Symbolic and Non-Symbolic Failure Interpretation and Recovery Using a Domestic Service Robot

Ron Snijders

April 2016

Master Thesis
Artificial Intelligence
Institute of Artificial Intelligence
University of Groningen, The Netherlands

Internal supervisor:
Dr. H.B. Verheij (Artificial Intelligence, University of Groningen)

Second evaluator:
Prof. Dr. L.C. Verbrugge (Artificial Intelligence, University of Groningen)

university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Abstract

Domestic service robots need to be robust against noise and a large degree of uncertainty. This also requires the ability to detect, recognize and resolve previously unknown failures during their lifetime. Existing research offers promising solutions, but typically depends on what was foreseen by its application. In this research we address this problem by the design, implementation and verification of an adaptive behavior architecture capable of autonomous failure recovery. The recovery performance of two different methods using a non-symbolic or a symbolic representation of the failure state respectively, is compared and evaluated. In addition, a method is proposed for the autonomous perception of symbols in the environment from low level sensory information.

The non-symbolic approach uses low level sensory information (RGBD data retrieved from a color and depth camera) to estimate the current failure state the robot is in. A dissimilarity measure is used to select the k most similar failure situations. The number of k samples to use is dynamically determined by a sudden change in either the dissimilarity or the score distribution of the closest samples, whichever comes first.

In its most basic form, the symbolic approach uses a Naive Bayes classifier to select the best recovery solution with the highest probability of being a success, given a set of symbols consisting out of concepts (nouns) and their properties (adjectives). In the extended form, the symbolic approach uses a set of transformed representations of the original symbolic representation, of which it is able to learn the best representation most suitable of a given failure situation.

The symbol perception module uses a region growing algorithm to segment the pointcloud, as retrieved from a RGBD camera, into multiple surfaces. From each surface, a collection of features are extracted, such as its similarity to known 3D models using the MLESAC algorithm, a binned color histogram and metric information using PCA. After accumulation of labelled training samples, a template is created to which an unclassified segmented pointcloud can be matched to. Each feature is weighted by estimating the inverse overlap of the probability density function of one class to all other classes prior to finding the most prominent prototype vectors of a given class during template creation. During classification, a concept is added to the symbolic representation if a sufficient number of segmented surfaces have been matched to the corresponding concept class template. Once a concept is added to the symbolic representation, its properties are classified using kNN.

Without knowing the different types or the total number of failure situations, both the non-symbolic and symbolic approach of failure recovery are able to learn recovery solutions at an adequate level. Using the symbolic representation yields the best recovery performance while being robust against misclassifications in the perception of the symbols. The symbolic approach is capable of learning the best simplification of the original representation, thereby increasing its performance while using this new representation to provide suggestive information as to *why* the failure occurred.

Acknowledgements

The author would like to thank the University of Groningen and the Institute of Artificial Intelligence in Groningen for facilitating the project in its needs, the BORG team for providing inspiration for new ideas in the early stages of the project and Enacer B.V. for providing the RITA robotic platform and its software libraries for experimentation. In addition, the author would like to thank Dr. Tijn van der Zant for his involvement during the initial phase of the project and Prof. dr. Rineke Verbrugge for her involvement towards the end of the project. Special thanks go to Dr. Bart Verheij for his continuous inspiration and support during the design and execution of the project.

Contents

Abstract ii

Acknowledgements iii

1 Introduction 1
- 1.1 Failure Recovery and Fault Tolerance . . . 1
- 1.2 Phenomenon of Blindness . . . 1
- 1.3 Symbolic and Non-Symbolic Representations . . . 2
- 1.4 Research Goals . . . 2
  - 1.4.1 Ground Truth Failure Recovery . . . 4
  - 1.4.2 Non-Symbolic Failure Recovery . . . 4
  - 1.4.3 Symbolic Failure Recovery . . . 4
  - 1.4.4 Symbol Perception . . . 5
- 1.5 Structure of this Thesis . . . 5

2 State of the Art 7
- 2.1 Types of Failures . . . 7
  - 2.1.1 Internal Faults . . . 7
  - 2.1.2 External Anomalies . . . 8
- 2.2 Failure Robust Behavior Architectures . . . 8
- 2.3 Reversible Computation . . . 9
- 2.4 Human Robot Interaction . . . 10
- 2.5 Robocup@Home . . . 10

3 Experimental Setup 11
- 3.1 The RITA Robot . . . 12
- 3.2 Behavior Architecture . . . 14
- 3.3 Robot Operating System . . . 14
- 3.4 Human Robot Interaction . . . 15
- 3.5 Simulation . . . 15
- 3.6 Test Scenarios . . . 17
  - 3.6.1 Required Recovery Solutions . . . 17
  - 3.6.2 Test Scenario 1; Basic Concepts . . . 18
  - 3.6.3 Test Scenario 2; Different Colors . . . 19
  - 3.6.4 Test Scenario 3; Different Locations . . . 19
- 3.7 Performance Measure . . . 19
- 3.8 Dataset Generation . . . 20
  - 3.8.1 Simulated Rewards . . . 20

4 Ground Truth Failure Recovery **21**
4.1 Failure Recovery Techniques 21
4.2 Rewards 22
4.3 Exploration Schemes 23
4.3.1 “Naïve” 23
4.3.2 ϵ-Greedy 23
4.3.3 Interval Estimation 23
4.4 Gaining Experience 24
4.5 Results 24
4.6 Discussion 24

5 Non-Symbolic Failure Recovery **27**
5.1 Low Level Sensory Information 27
5.2 Dissimilarity Measure 27
5.3 Experience Selection 28
5.4 Failure State Estimation 30
5.5 Results 31
5.5.1 Incremental Failure States 31
5.6 Discussion 33

6 Symbolic Failure Recovery **35**
6.1 Symbolic Representation 35
6.2 The Bayesian Approach 36
6.2.1 Results 37
6.3 Dynamic Representations 37
6.3.1 Ontology Generalization 37
6.3.2 Formulas in First-Order Logic 37
6.3.3 Representation Selection 39
6.3.4 Results 40
6.4 Discussion 40

7 Symbol Perception **43**
7.1 Learning Procedure 43
7.2 Specialized Perception Modules 43
7.3 Background Subtraction 45
7.4 Pre-Processing 45
7.4.1 Segmentation 46
7.4.2 Feature Extraction 46
7.5 Template Creation 48
7.5.1 Feature Weighting 49
7.5.2 Training Data Normalization 49
7.5.3 Finding Prototypes 50
7.6 Concept Template Classification 50
7.6.1 Property Classification 50
7.7 Results 51
7.8 Discussion 51

8 Conclusion **55**
8.1 Limitations 56
8.2 Future Perspective 56

Bibliography **56**

A Sliding Window Score Distribution Cut-off Algorithm **63**

B ROS Packages **65**

B.1 RITA Platform 65
B.2 Project Related 66
B.3 ROS Graph 67

Chapter 1

Introduction

It is clear that the development and application of domestic service robots is growing rapidly. Whereas basic household robots are already common practice [1], multi-purpose domestic service robots capable of handling complex tasks are soon on the rise [2, 3]. The complex dynamics of ever-changing domestic environments require these situated robots to be robust against noise and a large degree of uncertainty [4].

In the foreseeable future, this development and application of domestic service robots is starting to become more of a necessity rather than a luxury. This becomes especially apparent in the field of elderly healthcare, in which the ratio of elderly people to the working age population is projected to almost double by 2050 [5, 6]. This frail group of users puts an even higher demand for the domestic service robots to operate safely and sensibly with less down time compared to what has previously been possible [7, 8].

1.1 Failure Recovery and Fault Tolerance

From an engineering perspective, it seems natural to regard any failure during the operation of a domestic service robot, as something to avoid at all cost. This is especially apparent on a physical level, at which the robot has to interact safely with the environment and its inhabitants. The development of new standards [9] to ensure this safety should therefore come as no surprise.

However, on a functional level, which involves frequent changes to the domestic environment and demands of the user, it becomes increasingly more difficult to account for all possible failure conditions beforehand. It is therefore important to realize, that the constant anticipation, recognition and recovery of new failures, is in many cases the *default* state in which a domestic service robot operates, rather than the exceptional state as often described in literature. This implies that such a robot should be able to detect and, at later stages, recognize unseen failures or anomalies, in order to adapt its behavior in future events.

1.2 Phenomenon of Blindness

This need to overcome failures on the fly also implies that such a robot should have some sort of situational awareness of its environment which goes beyond its initial programming. However, in practice, the situational awareness of a robot is biased towards ideas of its designer. This is sometimes referred to as the phenomenon of blindness [10]. The demonstrated behavior is in such a case the result of manual behavioral programming, in which the programmer is doing most of the integration of sensory data and the classification of certain situations relevant for the given task. Failure recovery is often limited and requires the different types of possible failures to be known beforehand. This may result in a machine that is very brittle in a dynamic environment in which goals and various (failure) conditions may vary significantly over time. Such a robot requires constant manual programming and parameter tuning whenever a new type of failure is discovered. Instead, a more “skull-closed” approach is desired [11], in which the robot solely uses its sensory ends and motor ends to explore the world and communicate with its user. If a domestic service robot is to survive, on its own, in a domestic environment, it should be able to detect unseen situations or anomalies, be able to recognize them and adapt its behavior *autonomously* in order to respond accordingly in future events.

1.3 Symbolic and Non-Symbolic Representations

At the occurrence of a failure, the robot relies on its perceptual capabilities to learn and, at later stages, recognize failures. This perception can be represented in two different ways; a symbolic and a non-symbolic representation. In the non-symbolic representation, the system uses low level sensory information, such as retrieved from a laser range finder or a Color/Depth (RGBD) camera, to perceive failures, whereas in the symbolic representation, the environment is represented in a descriptive manner as a combination of symbols or words. This differentiation of representations can be compared to the Cognitivist vs. Emergent paradigms of cognition [12] in which respectively either the symbolic or non-symbolic representations is used. Unlike the non-symbolic representation, the symbolic representation can be understood by both the robot itself and any human user if some form of Human Robot Interaction would be utilized. However, this still begs the question as to how these symbols are learned and perceived in the environment.

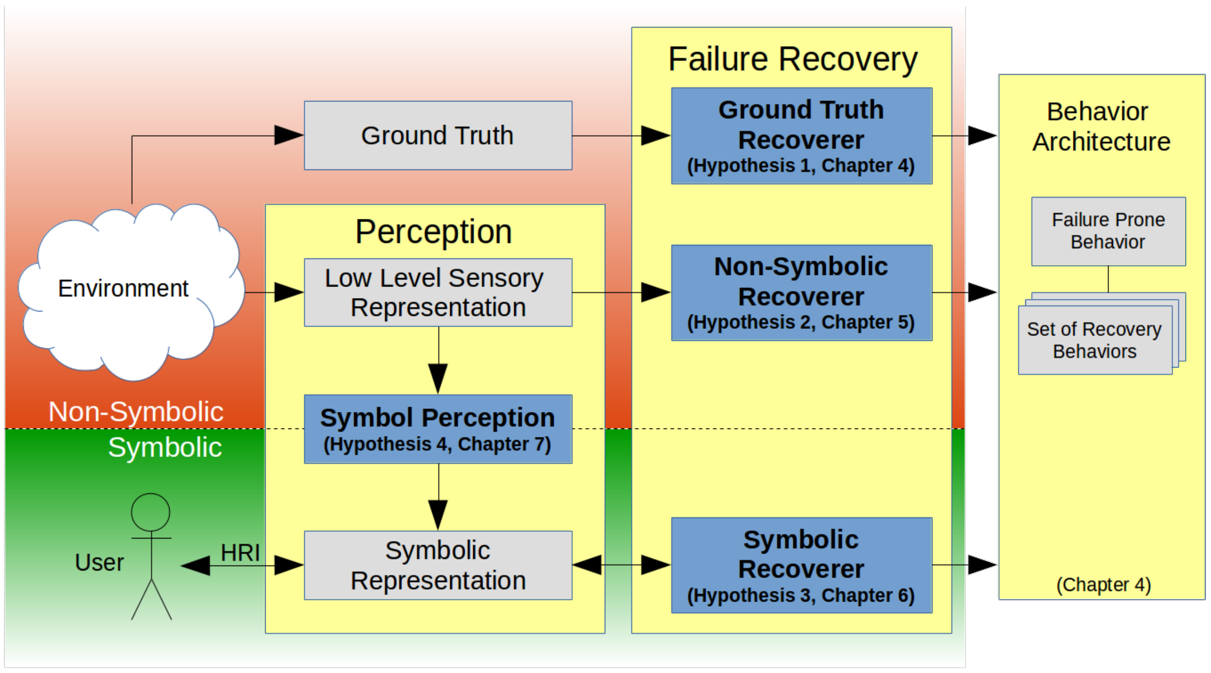
1.4 Research Goals

The research discussed in this thesis aims to design, implement and verify an adaptive behavior architecture capable of autonomous failure recovery, in which the robot is capable of distinguishing one failure situation from another and is able to learn the best recovery strategy. The robot has to do this while prior to learning, the set of all possible types of failures, is not known by the robot. An overview of the complete project as discussed in this thesis is given in Figure B.1.

