Master Thesis

Sebastian Meyer zu Borgsen

Incorporating Articulated Scene Models in Movement Strategies for a Robot Companion
Sebastian Meyer zu Borgsen

Incorporating Articulated Scene Models in Movement Strategies for a Robot Companion

Master Thesis submitted as part of the Master’s examination
in the graduate program Intelligent Systems
of the Faculty of Technology
of Bielefeld University

1st Assessor: PD Dr.-Ing. Sven Wachsmuth
2nd Assessor: Leon Ziegler

Bielefeld, August 2014
Abstract

At this time most of the mobile robots stay in laboratories. More and more they start leaving this controlled and often specially for them designed environment. This includes sharing their operation space with humans and other autonomous agents. In consequence the mobile companion has to be able to react dynamically to changes applied to its environment. Instead of only finding a way around the obstacle the mobile companion becomes enabled to solve such situation with cooperation.

The next step for a higher degree of autonomy in the future of robots is the ability to detect and clearance of obstacles by itself. Robots have to be enabled to separate unscalable walls from articulated obstacles in the scene to navigate self-determined in their environment. Thus navigation can be treated as a complex behavior involving the complete robot and not just as a task for a single component. This work enables the robot to perceive its world as a whole and thus remove everyday barriers.
# Table of Contents

1 Introduction 1
   1.1 Outline ........................................... 2
   1.2 Motivation ........................................ 3
   1.3 Objective ........................................ 3

2 Related Work 5
   2.1 Situation and People Awareness ..................... 5
   2.2 Exception Detection .............................. 7
   2.3 Mapping and Navigation ............................ 8

3 Essentials 10
   3.1 BIRON - The Bielefeld Robot Companion .............. 10
   3.2 ASM - Articulated Scene Model ...................... 12
   3.3 BonSAI - A Robot Behavior Modeling Framework ........ 14
   3.4 PCL - Point Cloud Library .......................... 15
   3.5 SeAM - Semantic Annotation Mapping .................. 15
   3.6 ObjectBuilder - Person Tracking .................... 16
   3.7 RSB - Robotics Service Bus ........................ 16
   3.8 ROS - Navigation Stack ............................ 17

4 Framework Details 19
   4.1 Example Situation - Path Blocked .................... 19
   4.2 General Structure and System integration ............ 22
   4.3 Pattern Matching ................................. 24
      4.3.1 Matching Structure ............................. 24
      4.3.2 Communication Pattern and Barriers ............ 26
      4.3.3 Implementation ............................... 28
# Table of Contents

4.3.4 Application ........................................ 31  
4.4 ASBP - Fusion of Navigation and ASM .................. 32  
4.4.1 Processing Steps ................................... 32  
4.4.2 Implementation .................................... 37  
4.5 Designed Strategy ..................................... 38  
4.5.1 Behavior Structure ................................. 38  
4.5.2 Analyze Scene Skill ............................... 39  

5 Evaluation ........................................... 41  
5.1 Pattern Matching ..................................... 42  
5.2 ASBP Component ...................................... 43  
5.3 Framework on the Robot ............................... 47  
5.4 Discussion of the Results ............................. 53  

6 Conclusion and Outlook ................................. 54  

Appendices ............................................. 56  
.1 Graphics .............................................. 57  
.2 Listings ................................................ 59  
.3 Statement of Authorship .............................. 60  

Literature ............................................... 61  

List of Figures .......................................... 64  

List of Tables and Listings .............................. 65
1 Introduction

At the present time most of the mobile robots reside in laboratories. More and more they start leaving this controlled environment which is often specifically designed for them. One important reason robots are needed in the future is the demographic change within society. This creates the difficult task to make sure there is enough care for the elderly people. Robots have to provide an important part for such tasks in the future. This includes sharing their operation space with humans and other autonomous agents. All these agents might manipulate objects and doors without informing the robot about these changes. In consequence the robot has to be able to react dynamically to changes applied to its habitat.

Especially robot companions and service robots are expected to find their way around in domestic areas. There already has been major progress in the field of mapping the environment for a mobile robot. Such maps enable modern robots to autonomously plan a path and navigate to a given target. Even small changes to the environment can be updated on the map over time.

The next step for a higher degree of autonomy in the future of robots is the ability of detection and clearance of obstacles by itself. When robots join people in their living space it is inevitable that things might change their location. Especially with children around robots have to expect things lying on the floor. In elder care where the robot might help people suffering from dementia placed in unusual locations without them remembering about that. These can be objects of almost any size and shape. Some of the objects might be easy to drive around but some have to be moved to continue. The most common obstacle a robot might
encounter in domestic environments is likely a closed door. The case of a closed door will be examined in this thesis as an example for scenarios where the robot encounters dynamic objects. Basis for the manipulation of such barriers is the ability to perceive them as such parts of the scene, named articulated objects. Robots have to be enabled to separate unscalable walls from articulated obstacles in the scene to navigate self-determined in their environment.

Adding such new behavior to an already existing robot companion demands for special treatment. Available functionalities have to be kept intact and only minimal changes to other parts of the system should be applied. This approach results in a general solution that can be applied to different kinds of robots that meet a set of requirements. Therefore the information exchange of already existing software will be improved to reach a sophisticated robot system. An important step to achieve improved and integrated systems is to harmonize interfaces in the system. This enables parts of the system to retrieve information from other components in order to enhance their capabilities. Thus navigation for example can be treated as a complex behavior that involves multiple components and not just as a task for a single component.

1.1 Outline

This master thesis starts with explaining the incitements for this work and gives an overview over the objectives of this work. Therefore sections 1.2 and 1.3 explain the difficulties involved with integrating new features into an existing robot companion and what challenges have to be overcome in a system that involves a multitude of components. Chapter 2 introduces previous research in this field. That includes situation and people awareness and the detection of exceptional cases. For the scenario of applying this work to a cooperative navigation trough a door an overview of state-of-the-art navigation and mapping is given. As such a complex task involves multiple components the ones that are involved
1.2 Motivation

in this scenario are explained in chapter 3, starting off with introducing the robot companion BIRON which was utilized in this thesis.

After detailing the example situation in section 4.1 the general structure and features of the extensions will be explained. The following sections describe the three new parts created for this thesis. Section 4.2 discusses the general scheme of the project and how to integrate it into the existing robot system. Pattern matching which was designed to react on communication patterns with a defined behavior is described in section 4.3. In section 4.4 a new component which was designed to fuse information from the articulated scene model and the navigation component is presented. A behavior that gets applied to the example situation and solves the situation by cooperating with a person is described in 4.5. Chapter 5 gives an overview how the system performed in the robots simulation and in the real world. The thesis closes with a conclusion and an outlook.

1.2 Motivation

A lot of component based robotic systems suffer from communication between the components and thus do not use information from multiple components to solve a task efficiently. State machines that control the robots’ behavior are often too static and cannot handle all kinds of exceptions. This work enables the robot to perceive its world as a whole and therefore remove everyday barriers. A part of this development in the field of robotics will be described in this thesis and approved in practice.

1.3 Objective

A big concern on robot systems that consist of a multitude of components is the integration of the generated information. A lot of approaches separate between sensors which generate information and actuators that can be used to generate behavior with little or none communication between
those components. The main target of this work is the incorporation of multiple sensors to solve more complex tasks. These tasks can be e.g. navigating through closed doors or cooperating with people to clear objects that are blocking the path of the mobile robot companion.

In most of the robotic systems the navigation component sticks out of the pattern of loosely coupled components. The laser range finder as direct sensor input and motor actuation are coupled tightly. Distance information directly generates a reactionary behavior of the robot platform without much reasoning or scene understanding. This is one of the challenges to overcome.

