[15] Andreas Fey. A Dual Reality Approach for Spatial Inference, Planning, Prediction and Prototyping. Bachelor's thesis, Saarland University, 2018.

[16] A. Fitzgibbon, M. Pilu, and R. B. Fisher. Direct least square fitting of ellipses. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(5):476–480, May 1999.

[17] A. Freedy, E. DeVisser, G. Weltman, and N. Coeyman. Measurement of trust in human-robot collaboration. In 2007 International Symposium on Collaborative Technologies and Systems, pages 106–114, May 2007.

[18] M. S. Ganeshmurthy and G. R. Suresh. Path planning algorithm for autonomous mobile robot in dynamic environment. In 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), pages 1–6, March 2015.

[19] Jean Gregoire, Michal Čáp, and Emilio Frazzoli. Locally-optimal multi-robot navigation under delaying disturbances using homotopy constraints. Autonomous Robots, 42(4):895–907, Apr 2018.

[20] Dongjun Hyun, Hyun Seok Yang, Gyung Hwan Yuk, and H. S. Park. A dead reckoning sensor system and a tracking algorithm for mobile robots. In 2009 IEEE International Conference on Mechatronics, pages 1–6, April 2009.

[21] R. Jarvis, N. Ho, and J. Byrne. Autonomous robot navigation in cyber and real-worlds. In Cyberworlds, 2007. CW '07. International Conference on, pages 66–73, Oct 2007.

[22] Eunwoo Kim, Sungjoon Choi, and Songhwai Oh. Structured kernel subspace learning for autonomous robot navigation. Sensors, 18(2), 2018.

[23] T. Li, S. Sun, and T. P. Sattar. Adapting sample size in particle filters through kld-resampling. Electronics Letters, 49(12):740–742, June 2013.

[24] Xiang Liu and Daoxiong Gong. A comparative study of a-star algorithms for search and rescue in perfect maze. In 2011 International Conference on Electric Information and Control Engineering, pages 24–27, April 2011.

[25] T. W. Manikas, K. Ashenayi, and R. L. Wainwright. Genetic algorithms for autonomous robot navigation. IEEE Instrumentation Measurement Magazine, 10(6):26–31, December 2007.

[26] M. Mansouri, H. Nounou, and M. Nounou. Kullback-leibler divergence -based improved particle filter. In 2014 IEEE 11th International Multi-Conference on Systems, Signals Devices (SSD14), pages 1–6, Feb 2014.

[27] Katja Mombaur, C. Javier Gonzalez, and Martin L. Felis. Towards a better understanding of stability in human walking using model-based optimal control and experimental data. In Jaime Ibáñez, José González-Vargas, José María Azorín, Metin Akay, and José Luis Pons, editors, Converging Clinical and Engineering Research on Neurorehabilitation II, pages 273–277, Cham, 2017. Springer International Publishing.

[28] Katja Mombaur, Anh Truong, and Jean-Paul Laumond. From human to humanoid locomotion—an inverse optimal control approach. Autonomous Robots, 28(3):369–383, Apr 2010.

[29] T. Nomura, T. Kanda, T. Suzuki, and K. Kato. Prediction of human behavior in human–robot interaction using psychological scales for anxiety and negative attitudes toward robots. IEEE Transactions on Robotics, 24(2):442–451, April 2008.

[30] Alessandro Vittorio Papadopoulos, Luca Bascetta, and Gianni Ferretti. Generation of human walking paths. Autonomous Robots, 40(1):59–75, Jan 2016.

[31] M. Ravangard. Fuzzy behavior based mobile robot navigation. In 2015 4th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), pages 1–7, Sept 2015.

[32] Einar Snorrason. Robot Localization in Dynamic Environments. Master's thesis, KTH Royal Institute of Technology, 2015.

[33] R. Sugawara, T. Wada, J. Liu, and Z. Wang. Walking characteristics extraction and behavior patterns estimation by using similarity with human motion map. In 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 2047–2052, Dec 2015.

[34] Daniela TarniŢă, IonuŢ Geonea, Alin Petcu, and DănuŢ-Nicolae TarniŢă. Experimental characterization of human walking on stairs applied to humanoid dynamics. In Aleksandar Rodić and Theodor Borangiu, editors, Advances in Robot Design and Intelligent Control, pages 293–301, Cham, 2017. Springer International Publishing.