The *primary* goal is to compare and evaluate the recovery performance of using a non-symbolic (Chapter 5) vs. a symbolic (Chapter 5) representation of the failure situation. To provide a baseline performance for comparison, the general failure recovery capabilities of the behavior architecture are first evaluated using ground truth information (Chapter 4). In this case, the exact failure state is known to the robot, in which neither the non-symbolic nor the symbolic representations are used.

A *secondary* goal is to allow the symbols in the symbolic representation to be learned and, at later stages, perceived autonomously by the robot from low level sensory information (Chapter 7). This allows the symbolic approach to indirectly use the same low level sensory information as used in the non-symbolic approach.

The following sections describe each component in more detail and set the specific hypotheses to be verified in the remainder of this thesis.



Failure Recovery
Ground Truth
Ground Truth Recoverer (Hypothesis 1, Chapter 4)
Behavior Architecture
Perception
Environment
Low Level Sensory Representation
Non-Symbolic Recoverer (Hypothesis 2, Chapter 5)
Failure Prone Behavior
Set of Recovery Behaviors
Non-Symbolic
Symbolic
Symbol Perception (Hypothesis 4, Chapter 7)
User
HRI
Symbolic Representation
Symbolic Recoverer (Hypothesis 3, Chapter 6)
(Chapter 4)

Figure 1.1: Schematic overview of the project and architecture discussed in this thesis. The behavior architecture supports failure recovery using either a symbolic representation (Chapter 6) or a non-symbolic representation (Chapter 5) using low level sensory information. Using Human Robot Interaction (HRI) [13, 14] (Section 3.4), the current symbolic representation can be explained by the robot to the user or vice versa. It furthermore allows for automatic labeling of training samples and verification of hypotheses inferred by the symbolic interpretation. The primary goal of this project is to compare the performance of different failure recovery methods using either a symbolic representation or a non-symbolic representation. The performance of the "Ground Truth Recoverer" (Chapter 4) is used as the baseline performance in which the actual failure state is known using ground truth information. A secondary goal of this project is to design and verify a generic perception module, capable of autonomous symbol grounding using low level sensory data. Using Symbol Perception (Chapter 7), the architecture is able detect and classify symbols (in the form of nouns and adjectives) from low level sensory information, while still being able to take advantage of the symbolic interpretation. See text for more details.

1.4.1 Ground Truth Failure Recovery

The Ground Truth Recoverer neither uses the non-symbolic nor the symbolic representation, but rather uses the ground truth information available during experimentation. In this case, the actual failure state is known beforehand and available to the recovery method as a unique label during experimentation. The robot is thus not required to learn or classify the failure state itself. Using this representation is expected to yield the highest failure recovery performance and is being used as the baseline performance for comparison with other methods. It also serves to test the basic failure recovery capabilities of the behavior architecture in addition to the different exploration schemes available for providing a good balance between exploitation and exploration. Chapter 4 discusses this method in more detail and aims at confirming the following hypothesis:

Hypothesis 1 *Given a known failure state, the robot is capable of learning the best recovery solution.*

1.4.2 Non-Symbolic Failure Recovery

Using the non-symbolic representation, the robot only knows when a failure has occurred, but is not being told any additional information. Purely using its sensory information, the robot must estimate the current failure state and learn its solution autonomously without knowing the total set of possible failures beforehand. Chapter 5 explains the failure recovery method using the non-symbolic representation in more detail and aims at confirming the following hypothesis:

Hypothesis 2 *The robot is capable of learning the best recovery solution in new failure situations using solely low level sensory information.*

1.4.3 Symbolic Failure Recovery

Using the symbolic representation, the robot aims to recover from new failures using an explicit interpretation of the symbols themselves. For the purpose of this project, the symbolic representation is limited to the presence of nouns and adjectives, which represent concepts and their properties (such as color and location) in the environment. Chapter 6 explains the failure recovery using the symbolic representation in more detail and aims at confirming the following hypothesis:

Hypothesis 3 *The robot is capable of learning the best recovery solution using a symbolic representation of observable concepts and their properties.*

1.4.4 Symbol Perception

In order for the robot to utilize the symbolic recoverer, it must be able to autonomously perceive the world in the form of a symbolic representation. The Symbol Perception module serves to translate the low level sensory information from the environment to this symbolic representation.

In the context of the Symbol Grounding Problem (SGP) [15, 16], the general solution provided in this thesis can be compared to work of [17], in which the symbolic representation is grounded in the sensorimotor activities of the robot. Here, the nouns and adjectives themselves represent the *form* (or "representamen"), the recovery solution the *meaning* (or "interpretant") in the context of the failure situation, while the actual perception of symbols using low level sensory information serves to define the *referent* (or "object") of the symbol in the semiotic triangle [16, 18]. Since during learning, the *referent* is typically unknown (e.g., it can be an object or a location, time of the day, etc.), the method proposed in this thesis aims at being as generic as possible.

Chapter 7 explains the method of autonomous symbol perception and serves to confirm the following hypothesis:

Hypothesis 4 *The robot is capable of learning generic concepts and their properties in the form of nouns and adjectives from low level sensory information.*

1.5 Structure of this Thesis

This thesis is structured as follows. First, Chapter 2 provides the reader with a discussion on some of the state of the art solutions in dealing with the detection and recovery of failures in the field of robotics. Next, in Chapter 3, the reader is provided with an overview of the robotic architecture and the experimental setup used to verify the methods proposed in this thesis. Chapters 4, 5, 6 and 7 serve to verify and evaluate the related failure recovery performance of respectively hypotheses 1, 2, 3 and 4, as defined in the previous sections. Finally, Chapter 8 discusses the results of the experiments in more detail and proposes ideas for future improvements.

Chapter 2

State of the Art

This chapter provides the reader with a short discussion on existing solutions in relation to fault tolerance and failure recovery in behavior architectures as seen in literature.

2.1 Types of Failures

In the context of this thesis, we define a failure as the outcome of some anomaly which prevents a domestic service robot from completing its task successfully. The success of a given task, depends on the final state of the environment (e.g., move a drink from the kitchen to the living room), but may also depend on some optimization criterion (e.g., go to the kitchen within 10 seconds). The underlying cause of an anomaly (and thus a failure), may originate from different sources. We can roughly divide these sources in two different categories; anomalies caused as part of the robot and those related to anomalies in the environment. The work presented in this thesis concerns itself primarily with the second category of anomalies. The remainder of this section discusses both categories and provides a summary of solutions as seen in the literature.

2.1.1 Internal Faults

Most research in relation to failure recovery in robotics has concerned itself with fault tolerance on a hardware or software level. Many solutions exist [19, 20], but they are often engineered towards a specific application.

Hardware Fault Tolerance.

Hardware failures may include defective sensors or actuators and loss in performance due to dust or wear and tear. Relying on a robotic system in which it is assumed that all actuators and sensors are working perfectly is often not practical. This is because in some cases, such as the application of robotics in space, it is impossible to repair the system after deployment. In case of a defective joint in space, one could utilize the dynamic coupling between joints to reposition a manipulator to a specific position [21]. In other cases, faults happen too frequently for routine maintenance or repair to be practical [7]. This is especially true for complex systems, which employ numerous sensors and actuators. For example, the hexapod Haniball robot inspired by [22], has over 60 sensors and in about every two weeks a sensor breaks down [23]. To compensate for these faults, the system uses a distributed network of concurrently running processes in which faults of sensors are detected autonomously and confined and abstracted away using virtual sensors [24].

Software Fault Tolerance.

Different techniques can be utilized to increase the general Software Fault Tolerance (SFT) of a system. Common practises during the implementation and test phase include Unit Testing to verify the specified functionality of a system [25] and Fault Injection to verify the error handling capabilities of a system [26]. During the actual deployment of the software, either single-version or multi-version SFT techniques can be utilized [27]. A commonly used multi-version SFT technique is N-Version Programming [28]. Here, multiple versions of the same functionally equivalent piece of software are written by different development teams independently. During execution, a decision algorithm is used to select the best output (typically by voting) from all versions. A similar approach can be used in Machine Learning with the use of Ensemble Methods [29] such as Bayesian averaging, Bagging and Boosting.

2.1.2 External Anomalies

The second category of anomalies include those related to the environment itself located outside the body of the robot. These anomalies are often the result of some change in the environment which prevents the robot from completing its tasks successfully. This results in failures such as the inability to navigate inside a room due to a sudden obstruction, or to grasp an object because the object is not located at its usual location. Many solutions exist for recovering from external anomalies or faults. This including using simulations to predict future faults [30], and logical reasoning to act in response to failures [31, 32]. However, in many of these cases, the solution is engineered towards a specific application and not suitable to be used as a generic solution to recover from an unforeseen external fault.

Compared to the first class of anomalies, traditional methods such as increasing the redundancy or designing application-specific solutions for the purpose of fault tolerance (as discussed in Section 2.1.1) work poorly. The solution must often be found on a behavioral level of the robot. Fault tolerance should therefore, as explained in the next section, be an integral part during the design of a behavior architecture used in domestic service robots.

Even though in general specific solutions have been designed for each specific type of failure, it is important to realize that these types are not completely independent. For example, the lack of internal (computational) resources, affects the way the robot is able to interpret the complexity of the environment and thus the ability to cope with external anomalies.

Furthermore, a fault or anomaly might occur without the actual occurrence of a failure. A system can be intrinsically fault tolerant without dealing with failures explicitly, for example by avoiding faults or anomalies. In the research presented in this thesis, we are explicitly interested in dealing with failing behaviors and learning a specific solution, rather than making the system fault tolerant in general.

2.2 Failure Robust Behavior Architectures

The number of robotic architectures seen in literature which explicitly employ fault tolerance and failure recovery techniques are relatively limited. The problem is often approached by making the specific sub-components of the system fault tolerant or by anticipating specific failure scenarios during the design and implementation of the system. In other cases, fault tolerance and the ability to learn failure recovery solutions are indirectly an inherent result of the algorithm. The remainder of this section discusses a collection of behavior architectures which have explicitly been designed with fault tolerance in mind.