At first the robot has to be enabled to detect a situation that defers from a standard task. The created model will be applied and demonstrated with suddenly occurring problems during a navigation task. The difficulty of this task is the fact that a lot of component based robots use a navigation actuator that tries to reach the goal all by itself, without informing the system about problems during the process of reaching the goal. After detecting a situation that may be solved in a more intelligent way by incorporating additional scene knowledge, a chain of accumulating additional information has to be triggered. This scene knowledge may consist of nearby people or information about articulated objects in the scene. A behavior has to be designed that solves such situations.
2 Related Work

As of today mobile robots are already able to solve a variety of tasks. Especially domestic service robots incorporate a large set of software components to show more and more complex behavior. As this thesis targets the advancement of simple map based navigation to a more sophisticated informed goal reaching behavior, a selection of already existing projects had to be intertwined as well as new parts had to be designed.

This section introduces already existing techniques that are closely related or had influence on this work. A brief relation to standard navigation and path finding algorithms in changing environments will be given. Methods that induce a situation and people aware behavior will be discussed as well as techniques that deal with detection of exceptions in a multi component robot system. Some of the incorporated components used in this work will be described in more detail in section 3.

2.1 Situation and People Awareness

If robots should be able to handle complex tasks autonomously, there has to be some kind of planning involved. Planning always requires the system to have the ability to perceive the current world state and then select the correct actions to get to the target world state. This can be achieved by generating a symbolic representation of current and goal states. Improving the planning by attaching semantic knowledge to a planner as external reasoning systems results in a more efficient way of planning. [DEK+12] It remains the problem of generating a correct world state and formalizing the goal in a symbolic representation.
A prerequisite for planning is an accurate world model. Especially for movement and manipulating tasks a model of the movement space objects becomes more and more important. Sturm et al. present a method to extract articulated models from 3D point clouds from cabinet doors and drawers. [SKSB10] These models can be used for manipulating and moving such objects. The presented algorithm starts with detecting planes and tracks the movements over a series of input clouds. Generative models can be learned for different objects to estimate the type of articulation. Furthermore predictions can be made about not yet observed configurations to allow a manipulation by a robot.

Ziegler et al. presented an informed search behavior for a service robot that uses a semantic map to fuse information of multiple sensors. [ZSKW10] This work showed that enriching a task with more information can have a severe impact on the quality and speed of the solution. Inspiring for this work was not only the mapping part but as well the possibility of generating viewpoints. These viewpoints allow the robot to reposition itself in order to have a good view on the scene to improve its knowledge about it. As well this can be used to position the robot relative to a marked region on the semantic map.

A lot of tasks for a robot companion incorporate the cooperation with a human. An essential requirement for cooperation and interaction with a person is the detection of people surrounding the robot. This information especially is vital for navigation tasks as these have to calculate movements in the environment which might be hard to predict, e.g. people that are moving around quickly, start and stop moving or change direction suddenly. A common technique for detecting people close by the robot is the detection and tracking of legs in the data generated by a laser range finder. [AGLB08, SBFC03] This has mostly practical reasons, as most mobile robot platforms are equipped with a laser range finder for navigation purpose as stated in section 2.3. Stückler et al. propose a method that helps increasing the people awareness of a service robot by utilizing the semantics of the robot’s surroundings. [SB11] Their work shows that enriching a part of the system that is designed to detect people
by information from other components like a furniture detection can improve the accuracy of such component. This leads to the conclusion that there are more cases where inter component communication and information exchange can improve the performance of individual components and thus the overall performance of the whole system.

With the information about surrounding persons the robot can either directly enhance its navigation capabilities by retaining bigger distances to such moving obstacles or by actively cooperating with persons to solve navigation tasks the robot could not solve on its own. Such interaction may increase the need for repositioning the robot multiple times. Getting the attention of a person [HSEGT06] may require the robot to move itself or at least turn into direction of that person. Most important for cooperative navigation tasks is to position itself in a way that allows its helper to move the obstacle e.g. open the door without getting blocked by the robot.

2.2 Exception Detection

Algorithms that detect system anomalies by analyzing the information flow in a robotic system [GWHM11] enable robots or their engineers to detect an exceptional state of the machine. Such system requires a robot with an event-based architecture that passes messages between components. Then a component can apply learning algorithms to generate a model of the standard data flow in such a system. Whenever the data flow changes significantly this component triggers an alarm.

This idea was inspiring to design a software that is able to detect special patterns in the communication flow of the robot to trigger not only an alarm but initiate a behavior that gathers more information for the given problematic situation to solve it in a more sophisticated way. Processing every kind of information with each component at any time may result in a system overload. Thus detecting a situation which really needs additional information is beneficial.
If an exception was detected but no solution could be found it is always beneficial to state the problem. [GKE+10] The idea relies on the fact that even if a stated task is not solvable for a robot it still might be possible to state the reason. A cost function is used to determine the quality of an excuse. A good excuse with minimal costs can be considered a perfect excuse.

2.3 Mapping and Navigation

Most of the established mapping and navigation algorithms [RS04, ST05] expect a quite static environment. These algorithms utilize odometry calculations and laser range finders on a mobile robot to simultaneously localize themselves and create a map (SLAM) of the environment. By moving around in the environment a map is incrementally generated and can be updated over time. The map consists of a grid that stores a probability in each cell for either free or blocked space.

Another aspect is the navigation on such maps. The most common techniques split this task into planning a global plan on a map to the goal and a local motion planner that keeps the robot on track. [GD00] The task of planning a path on existing maps is solved for some time now. The A* algorithm finds a collision free path to the goal by assigning a cost value to each field in the grid. [KD86] Reaction to suddenly arising obstacles on the planned path can be achieved by the dynamic window approach which plans the motion of the robot only in the near future or
even just the next step. Even by looking further in the future [SB02] not all situations can be dealt with.

A big concern with all those algorithms is the lack of using deeper scene knowledge to reach a given goal. This makes it almost impossible to find the ideal solution. In case of a scenario where the robot has no way to detect that its path is only temporary blocked by a door or a group of people, it is impossible to find a solution.
3 Essentials

Generating complex robot behavior requires the utilization of a multitude of software and libraries. Even for a minimal demonstration setup there are at least twenty different software components running at the same time. This chapter will introduce the mobile robot platform BIRON which is used for development as well as give an introduction into the most important components for the scenario.

3.1 BIRON - The Bielefeld Robot Companion

The Biron platform was designed by a group of scientists from the Faculty of Technology of the University of Bielefeld. [HHH+04] The current version [ZWMW14] is based on a platform from MobileRobots\(^1\) named GuiaBot. It is about 59cm long and 48cm wide with an approximate weight of 45 kilograms. Rotations of the robot can be performed with over 300 degree per second. Forward and backward movement is possible with 1.7 meters per second. This platform has a differential drive and thus the robot is able to turn in place and drive forward and backward. It is equipped with a SICK LMS laser range finder for navigation, obstacle avoidance and people leg tracking.

The sensor-head features two PrimeSense RGBD sensors for depth and video perception, a high resolution DSLR camera for object recognition, a wide angle video camera for face recognition and a microphone for speech input. A front mounted display and speaker provide feedback for human robot interaction.

\(^{1}\)http://www.mobilerobots.com/
3.1 BIRON - The Bielefeld Robot Companion

Figure 3.1: BIRON - mobile robot platform used by the ToBi robocup team in the @home league. Some of the sensors are shown in a larger image on the right.

For basic manipulation tasks Biron has a 5 degrees-of-freedom Katana arm mounted to its body. All computation is done on two workstation piggyback laptops. The components are evenly distributed on both PCs for a balanced workload. Ethernet allows communication between components on different PCs.

This robot is used by the Team of Bielefeld (ToBI) since 2009 in the Robocup@home competition. Features changed over the years to keep the robot competitive.
3.2 ASM - Articulated Scene Model

The Articulated Scene Model (ASM) [SBWK10, BSKW11] is a system that analyzes 3D data of a scene to extract essential knowledge about it. The ASM divides the scene into three parts.

“Definition 1. (Articulated scene mode):
- Static scene (Never changing parts)
- Moving entities (e.g. humans or robots)
- Movable objects (e.g. chairs, doors)” [BSKW11, p.1]

(a) Picture of example scene

(b) ASM result as depth image

Figure 3.2: A scene which contains a chair and a poster container as movable objects get analyzed by the Articulated Scene Model component.
This information is generated from a scene by observing it for some time. The algorithm updates its scene model with each new 3D frame received from the environment. The main idea of this system is to permanently update the background model which makes it possible to infer the other parts of the scene. Background information is updated by assuming that the background is defined by the farthest measurement.