An interesting behavior architecture designed with fault tolerance in mind is the work of [33]. The architecture has been applied to an autonomous underwater vehicle which must remain operational for several weeks without human intervention. Here, a distributed control system has been designed capable of failure handling, even if the source of the fault cannot be identified. The system is designed to do "whatever works" with the use of multiple behaviors providing redundant pathways for solving a problem in different ways. Different (possibly redundant) behaviors compete with each other using an activation net, in which the activation of the behavior depends on its relevance to the given tasks and its success during previous executions. The architecture provides an interesting approach in dealing with new, unknown and unexpected failures, but provides no direct means of explaining the failure to a human at a later stage.

The work of [34] provides another interesting approach to fault tolerance in a behavior architecture for the cooperative control of teams of heterogeneous mobile robots. It employs a hybrid solution of negotiation between team members and a motivational mechanism which activates or inhibits the output of the behaviors to the robot's actuators. Upon the occurrence of a fault (such as the removal of a team member), the activation patterns of a set of related behaviors is modified as a result of the changing motivation, impatience and acquiescence levels of the robot. The architecture provides an interesting solution for a team of robots to accomplish a specific task, such as hazardous waste cleanup, in a cooperative manner. However, the solution to a fault is provided as a result of the emergent characteristics of the system and is thus not explicitly known or being conveyed to a human.

Whereas in the previous examples shown above, the solution to failure recovery is an emergent property of the system, the alternative is to incorporate failure recovery mechanisms during the preparation or execution of a plan. The hierarchical planning paradigm is a commonly used planning approach (e.g., [35, 36]), in which first an abstract skeleton plan is constructed before refining the detailed steps later on (possibly during execution of the plan). If a failure occurs during the execution of a plan, the robot can either decide to reconstruct the whole plan or execute a specific recovery solution. Executing a recovery solution generally increases response time, but possibly at the cost of plan quality [37].

2.3 Reversible Computation

Another interesting approach is to use the idea of reversible computation (see [38, 39, 40] for a more in-depth discussion on the topic) in an attempt to recover from a failure. Here, the sequence of perception, reasoning and action of a robot could be back-traced in order to find the source of the fault and recover from a failure the moment a failure occurs. For example, the work of [41] uses such an approach in which a Domain Specific Language (DSL) has been designed to create reversible assembly sequences at the occurrence of an error. However, through its interaction with the environment and the loss of information during the perception of the environment, the entropy of the set of all possible failure scenarios increases significantly. The practical use of reversible computation is therefore often limited to the past reasoning of the robot and the distinct actions it took up till the occurrence of the failure. This also requires the (symbolic) representation of the world to be as precise and abstract as possible without losing too much information.

2.4 Human Robot Interaction

Since a domestic service robot often operates in close relation with its human user, it seems natural to include the user in the process of failure recovery. Rather than trying to resolve the failure situation completely autonomously, the robot could ask for help and learn more efficiently through Human Robot Interaction (HRI) [14].

An example includes the work of [42] which allows a robot to ask specific questions in order to resolve the failure situation. The help requests are constructed from a probabilistic graphical model, called Generalized Ground Graph [43], by using its semantic structure in reverse order. This allows the method to not just ask simple questions like "Can you help me?", but also to generate effective and precise help requests (such as "Please give me the white table leg that is on the black table.") in order to resolve the failure in a more effective way.

However, the willingness of the user to comply to the instructions of the robot provides no guarantees that the robot is recovering from a failure in the correct way [44]. Close interaction with the user is thus of utter importance for the robot to verify its perception of the situation.

Furthermore, errors might also occur within the dialogue between a human and the robot. The utilization of effective error handling strategies within the dialogue itself is therefore also of great importance. One possible solution to this problem, as shown by the work of [45], is to apply similar error recovery strategies as used in human-human dialogues to that of human-robot dialogues.

Alternative solutions related to symbolic models and logical reasoning to act in response to failures [31, 32] exist, but often suffer from the symbol grounding problem [15]. Symbols are either hard-coded or perceived by independent perception modules. This results in a system in which only part of the perceivable world is used to detect, classify and explain previously unknown failures.

2.5 Robocup@Home

For domestic service robots to be of practical use, research should go beyond theory and conducting experiments in controlled laboratory settings. This is especially true for testing the failure recovery capabilities of a robot in a highly dynamic and unpredictable *domestic* environment. Benchmarking competitions such as Robocup@Home [46] (one of the main leagues organized by the Robocup Federation [47]) aims to foster the research in autonomous service robots in a domestic environment. During the Robocup@Home competitions, teams from different universities and research institutes compete with each other by demonstrating the abilities of their domestic service robot in different ways. One of the most interesting tests used in Robocup@Home, is the General Purpose Service Robot (GPSR) test. Here, the robot is provided with a random command (for example with the aim to find and bring a specific object to another location) which needs to be executed inside the arena (a typical apartment layout including fully furnished rooms such as a kitchen, living room and bedrooms). However, due to changes in the environment, incomplete information or incorrect instructions, the robot is unable to execute the command in the usual manner.

The behavior architecture made by the BORG team from the University of Groningen offers a promising solution to the GPSR test [48]. It is capable of dealing with underspecified commands possibly with erroneous information, in which the robot is able to start a dialogue with the user to acquire more information or learn a new type of behavior. It is able to handle failures by executing alternative behaviors on the fly. However, the detection and classification of failures need to be programmed by hand and there is no learning involved for the purpose of recovering from unknown failures.

Chapter 3

Experimental Setup

The following experiments described below are used to verify each of the methods described in Chapter 1. This includes the methods which uses ground truth information (Chapter 4), the non-symbolic representation (Chapter 5) and the symbolic representation (Chapter 6). In case of the method which uses the symbolic representation, the experiments are conducted with and without autonomous symbol perception (Chapter 7). Without symbol perception, the symbolic representation is extracted using ground truth information and is provided as-is without the attempted recognition by the symbol perception module.

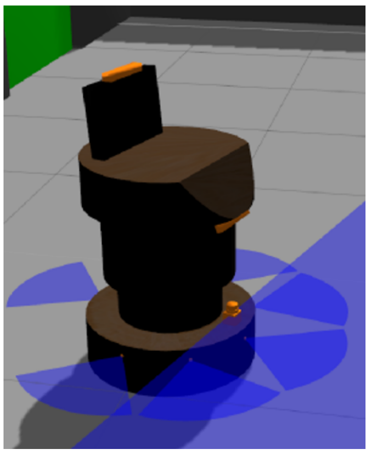
During these experiments, the general purpose of the robot is to enter a room from a random start location using one of the three entrances in the *least amount of time*. At each entrance location, a (possibly new) failure situation is introduced which prevents the robot from entering the room in the usual manner. The goal of the robot is to learn each type of failure and the optimal (possibly unique) solution in resolving the situation using either low level sensory information or a symbolic representation. The number and type of failures are not known to the robot beforehand.

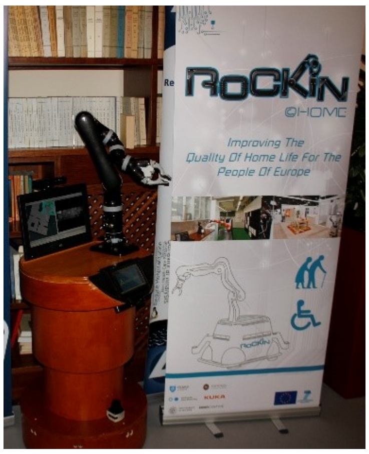
The failure scenarios used in the experiments are relatively easy to solve by a human programmer if all possible failures are known beforehand. However, the purpose of these experiments is not to show that the system can do a specific task (navigation, path planning and obstacle avoidance in this case) in a very efficient manner. Rather, the purpose here is to show that the system can cope with *unforeseen* situations and is capable of creating *its own situational awareness and strategy* in order to improve its general behavior.

The following sections describe the robotic architecture and simulation environment used during the experiments. The setup of the actual test scenarios, as used to verify each hypothesis (see Figure B.1), are described in Section 3.6.

3.1 The RITA Robot

The work presented in this thesis uses the RITA (Reliable Interactive Table Assistant) robot[1] (see Figure 3.1) for experimentation. RITA is an autonomous moving table and has been designed to assist elderly to live at home for a prolonged period of time. It has a variety of sensors, including a laser range finder, two RGBD cameras and a microphone. It has a differential drive and the tabletop (including most of its sensors) can be move up and down. An overview of the software and hardware components as used on the RITA robot platform is provided in Figure 3.2. The RITA robot has been designed and developed by Enacer B.V. of which the author of this thesis is a co-founder.


(a)



(b)

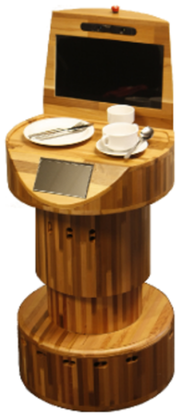

(c)

Figure 3.1: (a) The RITA robot as seen in the Gazebo simulator. The blue cone-shaped planes indicate the orientation and range of the laser range finder and sonars. The RGBD cameras are mounted on top of the screen and just underneath the tabletop (marked in orange). (b) An earlier prototype of the RITA during the RoCKIn@Home Camp 2014 [3]. (c) The newest prototype of the RITA.

[1]`https://www.enacer.com/en/en/rita.htm`

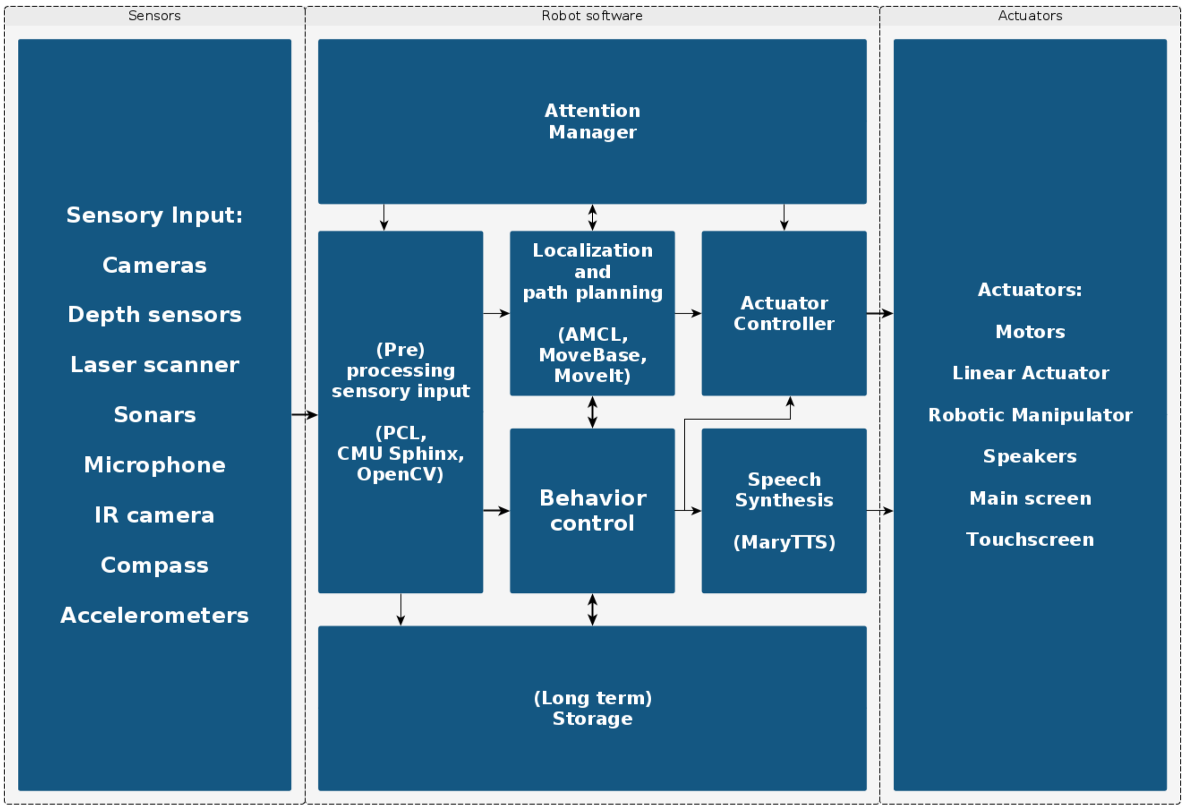


Figure 3.2: An overview of the software and hardware components as used on the RITA robot platform. The Robot Operating System (ROS) (see Section 3.3) is used for communication between the modules. A custom made controller board is used to actuate and control up to four actuators (of which two are used for the differential drive and one for the linear actuator) and a collection of sensors (such as encoders, sonars and accelerometers).