Figure 3.2a shows an example scene that has a wall as background. An office chair and a cylindrical object are parts of the scene as well. After presenting these objects to ASM by moving them in the field of view of the camera in such a way that allows the sensor to sense the background, it is able to generate a model that separates both the objects from the background. These objects are colored blue and the static environment is marked gray as shown in figure 3.2b.

“Definition 2. (Vista Space):

The vista space is part of the world, which can be viewed at the same moment, only be slightly moving the gaze.”

[BSKW11, p.2]

The algorithm works in the vista space. Thus it assumes that the 3D-Sensor is not moving. An extension by Ziegler et al. [ZSW13] enables the robot to generate and update the scene model not only from a single static viewpoint but from multiple different locations. This helps to continuously increase the knowledge about its surrounding by adding and fusing information from multiple different viewpoints to an exhaustive world model. The fusion is performed by saving a model of the current perception. The model consists of the extracted scene background and the robot’s position to keep a spatial relationship of already perceived information. Every time the algorithm is triggered it searches its database for a scene that was recorded nearby the robot’s current position. This scene is then transformed in the field of view applying the difference in position and orientation of the robot. As the robot position might not be exact, various correction steps can be applied, for example feature extraction and fitting between the cloud from the database and the current view.
3.3 BonSAI - A Robot Behavior Modeling Framework

BonSAI is a framework that acts as an abstraction layer to the hard- and software components, as well as it allows the execution and management of finite state machines. [Sie13, SZKW14, LSW14]

The Sensor and Actuator Interface provides abstraction from the hard- and software implementation. This interface provides access to both abstract as well as real hard- and software sensor and actuators through a java API. Sensors and Actuators can be added to a project by configuring them in a BonsaiConfiguration file. All configuration parameters can be directly provided by defining the values with Extensible Markup Language (XML) [BPSM+98]. XML makes these files easy to read and edit for the developer. Possible options for sensors and actuators can be e.g. information about which communication middleware should be used and other component specific options.

![BonSAI Layers - visualization of the abstraction layer provided by BonSAI.](image)

**Figure 3.3:** BonSAI Layers - visualization of the abstraction layer provided by BonSAI. This abstraction enables the developer to design hardware and software component independent behavior.

As can be seen in figure 3.3 BonSAI not only abstracts from functional and hardware components but also allows to create and use Behavior Modules. These behaviors are a set of concatenated skills, which represent...
the smallest unit in behavior design. Skills are directly implemented in Java to solve only one part of a task. This guarantees a high rate of reusability thus preventing code duplication. All defined skills can then be bundled and connected into a more complex behavior with a State Chart XML (SCXML) [BAA+07]. Multiple behaviors can be combined to a complete solution for a behavior that completely solves a task.

3.4 PCL - Point Cloud Library

The Point Cloud Library\(^2\) [RC11] is a framework for n-dimensional point clouds and especially 3D processing. This project was started by Willow Garage and is now continued as open source. This library includes a lot of state-of-the-art algorithms for filtering, surface reconstruction and segmentation. It is written in and for the programming language CPP. Features of this library were used in multiple parts of the system, for example in the components for object recognition and in the articulated scene model for merging scene from multiple different viewpoints. For this work features of this library were used, for example for visualization, path mapping, calculation of object hulls and projection onto the ground plane.

3.5 SeAM - Semantic Annotation Mapping

The Semantic Annotation Mapping (SeAM) software, which was approached in the master thesis of Leon Ziegler, is able to attach semantic information to the robot’s map. [Zie10]

This software creates a new layer for each kind of semantic information in the mapping system. Each map layer gets assigned to an abstract sensor that provides data for this layer. Such sensors can be of different types but should always deliver regions or points of interest, for example provided as a list of hulls in egocentric coordinates. The information

\(^2\)http://pointclouds.org/
is then stored as a probability for each cell. Each time this component perceives information for a given region, the probability value is raised by a developer defined value or function. The contrary happens if no information is provided for a region that is in the field of view of the robot. This ensures removal of outdated information over time. All layers are based on the SLAM map, thus having same dimensions. Though the resolution can vary for different kinds of information.

In this work this component will be used to keep track of areas in which objects are manipulated. This helps the robot to make use of this information by obtaining a position next to manipulation areas.

### 3.6 ObjectBuilder - Person Tracking

For human robot interaction robots need to be able to detect and track people around them. This is required to know to whom the robot can speak to or from whom to receive commands. As the BIRON platform is limited in terms of manipulation abilities due to the power of its arm, interaction and cooperation play an important role for solving tasks.

The ObjectBuilder component on the BIRON system is able to use multiple sources of information to combine them to one combined person hypothesis. Possible sources can be complete skeletons from a body detector, faces from face detection or legs from a laser range finder based leg detection. Section 2.1 already stated that leg detection is a reliable and fast way for tracking people. Thus primarily information from the leg detection component is fed into the ObjectBuilder to generate hypothesis of persons located around the robot.

### 3.7 RSB - Robotics Service Bus

Creating a complex behavior in most cases requires involving a lot of different components. Important is having them work together instead of side by side. The Robotics Service Bus (RSB) [WW11] is a middle-
ware that handles exchange of information between as well as within components. Components can subscribe to and publish information to the system. The exchange is message-oriented and event-driven. *Robotic Service Types* (RST) is a collection of data/message types that provide a base of common types of information that can be understood by the whole system. For reading a RST the component has to add a RSB converter (RSC). These converters provide the needed information on how to convert an incoming event to a usable data format. If the system receives a message of type for which no converter is registered, an exception is thrown. New data types can be easily defined with the *Google Protocol Buffer*\(^3\) format. Messages automatically get transported to all subscribed components, even to different computers and networks. RSB provides interfaces for the most common programming languages like C++, Java, Python and Common Lisp.

A lot of components on the BIRON system mentioned in this work had to be adapted to the RSB middleware to achieve a well integrated and consistent system. Most of them were still using the now outdated middleware XCF\([WFBS04]\) which is an XML based communication framework.

### 3.8 ROS - Navigation Stack

Lately the *Robot Operating System* (ROS) \([QCG^{+}09]\) becomes more and more popular in the robotics scene. It supports a broad spectrum of hardware in terms of drivers, interfaces and documentation.

BIRON primarily makes use of ROS for its navigation stack. This means mapping and navigation is performed by components of the ROS system. The navigation stack exposes its features to the rest of the system by a bridge that translates ROS messages into RSB messages and vice versa. The used navigation stack consists of multiple layers. Lowest layer is the controller on the platform itself. This controller is able to per-

\(^3\)https://developers.google.com/protocol-buffers/
form only basic tasks like driving a given distance, turning by a defined angle and driving at a constant speed. These features are provided to the system by the MobileRobots’ Advanced Robot Interface for Applications (ARIA)⁴ driver and made accessible to ROS by the ROSARIA⁵ software. On top of that run two planning components. The DWA-local-planner⁶ provides basic planning capabilities, that only calculates the very next velocity commands that are necessary to stay on track. NavfnROS⁷ is in charge of planning paths over bigger distances. It uses the Dijkstra-Algorithm to calculate the cost-wise cheapest path to a given target which is received by the RSB bridge mentioned before. The planning incorporates the already generated environment map and the current laser range data to generate a valid plan across the free space. A down side of such a complex stack is the limited ability to interact with it while it works on reaching the target location.

⁴http://robots.mobilerobots.com/wiki/ARIA
⁵http://wiki.ros.org/ROSARIA
⁶http://wiki.ros.org/dwa_local_planner
⁷http://wiki.ros.org/navfn
4 Framework Details

As a complex behavior cannot be achieved with a single component, a whole framework of components had to be designed. Most of the used components were already explained in chapter 3. The newly created components will follow in this chapter. This chapter starts off with giving an example scenario in section 4.1. Although the given situation makes use of applying information from the articulated scene model in a navigation situation, it is only one example for applying such components. During the design phase it was kept in mind to keep the software as generally applicable as possible. The challenge of integrating the framework into an existing system had to be overcome by creating interfaces and choosing a design that fits the systems architecture.