3.2 Behavior Architecture

The RITA uses a behavior architecture developed by Enacer B.V.[2] to perform its daily tasks. Its design is loosely based on the BORG architecture [49], in which behaviors can be run in parallel and are hierarchically structured. There exists one top level behavior and each behavior can create one or more subbehaviors to accomplish different tasks. Similar to the subsumption architecture [50], higher level behaviors allow for the execution of more abstract tasks (e.g., serve drinks at a cocktail party), while lower level behaviors are responsible for the more lower level control of locomotion and manipulation (e.g., avoiding an obstacle or opening a door). A behavior can retrieve observations from the environment using a central memory which is in turn populated by perception modules using low level sensory information.

For the purpose of this project, the behavior architecture has been modified to handle failing behaviors and is extended with two different failure recovery methods; one using only low level sensory information and another using a symbolic representation. The architecture is furthermore extended with a separate autonomous symbol perception module as described in the remainder of this thesis.

3.3 Robot Operating System

In addition to the behavior architecture, the Robot Operating System (ROS) framework [51][3] (specifically the Indigo Igloo release) has been used to facilitate the communication between different software modules (using ROS topics and services). A collection ROS software stacks (generic software components which use the utilities provided in ROS) are used for the purpose of navigation, perception and manipulation. The "move_base" package[4] is used for the purpose of Simultaneous Localization and Mapping (SLAM) [52] in which GMapping [53] is used to map the environment and Adaptive Monte Carlo Localization (AMCL) [54] for localization.

A detailed list of ROS packages as used and implemented for the purpose of the project is provided in Appendix B.

[2]http://enacer.com
[3]http://www.ros.org/
[4]http://wiki.ros.org/move_base

3.4 Human Robot Interaction

Using the Open Source speech recognition toolkit CMUSphinx [55][5], the robot is capable of understanding spoken sentences. As with many implementations of speech recognition software, it uses hidden Markov acoustic models [56] for the purpose of speaker-independent recognition of sentences. The multilingual Text-to-Speech Synthesis platform MaryTTS[6] has been used for the purpose of speech synthesis to the user.

3.5 Simulation

The experiments have been conducted using the Gazebo simulator [57][7]. Using a simulator makes the system more prone to failure in a real world setting (this is often referred to as the "Reality Gap"). However, running the experiments in a simulated environment allows for testing of more experiments in different conditions in less time compared to what would have been possible in a real world setting. Furthermore, the Gazebo simulator allows for different physics engines to be used and mimics the input and output of the robot as close to the reality as possible. The architecture is thus "unaware" of the fact that it is being run inside a simulator. Some randomization has been be applied to the structure and location of the environment and objects during the experiments. Figure 3.3 provides a screenshot of the simulation environment as seen in Gazebo and Figure 3.4 illustrates what the robot "sees" from it own point of view.

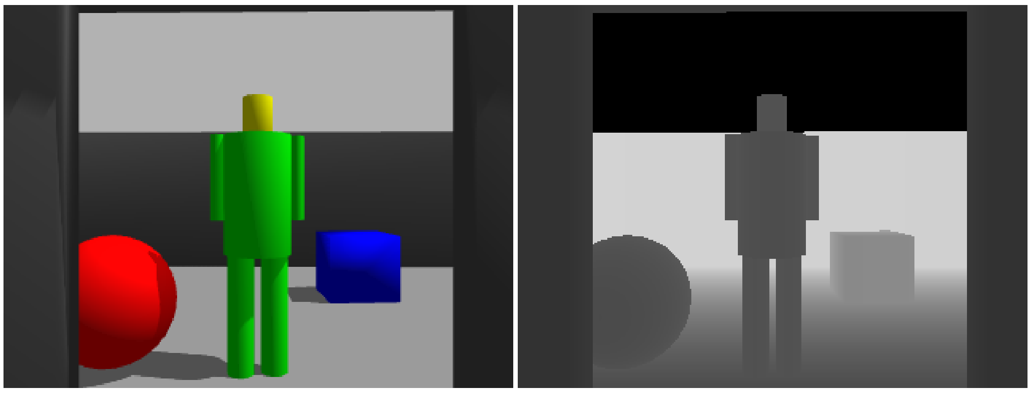


Figure 3.4: Point of view as seen by the robot during a possible failure scenario. Left; the color (RGB) image as seen by one of the RGBD cameras of the robot. Right; the depth (D) image as seen by one of the RGBD cameras of the robot.

[5]`http://cmusphinx.sourceforge.net/`
[6]`http://mary.dfki.de/`
[7]`http://gazebosim.org`

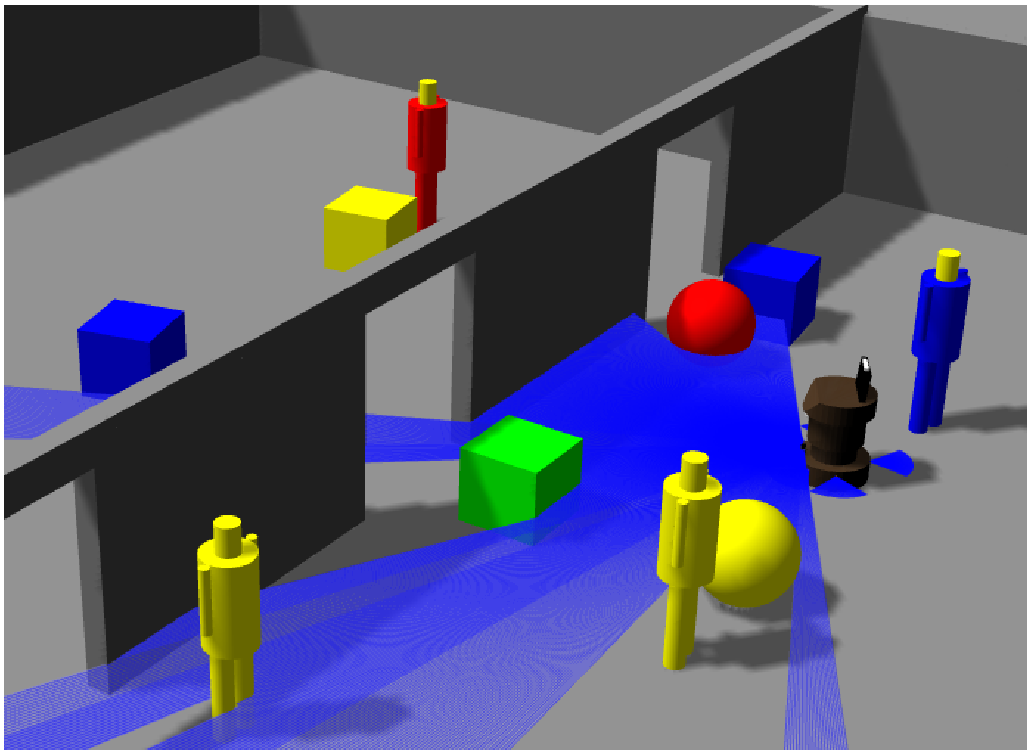


Figure 3.3: The 3-entrance simulation environment as seen in the Gazebo simulation. The goal of the robot is to enter the room using one of the three entrances. Each entrance could possibly be blocked by a different colored person, a box or a ball, causing the top level behavior (designed to enter the room) to fail. Given the location and color of the objects and persons, the robot must increase its overall recovery performance by learning the most optimal recovery solution (e.g., push, ask, continue or take an alternative route) in the fewest number of attempts. There exists one optimal recovery solution per failure situation which on average results in the highest reward. See Section 3.6 for more details.

3.6 Test Scenarios

The performance of the different methods is tested using three different sets of test scenarios. In each scenario, the aim is to model a situation in which a programmer has provided an initial solution (e.g., a top level behavior which is able to enter the room in most cases), while he did not account for all possible failures (e.g., objects and persons blocking the entrance, etc.), but does allow the robot to find new solutions whenever a (previously unseen) failure occurs using the methods described in the remainder of this thesis.

The basic setup of all failure scenarios is illustrated in Figure 3.5. The top level behavior of the robot aims to proceed from the start location to the target location. Different obstacles can be present which differ in type (either a box, a ball or a person), color (red, blue, green or yellow), location (either on the left, in the middle or on the right) and distance (either distant or nearby). At each entrance, there exists at most one obstacle per type (so at most three obstacles are observed per failure).

The entrance as well as the obstacles are either represented in a non-symbolic way (low level sensory information retrieved from the RGDB camera) or in a symbolic way (e.g., a sentence like "There is a red ball nearby on the left and a distant person in the middle."). In the symbolic representation, the obstacles are perceived as concept observations, in which each concept may have a different type (box, person or ball) and different properties (color, location and distance).

3.6.1 Required Recovery Solutions

At the occurrence of a failure (e.g., something is blocking the entrance), the robot may use any of the following recovery solutions to resolve the issue:

1. *Continue.*
   The robot may try to continue its original behavior in an attempt to gain entrance to the room. This solution is only useful if the failure has resolved itself (e.g., the obstacle moved just after the failure).

2. *Push.*
   The robot can try pushing against the object or person to gain entrance to the room.

3. *Ask.*
   The robot can try to ask for the object or person to give way to the robot.

4. *Alternative Route*
   Taking an alternative route using another entrance is a save option for the robot to use if it does not know any better solution. This does however cost more time and the robot might stumble onto another blocking obstacle at the alternative entrance.

The best recovery solution to use does not only depend on the type of obstacle, but also on the color and location of the obstacle. These dependencies and the best solution are not known to the robot: it must try to increase its performance with the fewest number of attempts. The performance of each of the methods discussed in the remainder of this thesis is tested using three different scenarios as described below. Each scenario increases the difficulty of finding the right recovery solution for a given failure and, as can be seen in the remainder of this thesis, tests different aspects of each method.

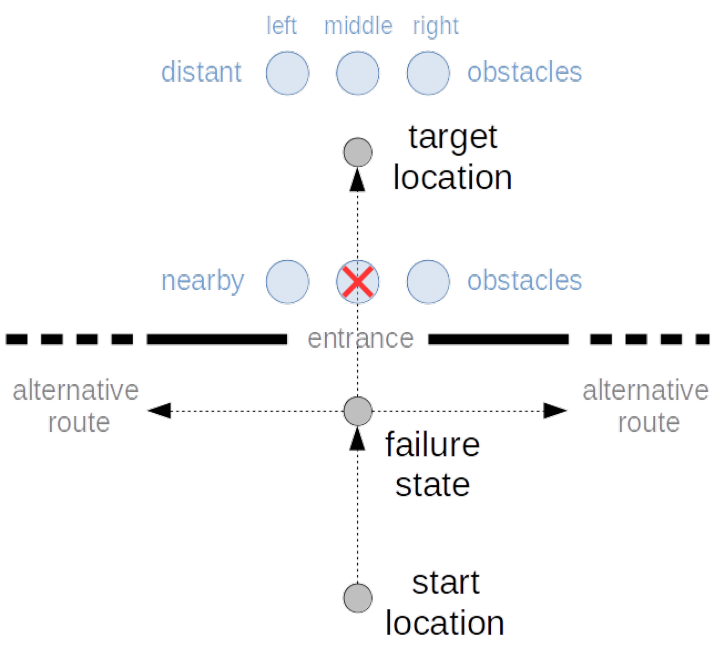


Figure 3.5: Schematic top-down overview of the simulated failure scenario used during experiments. The larger blue circles indicate possible locations for a concept (either a person, a box or a ball) to be present. Each concept may have four different colors (red, yellow, blue or green) and only one of each type may be present at each time (such that at most three unique concepts are present). Some uniform noise is applied to the location and orientation of the robot and any object or person. Only the location marked with a cross is relevant for the interpretation of the failure state. See text for more information.

3.6.2 Test Scenario 1; Basic Concepts

In this scenario there exists only one possible observable concept (either nothing, a box, a ball or a person), which blocks the entrance (marked with a red cross in Figure 3.5) and makes the top-level behavior fail. This results in the following combination of failure conditions and recovery solutions.

1. *There is no obstacle.*
   Best solution: The failure has resolved itself, *continue* top level behavior.

2. *There is a ball blocking the entrance.*
   Best solution: *Halt*, *push* against the ball and *resume* the top level behavior.

3. *There is a box blocking the entrance.*
   Best solution: *Cancel* the top level behavior and replace it with another top level behavior which uses an *alternative* entrance.

4. *There is person box blocking the entrance.*
   Best solution: *Halt*, *ask* the person to step aside and *resume* the top level behavior.

In each run there is a probability of 25% for any given solution to be a success if one would pick a solution at random.

3.6.3 Test Scenario 2; Different Colors

This test scenario is similar to test scenario 1, but now the observable concept may have four different colors; red, yellow, blue or green. For each unique combination of concept and color, a different solution may exist (either "push", "ask" or "taking an alternative route"). The exact combination of concepts, their colors and solutions, are uniformly randomized at each run. This with the exception of the case when there is no obstacle in front of the entrance, in which case the valid solution is always to "continue". In each run there is a probability of 25% for any given solution to be a success if one would pick a solution at random.

3.6.4 Test Scenario 3; Different Locations

This test is very similar to test scenario 2 with the same randomized combination of failure conditions and recovery solutions. However, now there may exist multiple observable concepts each having a different location. These locations are marked as the bigger blue circles in Figure 3.5. Only the obstacle marked with the red cross is responsible for blocking the entrance, all other observable concepts ought to be ignored by the robot. This scenario is expected to be especially difficult to solve using the symbolic representation, because in this scenario, the robot must infer the fact that only an obstacle located nearby and in the middle matters. In each run there is a probability of 25% for any given solution to be a success if one would pick a solution at random.

3.7 Performance Measure

Each recovery method discussed in the remainder of this thesis is tested using the test procedure mentioned below. The mean performance of each method is calculated over multiple independent runs. Each test for each method consists of 1000 runs. The order of failures is randomized for each run in which there is an equal uniform probability for each solution to be a success. This means that if one would pick a solution at random each time, the mean performance would be around 0.25.

A single run consists of multiple attempts in which the robot tries to recover from a single failure of the top level behavior. At each successive attempt, the robot gains more experience, thus allowing to gain a higher overall performance over the course of all attempts. This is a form of online machine learning, since the data is presented in sequential order and there is no explicit training phase. For all tests, a single run consists of 200 attempts.

It is not very informative to compare the performance of different recovery methods in terms of their reward as described in Section 4.2, especially since taking an alternative route, may still result in a very low reward. We are more interested in whether the method picked the best recovery solution or not for a given failure. We therefore record the performance of a single attempt as either being a one or a zero. The performance is a one if the method has picked the best recovery solution (as described in Section 3.6) and zero otherwise, even if this results in a non-zero reward.

For a given number of attempts (up to 200), the mean performance $(0-1)$ of all 1000 tests is calculated. This results in a learning curve, in which the mean performance increases from zero to (almost) one. The performance can also be seen as a measure of probability for the method to pick the best solution at any given number of attempts experienced in the past. A good recovery method is able to reach a mean performance of one using the *fewest number of attempts*. It is important to note that all methods still use the original rewards for learning. The $(0-1)$ performance measure is only used to compare the different recovery methods in a meaningful quantitative manner.

3.8 Dataset Generation

With the number of recovery methods to test (see Figure B.1) and total number of runs to execute in order to calculate the mean performance (see Section 3.7), it is impractical to execute all tests in the Gazebo simulator (see Section 3.5). For this reason, a separate dataset has been generated to test each variation of recovery method. The dataset consists of 2340 failure situations in which the top level behavior fails to enter the room. At each failure situation, in which the robot looks at the entrance and sees any observable concepts, a snapshot of all raw sensor data is stored. The dataset also includes ground truth information about the environment at each failure, such as the exact state of the observable concepts. This ground truth information is used to infer the best recovery solution during performance measuring (see Section 3.7) which the recovery method (with no access to this ground truth information) should have taken. Some uniform noise is applied to both the location (-0.25 to 0.25 meters) and orientation (-0.5 to 0.5 radians) of the observable concepts and the robot. During the experiments, the architecture is used the same way as it would run in the Gazebo simulator. However, the top level behavior now fails without actually moving towards the entrance. The raw sensor information which is presented to the perception modules is retrieved from the dataset.

3.8.1 Simulated Rewards

During each attempt of recovering a failing behavior, the reward (see also Section 4.2) for taking a given recovery technique and action a_r in a failure state s_f resulting in the final state s, is calculated as follows (in which $\mathcal{U}$ is the continuous uniform distribution for the purpose of introducing noise):

$$R(s_f, a_r, s) = \begin{cases} \frac{1}{d+\mathcal{U}(-\alpha,\alpha)} & \textbf{If} \text{ the recovery attempt succeeds with probability } p. \\ 0 & \textbf{If} \text{ the recovery attempt fails.} \end{cases}$$

Here, for each recovery solution (see Section 3.6), the base duration d, noise factor α and success probability p are defined as follows:

	d	α	p
Continue:	7.0	2.5	0.9
Push:	10.0	2.5	0.9
Ask:	10.0	2.5	0.9
Alternative Route:	15.0	5.0	0.5

These numbers are based on actual attempts run in the Gazebo simulator and calculated using the formula mentioned in Section 4.2. The order of prerecorded failure situations are randomized for each run.

Chapter 4

Ground Truth Failure Recovery

The easiest form of failure recovery is the one in which ground truth information of the environment is used. In such case, the robot is told (in the form of a unique label) what the current failure is. There is therefore no need for the robot to learn or recognize any failures itself. If there is no credit assignment problem [58] and we assume the failure state to be known (either by information provided internally or externally), the solution is straightforward and similar to solving the k-Armed bandit problem [59, 60] since the robot only has to learn the best recovery solution for a known failure state.

This method is expected to yield the best recovery performance in comparison to the other methods discussed in this thesis. It is, however, also an unrealistic representation, since in practice the robot never knows the exact failure state beforehand (unless it is told by a human or some fault detection module). Failure recovery using the ground truth representation of the failure is used as the baseline for comparing the performance of all failure recovery methods.

The following sections describe the different failure recovery techniques and exploration schemes, used in the behavior architecture, in more detail. The same behavior architecture is also used when either the non-symbolic (Chapter 5) or symbolic (Chapter 6) representation is used. Sections 4.5 and 4.6 discuss the actual results of the failure recovery method solely using ground truth information in each of the three test scenarios described in Section 3.6.

4.1 Failure Recovery Techniques

The main goal of the proposed method of recovering failing behaviors, as described in this thesis, is to allow the system architect to identify behaviors prone to failures and specify a set of possible recovery solutions whenever the behavior fails. This allows the system architect to not account for all possible failure conditions manually, but rather to have the robot learn the best solution for a particular failure autonomously.

In this project we limit ourselves to failures that occur within the behavior architecture as a result of an external anomaly in the environment (see Section 2.1.2). Here we assume that the behavior architecture is capable of executing a collection of tasks successfully most of the time (e.g., navigating to a specific location, fetching an object from a room, searching for person, serving drinks, etc.), but that due to changes in the environment (e.g., blocking of entrances, displacement of objects, etc.), demands of the user (e.g., changing preferences, different usability constrains, etc.) or other unforeseen circumstances, previously successful behaviors start to fail more frequently.

For the purpose of failure recovery, the architecture allows behaviors to either:

1. **Fail and give up** on achieving the goal of the failing behavior completely. This requires the parent behavior (the one initiating the behavior) to cancel the task or provide a custom (hand-coded) solution in resolving the issue.
2. **Resolve the failure autonomously** using one of the following techniques:
   (a) **Continue** the original behavior. In some cases the failure has resolved itself, and no special action is needed except to continue.
   (b) **Cancel and replace** the failing behavior with an *alternative behavior* in an attempt to resolve the failure. The alternative behavior can be of the same type as the original behavior but with different initial parameters set. Examples include taking an alternative route when trying to enter a room or trying a different grasping technique if the object cannot be picked up at the first attempt.
   (c) **Halt and resume** the failing behavior. Here the failing behavior is halted temporarily, and a specific *recovery behavior* is executed to resolve the failure in place. Once the recovery behavior has succeeded, the original behavior is (unlike the "cancel and replace" technique) allowed to be resumed. Examples of halting the original behavior and executing a recovery behavior include pushing against a door in order to open it while entering the room or removing clutter from a surface in order to free an object and allow it to be grasped by the original behavior.

For a given failure-prone behavior, the architecture allows the designer to specify a set of alternative behaviors and recovery behaviors to be utilized by respectively the "Cancel and replace" technique and the "Halt and resume" technique mentioned above. The methods described in the remainder of this thesis, use these behaviors to autonomously learn the best technique ("Continue", "Cancel and replace" or "Halt and resume") as well as the best behavior to execute in order to maximize the probability of resolving the failure.

4.2 Rewards

The methods described in the remainder of this thesis, use a reward or score to credit the provided solution in resolving the failure. The reward function $R(s_f, a_r, s)$ for taking a given recovery technique and action a_r in an unknown failure state s_f resulting in the final state s is defined as follows:

$$R(s_f, a_r, s) = \begin{cases} \frac{1}{d_o} & \textbf{If } \text{the failing behavior is succesfully recovered using the "Halt and resume" or "Continue" technique.} \\ \frac{1}{d_o+d_a} & \textbf{If } \text{the failing behavior is succesfully recovered using the "Cancel and replace". technique.} \\ 0 & \textbf{If } \text{the failure could not be resolved using any of the techniques.} \end{cases}$$

With d_o being the duration (in seconds) of the original failing behavior and d_a the duration of the alternative behavior. The efficiency of the solution is thus measured in terms of the time it takes to recover from the failure. Here the best action yielding the best estimated reward (the mean reward for a given training set for each failure state) is chosen during exploitation (after training).

4.3 Exploration Schemes

Without exploration, the system cannot learn all possible solutions efficiently. It is required to succeed as well as fail in order to identify the best possible recovery solution from the full set of possible behaviors. However, a balance must be found between exploration and exploitation to avoid failing too much in general but still be able to find an optimal solution as soon as possible. For this purpose, the behavior architecture offers several exploration schemes [61] to be utilized by the designer. The utilities of these different exploration schemes are evaluated in more detail in Section 4.5.

4.3.1 "Naïve"

Here each possible combination of a failure state and recovery action is first tested for a fixed number of times during the training phase. Then, during the exploitation phase, the best estimated reward is chosen for a given combination of a failure state and recovery action thereafter.

In many cases the Naïve method does not seem adequate to be used since a domestic service robot is expected to operate in a changing environment in which continuous learning is required. However, a specific exploration phase might be beneficial to quickly bootstrap the system in the early stages of development.