4.1 Example Situation - Path Blocked

Reaching a goal with a mobile robot like BIRON already consists of multiple steps. Figure 4.1 gives an overview on how to improve the movement of a mobile robot. Therefore, figure 4.1a visualizes the classic steps (see section 3.8) performed to reach a target position on the map. Right after setting a goal, the navigation component calculates the best path on the map to reach the particular goal. Then another part of this component generates local trajectories by varying motor speeds that keep the robot near the planned path without colliding with an obstacle. This is achieved by simulating sometime in the future where the robot is located after applying a given velocity to the wheels. All local paths which collide with the environment will be rejected and the one that is closest to the planned path as well as close as possible to the goal
will be applied. With these steps repeatedly applied the robots position will converge to the goal. This successful sequence is marked with black arrows in the figure.

![Diagram of navigation sequence](image)

(a) Classic Navigation  
(b) Improved Navigation

**Figure 4.1:** Visualization of the robots navigation sequence. The left graph shows the classic behavior and the right one shows how this could be optimized to have a higher success rate. Both graphs include a successful goal reaching with black arrows. Orange arrows mark unsuccessful cases and green marks extensions of this thesis.

If the robot approaches a situation where it is not able to generate a local trajectory near the global path without colliding with something, the component triggers to plan a new global path whilst taking the obstacle into account. This case could appear if a robot approaches a closed door and then plans a new global path through another door. If the navigation program is not able to plan a new global path, considering the freshly detected obstacle, it can only cancel to reach the goal.

A typical situation for a navigation problem on a mobile robot is when something is blocking its planned path. Figure 4.2 shows such a situation
in which the robot planned a path that is blocked by a closed door. The robot is marked with a red dot while its goal is marked with a green one. A yellow dashed line marks the path planned by the navigation component to reach the goal. The closed door in front of the goal could not be considered while creating a plan because it was hidden behind a number of tables. These tables prevent the laser range finder from covering the whole area. The figure shows the laser’s area of coverage in blue.

The aim of this thesis is to react to the case where the mobile robot companion tries to generate a global plan for the second time to prevent or reduce the number of erroneous cases. The green elements of figure 4.1b show that analyzing a scene and applying a fitting strategy to a certain situation lead to a higher success at reaching goals.

**Figure 4.2:** Top view of the robots environment in a navigation situation. The robot (red dot) plans a path (yellow dashed) to its goal (green dot). With its laser range finder he covers the blue area, thus he is not able to sense that its planned path is blocked by a door (red).
4.2 General Structure and System integration

The goal is to achieve a robot behavior that is able to solve navigation problems in a more sophisticated way. Adding additional information about a scene to a complex situation shall lead to a better solution or provide a solution for a situation that would not be solvable without that information. The Articulated Scene Model is able to provide information on which parts of a system are manipulable and thus are barriers that can be removed and then passed. After the analysis of the stated problem, three additional system components were designed. One component that detects special situations and triggers a specifically designed behavior is the second component of this thesis. As a third component a software was created that incorporates multiple sources of information and reprocess the data, relevant for the given situation.

Section 4.1 already described the default way the ROS navigation component works. The problematic of interacting and intervening in the navigation process of this software is described in section 3.8. Another problem is that this component is third-party software what makes it harder to add modifications. Besides this software is used on robots all over the world like that. This makes it even more desirable to create a framework that is compatible to this software without modifying it. At first a component had to be designed that is able to detect the urge for action. This component had to be monitoring based to be as independent as possible. Figure 4.3 gives an overview of movements of a mobile robot and how they can be handled more efficiently with additional scene knowledge noted in BPMN [Whi04].

An additional component that controls the robot’s behavior while executing the selected problem solving strategy is designed as finite state machine. It controls the gathering of additional information through the BonSAI framework as well as the behavior of the mobile robot companion in a complex navigation task.

The information fusion combines information from the articulated scene model and planned paths by the navigation stack. It intersects the paths
Figure 4.3: This graphic illustrates some problems that might occur during operation of a mobile robot companion. A very typical interruption is the replanning of a path because the original planned path is discovered as blocked while moving along. This BPMN plan shows a simplified reaction chain that could be applied by the robot to handle such situation.
4.3 Pattern Matching

The motivation to design a component that detects a situation that needs special treatment by observing the information flow in a robotic system arose because not all components state precisely if problems occur. In fact all components have in common that they interact with the system by sending different types of messages with varying content. If the robot companion does not act in the expected way the developer is often able to identify a special sequence of messages flowing in the system. Different kind of problems and situations most of the time have recurring patterns of messages. This sequence of events can then be expressed as a processable Communication Pattern. All this can be done without modifying the system in any way. One prerequisite for such component is that all components that shall be covered by the monitoring process send messages that are readable by the monitoring application.

4.3.1 Matching Structure

The pattern matching listens to predefined communication channels in the system. This can include either all messages or a cut down set to reduce the computation overhead. In addition to the configured listening sources, the patterns to wait for and what has to be performed when they fire has to be defined. Figure 4.4 visualizes the position of the pattern matching in the system context. It can be seen that this component is not affecting the system at all while performing the pattern matching. Only after a given pattern gets triggered, the matching starts to interact with the system by executing a defined behavior.

Each communication pattern consists of a list of so called barriers. Barriers explicitly define what kind of characteristics a message has to
4.3 Pattern Matching

Figure 4.4: A robot companion consists of many software components that emit their generated information to the system or process information of other components. These messages get analyzed by a pattern matching component that is able to influence the state machine by pausing and injecting a special behavior for a difficult situation.

deliver to pass this barrier. To bring a pattern to its final state, all barriers have to be passed.

All flowing messages get compared to the currently active barriers of each pattern in the component. If a message fits a pattern’s barrier, the internal state of the pattern changes to either the next barrier or to its final state. When reaching the final state, a pattern triggers the pause of all current ongoing behaviors of the robot. After pausing the execution of a state machine, which is responsible for the robot’s behavior, an interposed behavior gets executed. At the end the original behavior will be continued.

The figure 4.5 illustrates how messages modify the state of patterns by passing barriers. On top is a set of components that emit a variety of messages. These messages get stacked in order of creation time and then
are forwarded one by one to perform the actual matching. Each message contains at least a creation time, a scope, the type of the content and the actual content represented as key value pairs.

In this example there are two different patterns with one of them in different states, as if a pattern gets activated by overcoming its first barrier a new instance of that pattern is created. A newly created pattern instance always starts in the hibernated state which means that yet no barrier is crossed. If at least one barrier is crossed the pattern has an active state until it reaches either its final state by passing its last barrier or gets canceled by reaching a deactivator barrier. The canceled state resets a pattern to its first barrier.

### 4.3.2 Communication Pattern and Barriers

A communication pattern always consists of an identifier, a list of barriers and a reference to a behavior that shall be triggered by this behavior. The identifier should be unique to prevent the system to detect it as duplicate and thus remove it from the list of active patterns.

A barrier can either be a simple condition that can be directly compared against an incoming message or it can be a complex barrier that holds a whole list of barriers. The simple conditions check whether scope, type of message and a list of key value pairs is contained in a message. For the complex barriers there are again two types: The `and` and `or` link. Where the `or` holds a list of barriers only one of them has to match to pass the whole `or` barrier. The `and` barrier expects all its contained barriers to be matched until this barrier is passed. While the whole communication pattern demands an order of the message the `and` does not. In conjunction those barriers can produce complex patterns.