4.3.2 ϵ-Greedy

With this exploration scheme, the system explores after each failure with probability ϵ and exploits after each failure (i.e., be greedy) with probability $1 - \epsilon$. This allows continuous learning, but at the cost of a lower overall performance if the environment does not change.

4.3.3 Interval Estimation

This exploration technique is based on the work discussed in [62, 63] which has in turn been inspired by [64]. Here, for each combination of a failure state and a recovery action, the exploration scheme calculates a confidence interval of all previous experienced rewards. After each failure, the method chooses the recovery action with the highest upper confidence interval. This results in the method exploring relatively untested combinations of failure states and recovery actions (i.e., those with a large confidence interval) more often in the early stages of development. In contrast, the method starts to exploit more often, the more experience has been accumulated at later stages of development (i.e., when the mean confidence intervals for each combination become small and more towards the mean of the reward distributions).

4.4 Gaining Experience

During its lifetime, the robot gains experience by storing information at each failure. This includes information such as the name of the failing behavior, configuration parameters being used, the solution which has been selected and the reward after execution of the solution. Furthermore, depending on the method being used, either symbolic or non-symbolic information is stored for the purpose of failure recognition in future times.

4.5 Results

The results of the failure recovery using ground truth information are shown in Figure 4.1. Using the Naïve exploration scheme, the method is allowed to explore for either 25 or 100 attempts, after which it will solely exploit and try to perform as well as possible. In the ϵ-Greedy exploration scheme, ϵ is set to 0.05, meaning that the method will explore and pick a random solution in 5% of all attempts. In case of the Interval Estimation exploration scheme, α is set to 0.05 to select the upper bound of the $100(1-\alpha)$ confidence interval.

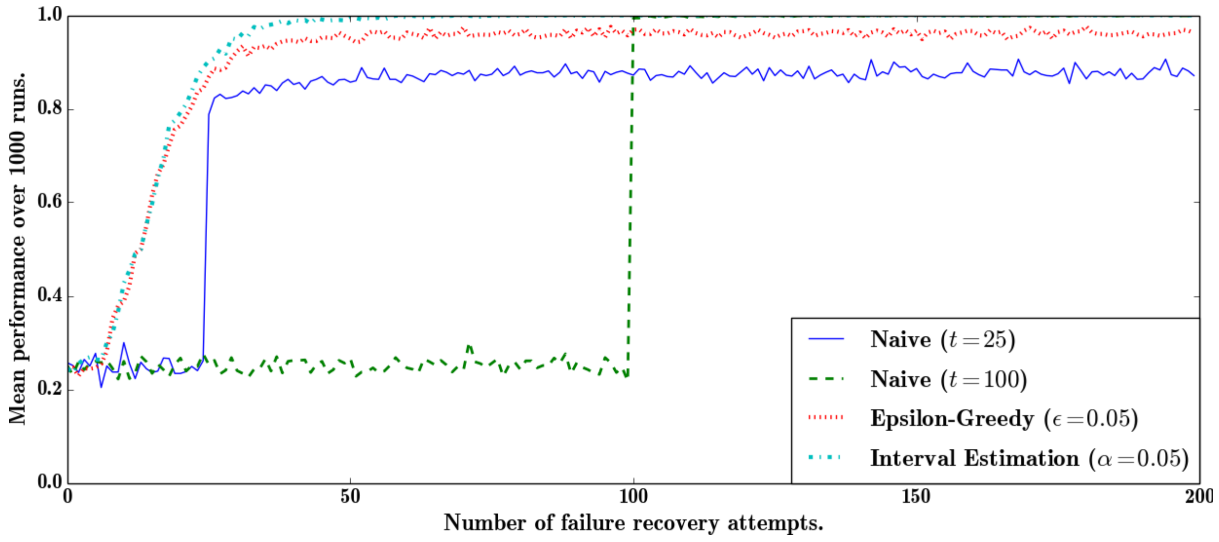
The results of test scenario 1 clearly show that all exploration schemes are able to reach a good performance of more than 0.8. Results of the Naïve exploration scheme indicate that initial training for an arbitrary number of attempts may lead to a suboptimal performance during exploitation. This suggests that good balance between continuous exploration and exploitation is indeed required. The ϵ-Greedy exploration scheme performs well, but is never able to reach a mean performance of 1.0, due to random exploration in 5% of all attempts. The Interval Estimation exploration scheme, however, is able to reach a mean performance of 1.0 once the confidence interval for each failure state shrinks, thus allowing it to use the mean expected reward with a minimal bias from the true mean.

The results in test scenarios 2 and 3 clearly show that the method starts to struggle to reach a good performance once the complexity of the environment increases. This can be explained by the fact that the total number of possible failure states increases significantly with the increase of complexity of the environment. Where in test scenario 1, there were only 4 possible failure states, test scenario 3 has 13 possible failure states. With the added possibility to have extra visible concepts of different types and colors, test scenario 3 has a total of 741 possible failure states, thereby diminishing the performance of the method significantly.

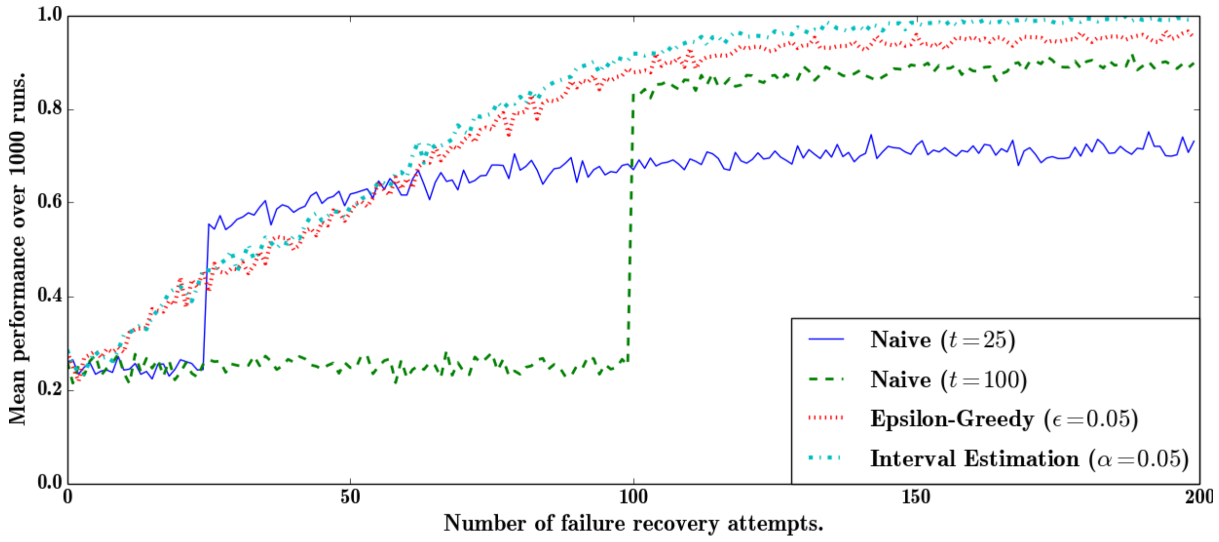
4.6 Discussion

Among all exploration schemes, Interval Estimation performs best and avoids performing worse due to excessive exploration at a later stage. However, the amount of exploration is unpredictable since it very much depends on the training sample size and the $100(1-\alpha)$ confidence interval. The ϵ-Greedy exploration scheme is therefore used instead for the Ground Truth failure recoverer and Non-Symbolic recoverer in the remainder of this thesis.

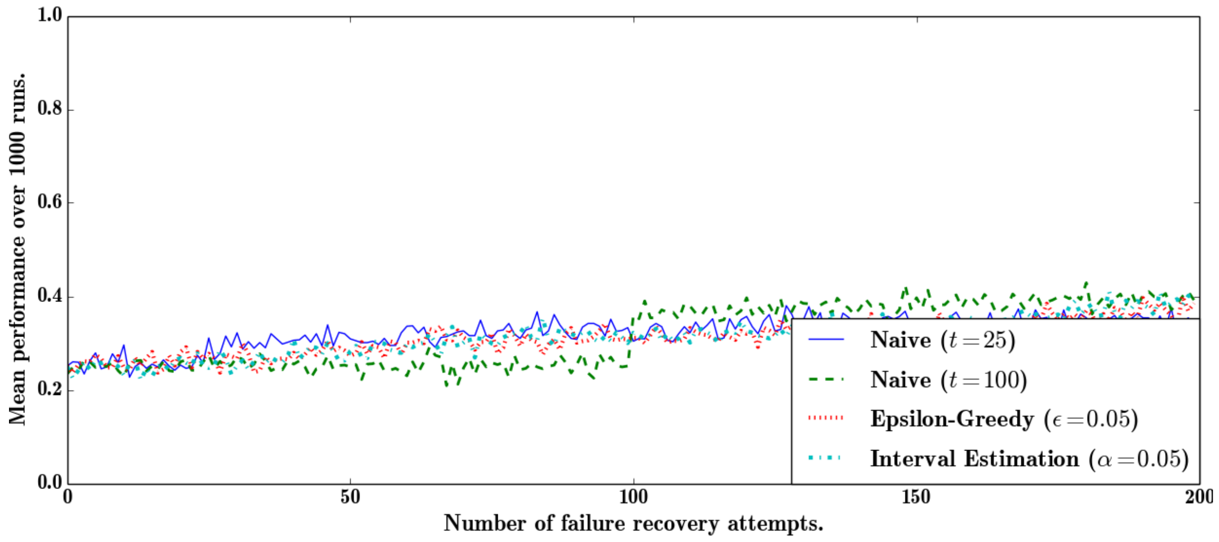
Results from test scenarios 2 and 3 clearly show, that even if perfect state information is known to the robot, learning failure recovery solutions on the exact state of the failure is impractical. The failure state space is therefore required to be reduced to a lower dimensional state space in which there are more training samples to be utilized per state. This suggests that both the non-symbolic and symbolic methods of failure recovery, as discussed in the remainder of this thesis, require to find the right level of abstraction in their attempt to make sense of the failure situation.



(a) Test Scenario 1.



(b) Test Scenario 3.



(c) Test Scenario 4.

Figure 4.1: Learning curves for different exploration schemes when ground truth information is being used (see Chapter 4 for details).

Chapter 5

Non-Symbolic Failure Recovery

As shown in Chapter 4, once the failure state is known, it becomes relatively easy to learn the best solution to a given failure situation. This is especially true if the total set of possible failure states is small (see Chapter 4). However, in practice the robot does not know what type of failure has occurred, it only knows that *a* failure has occurred. Furthermore, the set of all possible failures is also not known: the current failure could be a *previously unseen* type of failure, in which case the robot cannot utilize everything it has learned in the past.

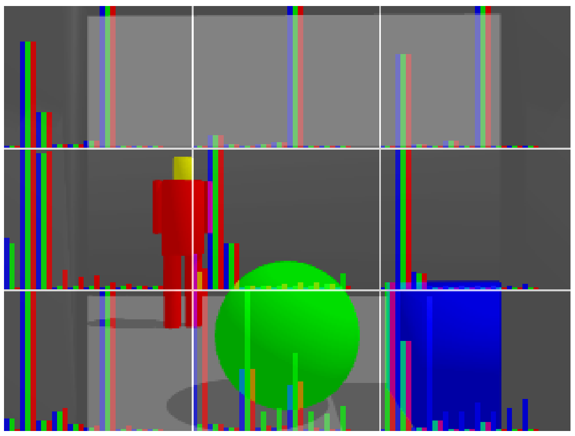
This section proposes a method of learning how to resolve failures, without a pre-learned model on how to do so. Furthermore, the robot is required to only use low level sensory information (such as retrieved from a color and depth camera) to create some sense of what type of failure state it is in. This also requires the method to make an optimal selection of its past experience (as accumulated by previous attempts to resolve failures in the past) which most likely belongs to the same type of failure it is currently facing. Moreover, the robot should be able to cope with *unseen* types of failures (after extensive learning) and allow for efficient exploration for these new type of failures using the methods provided in Section 4.3.