A pattern’s state can be one of the five listed in table 4.1. From each type of pattern there is always one pattern in the hibernate state. As when the pattern becomes one of the other four states a copy of this pattern will be created and reset to the hibernate state. Patterns that become finalized or canceled will be removed from the set in the next
4.3 Pattern Matching

**Figure 4.5:** Illustration of how an incoming message modifies the state of a set of patterns. For demonstration there are two different patterns. Only pattern 1 is shown in two different states. Messages get stacked up in the component by other components that emit them during their life-cycle. The matching is then performed message by message against all active barriers.
4.3 Pattern Matching

cycle. These states exist solely for the logging purpose as they cannot pass any barriers or fire afterwards. The activated state marks the point of time at which a pattern just got its first barrier passed. This state differs from running as in this state the hibernated copy of the pattern will be created.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIBERNATE</td>
<td>No barrier is passed yet</td>
</tr>
<tr>
<td>ACTIVATED</td>
<td>Transit from hibernate to running</td>
</tr>
<tr>
<td>RUNNING</td>
<td>At least one barrier passed</td>
</tr>
<tr>
<td>CANCELED</td>
<td>Canceled by timeout or deactivator barrier</td>
</tr>
<tr>
<td>FINALIZED</td>
<td>Last barrier of a pattern got passed</td>
</tr>
</tbody>
</table>

Table 4.1: The five states a pattern can reach during its lifecycle. Activated, canceled and finalized patterns exist only one cycle. After that they get either activated or removed from the set of patterns.

4.3.3 Implementation

The pattern matching is implemented with the Java programming language. Patterns can be defined with the JavaScript Object Notation (json)\(^1\). Generating java objects from the json files are handled by features of the Gson\(^2\) library. The message input comes from an event based asynchronous RSB listener with buffering.

The implementation of the pattern structure is realized with five classes as can be seen in figure 4.1. Everything described in section 4.3.2 is transferred into the class structure. The \textit{and} and \textit{or} barriers both extend the link class which prevents code duplication as both of them hold a set of barriers as well as they are barriers themselves. This allows these links to contain instances from links again to allow patterns of any complexity. All classes that extend the Barrier class are required to implement their

\(^1\)http://tools.ietf.org/html/rfc7159 [Ihr13]
\(^2\)https://code.google.com/p/google-gson/
4.3 Pattern Matching

Listing 4.1: Example pattern for the matching component. This contains only one barrier that waits for a state change to a specific skill in the state machine.

```json
{
  "identifier": "example",
  "barriers": [
    {
      "$type": "Condition",
      "scope": "/bonsai/transition/",
      "msgType": "StateChange",
      "kvp": {
        "toState": "navigation.DriveDirectly"
      }
    },
    "action": ".//doSth.xml",
    "config": "./doSthConfig.xml"
  ]
}
```

Patterns are defined in JSON. With starting this application all pattern files from a user-defined folder will be parsed and loaded with help of the gson library. Listing 4.1 shows a simple example pattern. The identifier of this pattern is “example”. The list of barriers contains only one barrier which has the type “condition”. The condition type directly compares its scope and the type of message with the incoming messages. The list of key value pairs contains one element with the key “toState” and the value “navigation.DriveDirectly”. This means that this barrier is crossed when the state machine changes its state to a skill with the name “navigation.DriveDirectly”. When this pattern fires it will execute the “doSth.xml” behavior with the “doSthConfig.xml” configuration.
4.3 Pattern Matching

All pattern instances are managed by one central preprocessing class. This way the system can prevent duplicates of patterns, executing multiple behaviors at once and make sure that there is always at least one pattern of each type in hibernate state. Guaranteeing thread safety is provided by protecting critical regions with the java synchronized mechanisms. As the matching itself is directly performed by the pattern, respectively their barriers, the controller class only has to check the resulting state of a pattern after forwarding a message to it. Interfaces to the system’s middleware are provided by message or event forwarders which are able to pick up messages from the system and unify them for being processed by the central processing class. As this component does not only monitor but at some point has to execute some kind of behavior, a client for bon-sais state machine was created that is able to pause currently running state machines, continue them and inject events as well as execute complete behaviors. Another extension of the BonSAI state machine was the integration of state publishing. Since the monitoring had to be enabled to directly react and match barriers to specific states of the system it was necessary to be able to inform the system about state changes in the state machine. A newly created message type for the RSB middleware now informs the system about every state change that is performed inside the state engine of BonSAI.

As stated in section 3.7 the RSB middleware expects to be able to parse all incoming events, therefore there needs to be a valid converter for each type of event that may appear registered in the converter repository. If a message is received where no converter is present, the RSB listener throws an exception and exits itself. As a monitoring component cannot be expected to have a converter for any kind of message this behavior had to be modified. A modified converter repository was created that applies a special strategy that always returns a valid converter to the system. This is achieved by returning a fitting converter if such is present and a simplified converter for all types of events that are not fully translatable but at least generates a message that contains the scope and message type. With this converter the component is restricted for some new data.
types as it cannot create valid key value pairs but always will be able to keep running and extract important information from any type of event.

### 4.3.4 Application

For the application of detecting a problem while reaching a navigation goal, a pattern that fits this case was designed. The pattern shown in listing 1 was designed with the example situation from section 4.1 in mind. The basic idea of this pattern is to watch out for a replanning of the navigation component without receiving a new goal. While a lot of patterns with a similar behavior can be created, this variant proofed to detect a navigation problem reliable without firing too often without need.

The pattern is identified as `doublePath` and gets activated by a first barrier from the condition type. It waits for a new message of the type `Path` from the navigation component on the `/ros4rsb/navigation/` scope. After crossing this barrier, the pattern activates a barrier of the type `Or`. This barrier checks all its contained barriers against an incoming message. In this case it contains two different barriers. One that equals the first barrier of this pattern except that it contains an additional timeout as well as a barrier of the `REJECTOR` type this barrier checks for a `StateChange` inside the bonsai state machines transitions to the state `nav.drive.DriveToPosition`. This ensures that this pattern gets canceled and thus returned to hibernated state with pointing to its first barrier, right after a new navigation goal was set by the state machine.

When this pattern gets finalized a special behavior is executed that enriches the navigation problem with additional information to resolve the situation. This behavior will be described in detail in chapter 4.5. While it is possible to already differentiate between different navigation problems with different pattern the problems were grouped together and then distinguish different scenarios inside the triggered behavior.
The Articulated Scene Based Planner (ASBP) component was designed to combine information from the Articulated Scene Model and the navigation component. After detecting a situation in which one of the available resolution strategies can be applied the robot has to be enabled to extract the relevant scene information to apply the corresponding strategy to be able to react accordingly. This means finding nearby persons that can be asked for help on the one hand and movable objects on the other hand. Finding persons is achieved with the component described in section 3.6, information about movable objects can be achieved with the Articulated Scene Model and paths from the navigation component can be directly observed. As all this information is not of great use considered individually they have to be brought in relation to one another. Therefore a component was created that combines information from different components to a world model that fits the task. For movement tasks, the main target is an intersection of planned paths with inflated obstacles.

4.4.1 Processing Steps

The task of intersecting paths with obstacles around the robot starts off with collecting all required data. Collecting the data has to be performed for the different types of input in parallel to ensure that only data from the same point in time is analyzed. Figure 4.6 visualizes the steps performed for extracting objects from the scene that make it impossible for the robot to follow a planned path. Possible types of input data for this component are depth images from the Articulated Scene Model for each of its three classes of objects, like movable, static and dynamic. A history of generated paths to a navigation target from the navigation stack is stored to intersect the retrieved scene with movement possibilities. In addition, the robot’s position is needed to transform all other information into an egocentric view. As software for mobile robots has to be as
efficient as possible, the scene information and paths only get converted if there is a significant change in position or angle.

Figure 4.6: Visualization of the processing steps performed by the fusing component. The three types of depth images from ASM will be converted to point clouds and finally inflated convex hulls which get intersected with the planned path of the navigation component.

3D Processing

For each incoming depth image from the ASM component this information is converted to a 3D point cloud for further computations. As next step some optimization steps are performed to reduce later computation effort. Cutting off out-of-range values and reducing the number of points
with a voxel-grid\(^3\) filter that keeps only one point in a given radius that represents the neighbors in this region ensure efficient data processing. This 3D point cloud is then transformed into the robots perspective as the sensor might not be mounted facing straight forward. Compensating the transformation the camera has relative to the robot ensures that the used coordinate system is aligned to the robot and thus to the real world. Having all objects in 3D aligned to the real world’s coordinate system allows easier visualization and analysis of spatial relationships. In addition this alignment results in axis aligned bounding boxes that refer to depth, width and height in the real world.

As the mobile robot navigates in two dimensions and its footprint can be approximated with a circle with thirty centimeter radius the problem can be reduced from 3D to 2D. This allows a map-like representation of the scenario. The current version of the Articulated Scene Model component is able to generate axis aligned bounding boxes for the objects out of the depth image. Unfortunately these bounding boxes were not precise enough for navigation purpose. Thus a new representation of articulated objects in the scene had to be designed. The point cloud representing all movable points in the scene was grouped into sub clouds with Euclidean clustering. Thus all spatial related points were used to support a single movable object in the scene. To be able to cluster not only the movable objects in the scene but also the static scene euclidean clustering could not be used. This would have resulted in too big clusters as everything would have been joined by either the floor, walls or other large objects that have points close to other objects. Normal based region growing showed to be able to generate meaningful groups of points. This way tables, walls and doors could be separated from each other and sensed as a single articulated object. It might happen that this technique results in multiple point clouds for an object. For example a table might get split into a cloud for the legs and another cloud for the tabletop. For most movement tasks this is of no concern as long as the robot avoids collision with each of the parts.

\(^3\)http://pointclouds.org/documentation/tutorials/voxel_grid.php
4.4 ASBP - Fusion of Navigation and ASM

Generating Hulls

All the point clouds that represent an object or part of one are then projected to the ground-plane. This can be done as the robot companion, like stated, moves only in 2D. A convex hull around each cluster of points is generated to circumscribe the outer limits of each object. Figure 4.7 shows an enlarged 3D view of the scene after all those steps applied. The viewer allows to visualize the results from any arbitrary position. An example from the side can be seen in 4.7b. The convex hulls are then inflated by the robots radius for easier intersection with the planned paths while preventing collisions. It is not sufficient to simply move all points from a hull by a given distance away from the center. The inflated hull has to maintain the given distance not only to each point from the hull but as well as to the connecting lines between the points.

A calculated path is received every time the navigation component calculates a new one. This can occur in two cases. Either some part of the system requested the navigation component to try to reach a new target or it tries to reach an old target with a new path. As we are only interested in the cases where a replanning occurs, the algorithm always compares the target of a received path to the last known target. If the target changes, all old paths get discarded. In the case where a new path for the same target is received, the new path is added to a list, enabling the system to intersect all possible paths to a target location with the current scene.

Intersecting Path with Scene and Publish Information

All paths that belong to the current target get transformed with the current robot position to map them in the egocentric view of the robot’s camera. With this transformation applied it can be tested whether one of them would lead into collision with an object in the field of view. All of the inflated convex hulls of movable and static objects received from the ASM get checked against the planned paths to the currently active goal. For each way-point of a path it is checked whether it is inside the
4.4 ASBP - Fusion of Navigation and ASM

Figure 4.7: Visualization from the scene of figure 3.2 in the ASBP component from two different perspectives. On the ground plane the area that is occupied by an object is visualized by a convex hull.
inflated hull. As the distance between two way-points of a path is less then a few centimeters it is sufficient to only check the points instead of calculating additional interpolated points between them.

After all information has been generated it gets published over the Robot Service Bus. It distinguishes between either static or movable and blocking or non-blocking. This results in four different types to publish. Each type is published on its own RSB scope to differentiate between them. For compatibility reasons the information is encapsulated in an XML document and then send as RST.

### 4.4.2 Implementation

The ASBP component is written in the C++ programming language. Main reason for this programming language is the ability to incorporate the PCL library which was used to implement many of the beforehand mentioned feature. Another benefit is the high performance and low resource consumption. Especially the low memory footprint of applications written in C++ is a big advantage over Java applications for example. For even better performance of the component the different streams of data are received and processed in parallel with the OpenMP framework\(^4\).

For a convenient use of the component `Boost.Program_options\(^5\)` were integrated to control various parameters such as height and angle of the used camera. Creating and managing logs is achieved with `log4cpp\(^6\)`. The inflating of the object hulls by the robots radius and intersection with the paths is performed by features of the `Clipper\(^7\)` library which was integrated into this project. This library features three different types of inflating a polygon. In this case the round type was chosen as it does not produce additional sharp edges which might lead to the false assumption that an object would block the robots path.

\(^4\)http://openmp.org/
\(^5\)http://www.boost.org/doc/libs/1_56_0/doc/html/program_options.html
\(^6\)http://log4cpp.sourceforge.net/
\(^7\)http://www.angusj.com/delphi/clipper.php
4.5 Designed Strategy

After the Pattern Matching detected an exceptional situation by the message flow it runs a behavior that suits the scenario. This chapter describes one strategy that can be applied to solve a blocked path with help of a person. This strategy solves more complex navigation tasks by analyzing the information provided by the component described in the previous chapter. It utilizes them to reach a navigation target in a more sophisticated way. The behavior is defined as a state chart for the BonSAI framework and can be directly executed by the pattern matching component.

4.5.1 Behavior Structure

The behavior is structured into skills which define a single behavior of the robot and transitions between the skill which allows to combine them to a more complex behavior. Figure 2 shows a visualization of the behavior designed for the task of cooperatively resolving a situation in which the robot encounters a blocked path. The boxes with a light blue tint in the figure are representing the skills which are small java programs that focus on a single task. The arrows between them mark transitions between them. If a skill ends, it always has at least one exit state that can trigger a transition to another skill. The state machine starts off with a skill that stops the current movement actions performed by the robot. As this skill is the initial skill it is highlighted with an orange tint. This allows observing the scene without having the camera moving. Before the robot starts the actual scene analysis it announces that a special situation scenario was detected. Then a new skill which analyzes the scene is run. Depending on its outcome, the robot either asks for help, drives to another position before he asks for help or continues because there is no one to help or no obstacle is detectable. After it asked for help it continues to observe the scene, waiting for the path to become cleared and learning from the observations. In the end the state machine that was running before the interruption continues.
4.5 Designed Strategy

4.5.2 Analyze Scene Skill

After the initialization steps, the state machine goes to a newly created scene analysis skill. The AnalyzeScene skill plays a central role in this behavior. It requests the data from the semantic annotation mapping and the ASBP component as well as it requests information about persons surrounding the robot. If there is no object blocking the path a talk skill announces this situation and the state machine is exited. Due to its limited arm strength BIRON is not yet able to open doors by itself, therefore this state machine relies on help to dissolve the blockage of its originally planned path. Thus this skill checks for a person early and exits if no one is around to be asked for helping the robot.

If the requirements are met, e.g. having a person nearby and something is blocking the robots way, the skill differentiates between two scenarios. One possibility is that the scene is yet unknown for the robot, which means that it has no knowledge which objects in the scene are movable and in which areas they are moved. In this case the robot stands still and observes the situation while a person clears its path way. This way the mobile companion is able to add information about this situation to its semantic map. All movable objects that are manipulated in the range of the camera are added to a layer of the SeAM component. This information can then be utilized when the robot encounters this situation the next time.

The other possibility is the robot in a situation which has been already observed. This observation can be used in a sophisticated way as a lot of objects have distinct areas of movement. Especially doors move always in the same area and chairs will move mostly around tables. In this case the information about areas in which objects are manipulated is utilized to reposition the robot to a place from which it is able to observe the scene without disturbing the assisting person. Therefore viewpoints are calculated around the current manipulation area. The closest viewpoint that is reachable will be set as target. The robot drives to this position and observers the scene from there.
4.5 Designed Strategy

Not only SeAM is utilized to store information from observing. ASM stores a background model of each scene already observed. In conjunction these two components can deliver more information to the `AnalyzeScene` skill if the robot encounters the same obstacle again. Especially if a more complex interaction with the helping person gets designed, which includes e.g. sending the robot back if it is blocking the door, additional knowledge comes in handy. At the second encounter the robot already knows in which area the object moves and thus does not need to be told do step out of the way.
5 Evaluation

The work presented in this thesis aims to improve the robot’s behavior in everyday situations. An important requirement was to design a software framework that can be applied to an existing robot companion while keeping existing properties and behaviors intact. In addition the reusability and general applicability to a wide set of situations of the newly designed components had to be kept in mind. The individual evaluation of the components focuses on proving the general operating ability. For the ASBP component it was important to show that it is able to extract the important information out of different kinds of input from ASM and the navigation component. The pattern matching had to prove being able to detect any kind of given pattern reliably, even in large streams of messages and to execute a strategy that suits the situation. Finally the whole system has to work together to demonstrate the collaboration on the real robot.