5.1 Low Level Sensory Information

For the purpose of this research, the robot is allowed to extract low level sensory information from the color and depth camera of the RGDB sensor (see Section 3.1). At any given failure, a snapshot of both the depth and color image is stored. These two images are divided in nine consecutive areas as shown in Figure 5.1. From each area a binned histogram is calculated from the depth image and from each channel (red, blue and green) in the color image. Each histogram in the depth image has 20 bins while each histogram of each channel in the color image has 10 bins. After normalization of each histogram, all histograms are merged into a single 450-dimensional feature vector. This feature vector is used to calculate a dissimilarity measure as discussed in the following section.

5.2 Dissimilarity Measure

During exploitation, when the robot should perform at its best, it seems sensible to assume that the best recovery solutions to choose, can be found using its experience which is most similar to the current observed failure. For this purpose, the method calculates and orders all training samples (its experience, as described in Section 4.4) in terms of increasing dissimilarity. Figure 5.2 illustrates an example of such selection relative to the current observed failure.


(a) Example histograms of the color image extracted from nine consecutive areas.

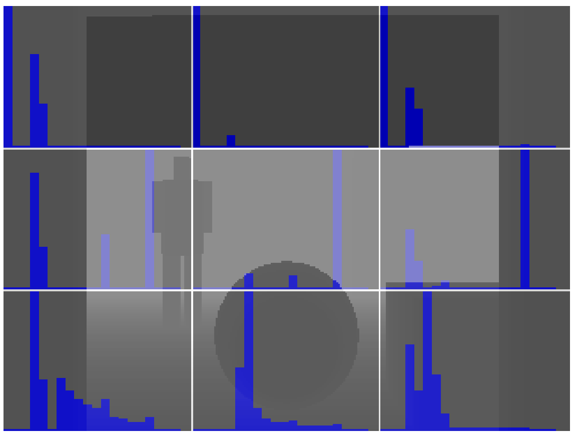

(b) Example histograms of the depth image extracted from nine consecutive areas.

Figure 5.1: Example snapshot of the low level sensory information as retrieved during a single observation of a failure situation. Both the color and depth image are segmented in nine consecutive areas. From each area an x-binned normalized histogram is calculated for each channel. The color image has three channels (RGB) while the depth image has one channel (a gray-scale value).

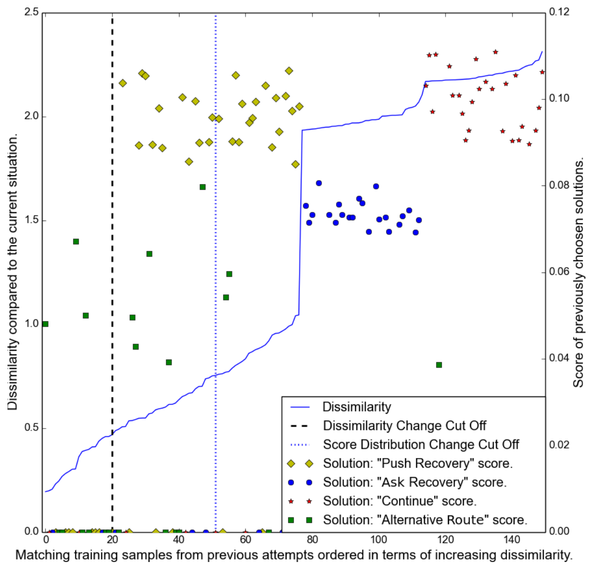
For low level sensor information, one can estimate a dissimilarity measure by calculating the euclidean or Mahalanobis [65] distance between the current feature vector and all other feature vectors experienced in the past. For more abstract observations, in which the ability to quantify each observation is limited to a boolean value, a more generic distance measure such as the Tanimoto coëfficient [66] can be used. Since we assume as little as possible about the types of failure situations, low level sensory information (see Section 5.1) in combination with the euclidean distance (to calculate the dissimilarity measure) has been used in this research.

5.3 Experience Selection

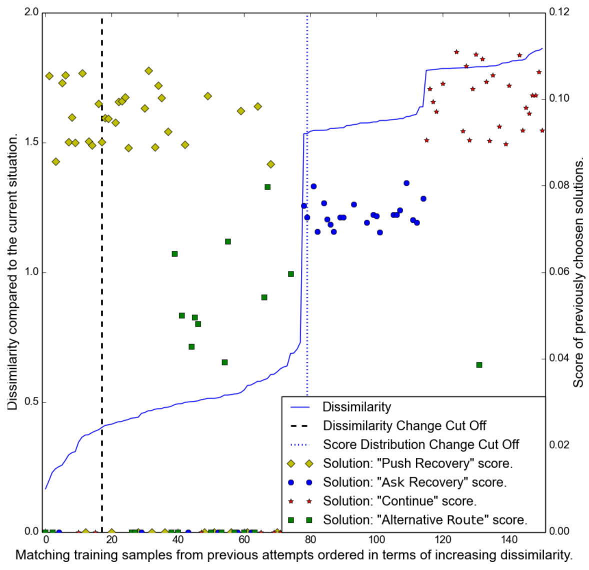
Similar to using the k-Nearest Neighbor algorithm [67], one could simply pick the k most similar training samples and pick the recovery technique and action with the highest mean expected reward. However, choosing a random k is prone to errors since the sample distribution changes over time and differs from one failure state to the other.

An alternative solution is to use cross-validation [68] for different numbers of k over the dataset to determine the best k to use. However, apart from being computational expensive, this is also expected to be suboptimal since the best k depends very much on the different sized and shaped failure state distributions in the training set.

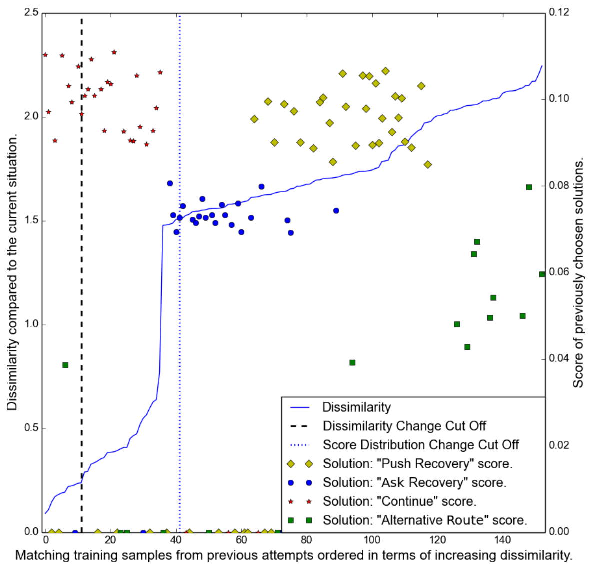
A better alternative, as described in the following section, is to choose k on the fly the moment a failure occurs and select just the right amount of experience to utilize. This is especially important for the Interval Estimation exploration scheme (see Section 4.3) in which the confidence interval will otherwise be too small for novel failures if k is too large.



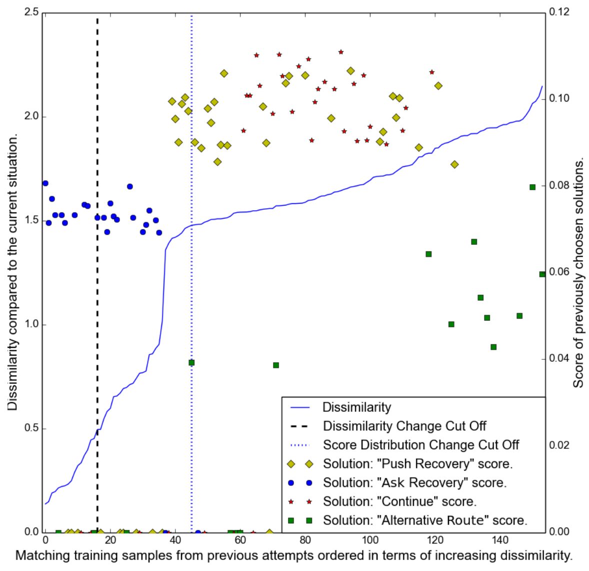
(a) Example dissimilarity graph of a failure state in which the "Alternative Route" is the best solution to use.



(b) Example dissimilarity graph of a failure state in which the "Push Recovery Behavior" is the best solution to use.



(c) Example dissimilarity graph of a failure state in which the "Continue Current behavior" is the best solution to use.



(d) Example dissimilarity graph of a failure state in which the "Ask Recovery Behavior" is the best solution to use.

Figure 5.2: Examples of different dissimilarity graphs. Each graph illustrates the dissimilarity (*left* axis) of all training samples compared to the current failure state being observed. Each diamond, circle, star or square represents the score (*right* axis) of taking a specific recovery solution. Recovery actions that have failed have a score of 0.0 (visible at the very bottom of the graph). The vertical lines represent two of the possible cut-off selection criterion, see text and see Section 5.4 for more details. The examples shown here have been generated from a single random test using test scenario 1 as discussed in Section 3.6.2. For each of the possible failure states used in the experiments, there exists one recovery action (as described in Section 3.6.1) which provides the optimal solution. The solution can be found as a single grouping of a single type of recovery action most to the left side of the graphs (those with the least amount of dissimilarity) with the highest mean reward. The method estimates this recovery action by taking the maximum mean score over all scores *before* the lowest cut-off selection criterion (all samples before the most left positioned vertical line).

5.4 Failure State Estimation

Once a proper dissimilarity measure is used, in which preferably variance, feature selection and normalization are taken into account, one can expect a given failure state to consist of one or more separable clusters, in which each cluster represents a group of similar experiences. A traditional unsupervised learning method such as k-Means [69] (a Vector Quantization method) or Hierarchical Clustering [70] could be used to identify these clusters autonomously. The use of the resulting clusters as a possible selection is expected to perform better than taking an arbitrary or maximum k, but still suboptimal since the number of clusters (and thus failure states) is unknown. This in turn requires intensive cross-validation[1] for these clustering algorithms to work.

An alternative solution as proposed in this thesis, first regards all matching experiences (ordered in terms of dissimilarity as seen in Figure 5.2) as an initial cluster, and then removes a specific range of samples (with an overall higher dissimilarity) from the cluster based on the following two cut-off criteria;

1. a sudden increase in dissimilarity (the dashed horizontal line in Figure 5.2), or;
2. a sudden change in the distribution of scores (the dashed horizontal line in Figure 5.2).