The evaluation of the components created as part of this thesis was at first performed component-wise. These tests included unit testing and analyzing the results in the generated log files. The unit tests were already incorporated into the pattern matching in an early stage of development and since then executed each time something was changed in the code. This ensures not to break already functioning features of this software. In addition the components were tested with simulated and real input from other components. Multiple runs in the simulation of the BIRON system during the design and testing phase led to stable components. While the test-driven development already led to components that fulfill their purpose, they had to be evaluated on the robot companion. A test on the BIRON platform where it had to solve the stated
example situation as stated in chapter 4.1 was performed in a way that might occur similar in the real-world.

None of the implemented unit tests was allowed to fail after a change of the pattern matching program. To evaluate the reliability of this component the cases in which it did not select the correct behavior could be logged. Therefore the pattern from listing 1 was loaded during simulated and real-world tests. As the behavior performed by this pattern always states the sentence “That was unexpected!” it is easy to determine whether the pattern executed this behavior or not. The performance of the ASBP component was evaluated by observing the output in a controlled situation with the camera on a tripod and multiple simulated runs. The runs of the whole framework in the simulation were documented with photos and screen captures.

5.1 Pattern Matching

The most important task for the pattern matching component is to execute a behavior that helps to solve a difficult situation. Therefore it hast to be able to reliably detect a pattern in the communication that is defined in a Json file and to execute the behavior defined in a state chart. Correctly interpreting the Json file to detect the described pattern in communication is ensured by implemented unit tests. These tests were designed for the JUnit\(^1\) framework which allows defining multiple tests for each method and combining them to test complex scenarios. About twenty kinds of tests were run each time the software was changed. That includes generating chains of example messages and different patterns and automated verification of the outcome.

The JUnit tests were supplemented by testing the component with message streams generated with tools from the communication middleware and from the BIRON simulation. In the simulation a situation was repeatedly reenacted to verify that the correct behavior is triggered. For

\(^1\)http://junit.org/
example the simulated robot was given a navigation target and while he was trying to reach it the way was blocked. This showed to be reliably detectable by this component.

5.2 ASBP Component

The ASBP component edits data from the Articulated Scene Model and compares it to paths from the navigation component. Therefore input from at least one of those components is needed for testing this part of the framework. For a complete test of this component input from both ASM and navigation is needed at the same time. The simulation of the BIRON system features a simulated 2D environment that allows simulating the navigation and laser input with the Stage\(^2\) software from the Player Project. ASM does not provide a simulation, thus it was run with a 3D camera attached, providing real data perceived from the laboratory. For testing different kinds of scenes, as shown in figure 3.2, with some static parts and mostly larger movable objects that are able to block the robots way were setup in front of the camera. In the simulation for the navigation component the same situation as in the real world was reconstructed to generate paths that can be processed by ASBP.

Most important was the analysis of the messages the ASBP component sent while getting a certain input from the other components. These messages could successfully be logged with a skill of the BonSAI framework as well as being received by SeAM. As SeAM features a graphical user interface, screen captures were created throughout the simulated- and real world tests. Captures from one of the real world tests can be seen in figure 5.7 were a part of the output of ASBP gets correctly mapped into SeAM. The internal representation of ASBP can be observed with its 3D graphical interface as well. It could be seen that it reliably extracts information about blocking static and movable from the given input. Figures 5.1a to 5.2f show screen-shots from one of multiple runs where a person opens a door in front of the robot.

\(^2\)http://playerstage.sourceforge.net/
5.2 ASBP Component

On the performance side ASBP was always able to process the whole data received from the ASM component. This means that if both components are run on the same or similar computers the processing of ASBP showed to be able to keep up with the 10fps provided by ASM. As the screen capture from figure 5.2f shows above it might happen that ASBP is not able to cluster the received depth data from ASM into single objects. This occurs when parts of the scene are partly covered or too far away. In this case another door on the corridor was split into multiple small objects which resulted in five generated hulls around that door. For the application of this component this is no problem as the generated hulls still mark the same area as occupied as the if the door was clustered as a whole object.
5.2 ASBP Component

Figure 5.1: The robot is standing in front of a closed door. It observes the scene with ASM and processes the data with ASBP.

Figure 5.1: A person enters the scene and opens the door for the robot while it is observing the situation.
Figure 5.1: The door gets fully opened allowing a view into the corridor. Right after the person steps back ASBP marks the path as clear, as neither static nor movable objects are blocking the originally planned path.
5.3 Framework on the Robot

After the parts of the created framework had been tested and evaluated by themselves and in smaller groups in the simulation the whole behavior was run in the simulation combined with real world input for ASM. One crucial down-side of this test method is the fact that the camera does not move along with the robot. The camera either stands at the same position for the whole run or has to be moved by the experimenter during the test run. This might influence the output of ASM, especially the merging of already known scenes gets simplified significantly. These simplified runs were followed by tests on the actual BIRON platform. Therefore the components were evenly distributed on the two laptops and the parameters of the components were adjusted according to the robot.

The scenario in the real world was identical to the one tested in the simulation and stated in the example situation from chapter 4.1. As picture 5.3a shows the robot starts in the middle of the laboratory and the door gets closed. When the robot reaches the closed door (fig. 5.3b) the optimized behavior for such situation should get selected by the pattern matching. Figure 5.5 shows a series of screen captures of the visualization of the navigation component’s internal state. Part 5.5c shows the default reaction of the navigation component when approaching an obstacle. It generates a new path along the corridor which is significantly longer.

Figures 5.6 and 5.7 show how the internal representation of that scene changes when the robot observers a scene for the first time. Screen capture 5.6a shows in the smaller picture in the bottom right corner that the navigation component detected an obstacle on its planned path which caused the planning of a new path. This resulted the behavior described in chapter 4.5 being executed and thus leading to the point were the scene is observed. The screen captures 5.6b and 5.6c show how the internal model is created during the opening of the door and 5.6d visualizes the internal representation after the door has swung open. The perception of the articulated scene model gets stopped and a model of this scene is
5.3 Framework on the Robot

(a) The experimenter closed the door. (b) The robot continues driving along its path, approaching the door.

(a) After the robot asked for assistance (b) The robot detects that the obstacles were removed and thus continues driving towards its target. The robot observed this scene to remember the door as a movable object.

Figure 5.4: This series of pictures illustrates the evaluation process with the real robot leaving the laboratory. The experimenter assists the robot reaching the desired target at the corridor.
5.3 Framework on the Robot

(a) The robot has planned a path to its target and starts moving towards it.

(b) After half the way the closed door is not yet considered as an obstacle.

(c) About two meters in front of the door the robot can not follow its short-term plan anymore and thus creates a new plan all the way along the corridor.

Figure 5.5: Recordings of the robot’s navigation component’s internal state while reaching a target on the map. The red arrow marks the target with the robot’s orientation. The long-term plan to the target is marked with a yellow line while the short-term plan is colored blue. The minimum distance that has to be kept to obstacles while navigating is marked green.
saved attached to the current position. ASBP successfully detected that the path is clear again and reported this back to the running behavior which then requests the navigation to plan a new path considering the open door. Screen captures 5.7b, 5.7d and 5.7f of SeAM visualize how the information about the movement area of the door is added to a semantic layer. If the robot encounters a similar situation for the second time due to repeating the same test again without restarting or clearing the memory it could be observed that the robot combines the already accumulated information with the new data. This is possible because of the extension by Ziegler et al. [ZSW13] to ASM described in chapter 3.2 Unfortunately the repositioning for not blocking the assistant while helping could not be observed. The robot always kept its position until the door was opened, thanked the assistant and continued towards the target.

As already stated the BIRON platform lacks the ability to open a door all by itself. Because of that the behavior requests information of persons around the robot. If nobody is near it who could be asked for help, the robot has to exit the advanced behavior which means to follow the longer path all along the corridor. In case the other door to the corridor is closed as well he is not able to reach the goal at all. This kind of behavior could be reproduced by not having anyone near the robot while the pattern matching executes the designed behavior.

While walking next to the robot or moving an object close to its laser range finder normally should not trigger the navigation component to generate a new path to the target, it still might happen sometimes. With the pattern, shown in listing 1, loaded into the pattern matching in terms of messages in the system it is identical to the tested case. This results in the same behavior being selected and executed for that situation. As such case should not lead to worse behavior it had to be evaluated as well. It showed that this case is handled well by the behavior. As the path was already clear again when the scene observation took place the behavior reported that nothing is blocking its way and the original behavior continued.
5.3 Framework on the Robot

Figure 5.6: Comparison between the ASBP’s interpretation of the scene and information from the navigation component. After ASBP reported the scene as clear to the behavior it triggers the navigation component to plan a new path. This causes the navigation component to drop the path along the corridor and take the short cut through the door way as can be seen in figure 5.6d.
5.3 Framework on the Robot

(a) ASBP receives scene as static
(b) No data on movement areas yet
(c) Door half open
(d) Added first movement area
(e) Door open
(f) Movement area completed

Figure 5.7: The information about movement areas (black) of objects is stored in a map layer of SeAM (fig. (b), (d) and (f)) while watching the assisting person clearing the path-way. This information can be used to position the robot outside this area to allow the assisting person to help without disturbing. Especially for doors this helps a lot as doors are always moved in the same location due to their hinges.
The results showed that the work of this thesis was able to improve the robots navigation behavior in complex situations. Such situations can be detected by only observing messages flowing in the system. Even minor misclassifications due to people stepping in the robot’s way just lead to small interruptions during the navigation. Screen captures during the test runs make clear that the robot is able to generate an internal representation about movable objects. This information could be added to semantic maps and can later be used for an even faster cooperation. The repositioning might not have been observable because the scenario was too simple. The navigation component created a new plan early enough to allow the robot to stop in a distance that made it unnecessary to step back. Nevertheless it could be shown that all necessary information is gathered and stored for later use. This way a scenario with more interaction can be solved in the future.
6 Conclusion and Outlook

The framework designed for this thesis enables a robot companion to dynamically react with a more sophisticated behavior to upcoming obstacles. This goal was accomplished by incorporating multiple existing software projects and components. Most of them had to be only slightly modified in terms of updating the communication middleware to be able to process the data in a standardized way. Bigger Changes could be prevented at the outset of the project by designing a system that has as few requirements as possible.

As central part of the framework a software component was designed that is able to apply complex strategies to exceptional situations just by observing the robot’s internal message flow. As an exemplary situation in which the robot’s path gets blocked by an obstacle a behavior was created as state chart for the BonSAI behavior framework. This behavior analyzed the scene and resolved the blockage cooperatively. It searches for a person, asks for help and observes the situation if it is encountered for the first time in that location. When the robot has already observed this situation the behavior makes use of the gathered data by positioning itself outside the typical movement area of the articulated object. If problems occur during execution, like finding no person or being unable to detect a blocking object the behavior lets the robot state these situations with speech output. It could be shown that even in systems with separated components it is possible to fuse information for an informed behavior with the created ASBP component. This component processes data from the navigation and ASM to make these usable in behaviors that make use of typical movement regions of articulated objects.
The created framework was evaluated incrementally, starting with testing parts of the created software individually to be sure that basic requirements are met. Later on larger groups of components were evaluated in the simulation. This made sure that they serve their purpose. A test on the robot proved that the framework is able to detect a situation in which the robot’s path is blocked. The strategy to solve such situation was repeatable selected correctly. The obstacle could be removed by cooperation with a person and the robot was able to extract new information about the moving object by observing. Repositioning due to learning from the first run was hard to show in the chosen scenario. The robot kept its position and waited for the assisting person to open the door. Then the robot continued through the door to the target.

The BIRON platform features multiple existing software components with two different kinds of communication middleware. This resulted in inhomogeneous types of messages which made it challenging to combine information from multiple sources. A lot of effort was put in the task of creating a more homogeneous communication infrastructure. Parts of the communication of the navigation component, asm, SeAM and the BonSAI framework were updated as part of the work of this thesis.

In the future this problem will be eliminated by switching all components to the RSB middleware. Additional improvements to the framework can be applied by adding more behaviors for different scenarios. The created behavior can be tested in slightly modified situations which might benefit more from learning by observing a scene. This might improve the results further and lead to an even more polished overall behavior. Directly incorporating a door detection [MSZW14] or integrating furniture/object detection might improve the behavior as well. As knowledge which objects can be moved by BIRON itself could be added and make it less dependent on an assistant.

The aim of this thesis was to enrich a robot’s behavior with additional scene knowledge to solve more complex scenarios. It could be shown that an optimized behavior can be achieved without modifying the system architecture or existing components.
Appendices
Figure 1: UML class-diagram of pattern and barriers. Realization of the general structure into java classes. Pattern and links hold list of barriers while a link is a barrier itself that appears as and or. Through this inheritance it is ensured that any kind of pattern is possible.
Figure 2: State-Chart of the designed behavior. Shows possible states during the behavior of solving a cooperative navigation task and possible transitions between them.
Listing 1: Pattern that is used to detect a problematic situation while navigating to a target on the map, that can be optimized by a more complex behavior.
.3 Statement of Authorship

I declare that this document and the accompanying code has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. It has not been accepted in any previous application for a degree. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

__________________________    ________________________
Ort, Datum                                Unterschrift
Literature


[BAA+07] Barnett, Jim; Akolkar, Rahul; Auburn, RJ; Bodell, Michael; Burnett, Daniel C.; Carter, Jerry; McGlashan, Scott; Lager, Torbjörn; Helbing, Mark; Hosn, Rafah et al.: State Chart XML (SCXML): State machine notation for control abstraction. In: *W3C working draft* (2007)


[QCG+09] Quigley, Morgan; Conley, Ken; Gerkey, Brian; Faust, Josh; Foote, Tully; Lehbs, Jeremy; Wheeler, Rob; Ng, Andrew Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software Bd. 3, 2009


[Sie13] Stepmann, Frederic: Behavior coordination for reusable system design in interactive robotics, Bielefeld University, Dissertation, 2013


[Zie10] Ziegler, Leon: Developing a Vision-Based Object Search Behavior for a Mobile Robot, Universität Bielefeld, Diplomarbeit, 2010


## List of Figures

2.1 Exemplary SLAM Map ........................................ 8

3.1 The BIRON Robotic Platform ................................. 11
3.2 Articulated Scene Model Example ........................... 12
3.3 BonSAI Layers .................................................. 14

4.1 Navigation Sequence ........................................... 20
4.2 Example Situation - Path Blocked ............................ 21
4.3 Problem Structure ............................................. 23
4.4 Pattern Matching Integration ................................. 25
4.5 Pattern Matching Application ................................. 27
4.6 ASBP Sequence .................................................. 33
4.7 ASBP 3D-View .................................................. 36

5.1 ASBP Evaluation - Closed Door ............................... 45
5.1 ASBP Evaluation - Assistant Opens Door .................. 45
5.1 ASBP Evaluation - Door Opened .............................. 46
5.4 Series of Pictures from Evaluation on the Robot ........ 48
5.5 RVIZ - Path Planning Visualized ............................ 49
5.6 ASBP Compared to Navigation ................................. 51
5.7 Using SeAM for Mapping Movement Areas for Objects . 52

.1 UML class-diagram of pattern ................................. 57
.2 Behavior State-Chart ........................................... 58
## List of Tables and Listings

4.1 Pattern States ........................................ 28

4.1 Example JSON pattern ................................. 29

1 JSON pattern for navigation problem detection .... 59