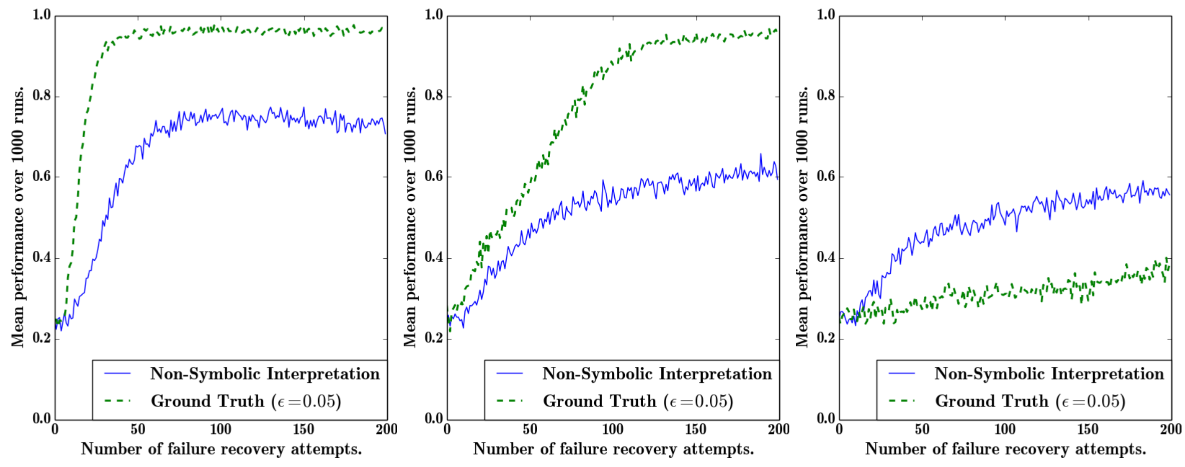
For both cut-off criteria, a sliding window (with size 10) is used to determine where either the samples themselves become too dissimilar or where the score distribution becomes too dissimilar compared to the samples within the window. A more detailed description of this algorithm is illustrated in appendix A. A similar algorithm is used for the determining the selection criterion based on the dissimilarity of the samples themselves. Figure 5.2 gives an example of the selection procedure for different failure state instances. The lowest cut-off criterion (e.g. the one most to the left in Figure 5.2), is used for the final selection and is regarded as belonging to the same cluster and thus the same failure situation.

The window size m needs to be chosen carefully and should as a rule of thumb be at least the number of recovery action types to the power of two. A too small window size is preferred over a too large window size since taking a bit of all correct information is always better than taking all the correct information in addition to a lot of false information.

[1]Such as testing the algorithm for different numbers of clusters and determining the goodness of the fit using a measure such as the chi-squared and Kolmogorov-Smirnov statistics [66].

5.5 Results

The results of using failure recovery using low level sensory information in the different test scenarios is illustrated in Figure 5.3. In test scenario 1, ϵ-Greedy using Ground Truth information easily outperforms the non-symbolic failure recovery method. As the complexity of the environment increases in test scenario 2, the performance of both methods drops. However, the performance drops significantly less for the non-symbolic failure recovery method when the complexity increases even more in test scenario 3.



(a) Test Scenario 1. (b) Test Scenario 2. (c) Test Scenario 3.

Figure 5.3: Learning curves in the different test scenarios for the non-symbolic recovery method (see Chapter 5 for details).

5.5.1 Incremental Failure States

Another interesting case is to see how the method performs when the set of all possible failures changes over time. This new test scenario is similar to the first test scenario mentioned in Chapter 3, but with the difference that new previously unseen failure states are introduced to the robot slowly one by one. For each test, the set of all possible failure states increases by one additional type of failure after every 70 recovery attempts. This means that it becomes increasingly more difficult for the robot to provide the optimal solution the more failure states are introduced. Furthermore, the distribution of the probability of possible failures changes over time. This means that the method should thus be robust enough to cope with previously unseen failure states even after significant learning has occurred.

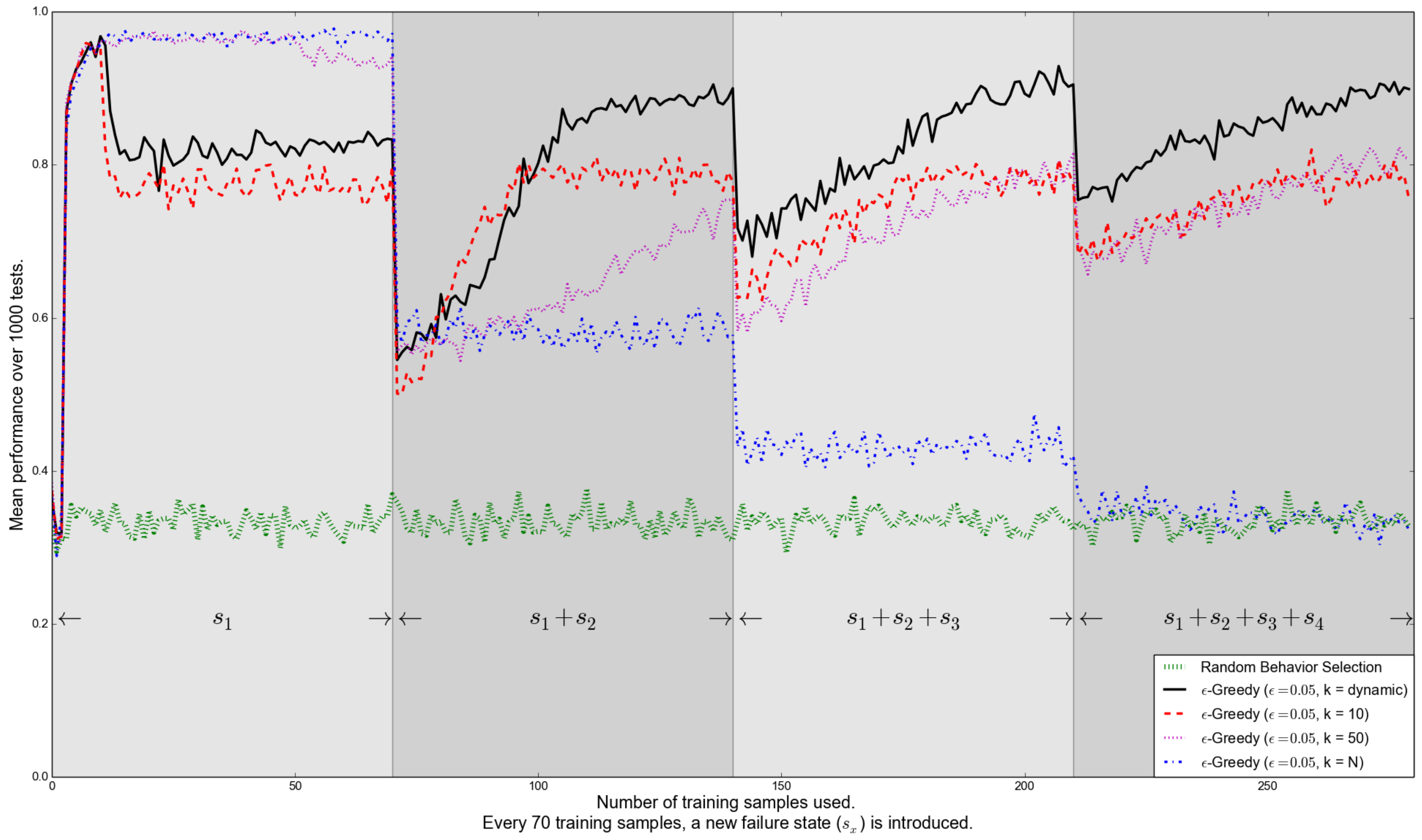


Figure 5.4: Learning curves for different experience selection methods. Every 70 recovery attempts (each shade of gray), a new previously unseen failure state is introduced to the set of possible failure states. The exact combination and order of failure states is randomized at each test. There is no unique number (k) of training samples to use which yields an optimal performance. The method of dynamically selecting just the right number of training samples ($k = dynamic$), outperforms all other methods. The method needs at least 10 training samples before experience selection is enabled. This explains the initial high performance for all methods (that temporarily behave similar to $k = N$). See text for more information.

The performance of the method is shown in Figure 5.4. As expected, the mean performance quickly drops after the introduction of a previously unseen failure state. If the complete set of matches is used ($k = N$), a performance of almost 1.0 is easily reached if there is only one possible failure state (s_1). However, if there is more than one failure state, the performance quickly drops since taking the mean best action does not provide a good overall solution. This thus shows, that it is essential for the method to discriminate from one failure state to the other. If one uses a fixed number of matching samples to use ($k = 10$ or $k = 50$), the performance is significantly increased compared to using all matches ($k = N$). However, using a fixed number, introduces the risk of either using too much experience (including those not related to the current failure situation) or too little. Only by using a dynamic number of matching samples ($k = dynamic$, see Section 5.4 on an explanation on the dynamic selection of matching samples), it is possible to reach a more optimal performance of about 0.9.

5.6 Discussion

The implementation is similar to the k-Nearest Neighbor algorithm [67], in which the k most similar training samples are used to estimate the best recovery solution. However, in the proposed method, k is dynamically chosen with the aim to include just the right number of training samples most similar to the current failure situation. Unlike many traditional unsupervised clustering methods, the method only identifies the first most similar cluster of training samples without making any further assumptions such as the total number of clusters (or type of failures) and statistical properties of the distribution (see Section 5.4).

Results in Figure 5.4 show that the proposed method is effective compared to the baseline of randomly selecting a recovery solution or simply picking the one with the mean best reward. The dynamic selection of k-closest training samples has also proven to be more effective than using a fixed number for k during online learning. Furthermore, the method has proven to be robust against the introduction of new failure types even after extensive learning has occurred. However, the method is not able to reach a final performance of anything near 1.0, even in the simpler first test scenario.

Learning faster and with a higher final performance requires the need to: a) explore for new solutions at the right moment and b) discriminate between different types of failures more clearly. Better discrimination is possible by using more sensor information and better feature selection. However, in order to overcome the curse of dimensionality and reduce the computational load on calculating the dissimilarity between failure situations, more abstract observations (such as the observation of different objects and locations in the environment) as preprocessed and provided by other perception modules, could be used. Better exploration could be improved by modifying the ϵ-Greedy exploration method in such a way that the probability of exploration depends on how dissimilar the current situation is compared to the full set of known training samples (i.e., the robot's experience). This might however decrease the initial performance of the method since exploration is preferred whenever a new type of failure is introduced, but is expected to yield a better performance in the long run.

Chapter 6

Symbolic Failure Recovery

Failure recovery using low level sensory information (see Chapter 5) has shown to be effective, if and only if sufficient training data can be collected. However, this means that in practice the robot will have to fail a lot before it is able to collect sufficient training data to achieve its optimal performance of resolving (known) failures[1]. This is impractical for a domestic service robot which from the start ought to operate at its best with minimal efforts in recovering from new failures.

Furthermore, small changes to the environment have little impact on the context of the failure, but may be of large influence on the low level sensory representation. The latter greatly increases (for the wrong reasons) the dissimilarity of the new change compared to similar failures experienced in the past, thus requiring the method to retrain what it has learned before for every new condition.

The low level sensory representation suffers from the same old problem: the curse of dimensionality, and as a result, data starvation. The solution to the problem is to lower the dimensionality, increase the level of abstraction of the representation and thereby to lower the entropy of this representation. This section proposes a method of failure interpretation and recovery using a symbolic representation with a very high level of abstraction compared to the low level sensory representation.

6.1 Symbolic Representation

The symbolic representation used in this project aims at being as close to human language as possible, in order for both the robot as well as the user to understand the failure state and solution as well as possible. For the purpose of this project, we limit the symbolic representation to nouns and adjectives. Here, nouns (box, person or ball) are represented as a collection of observable *concepts* and adjectives (red, green, blue, yellow, distant, nearby, middle, right or left) as the *properties* of these concepts. The concepts and their properties correspond to the simulated failure scenario shown in Figure 3.5 in Chapter 3.

Using the symbolic representation is expected to yield a better performance compared to using solely the low level representation. However, the symbolic representation is useless without proper symbol grounding. That is, the meaning of the symbols must be grounded using the perceptual capabilities of the robot. Chapter 7 provides a solution to this problem in which the robot is able to detect and classify observations of concepts and their properties in the environment autonomously. In this chapter, recovery performance results are shown for both with and

[1]This takes at least 100 recovery attempts for four different failures using the solely low level sensory representation as can be seen in Chapter 5.

without autonomous symbol perception (see Chapter 7 for more details). Using autonomous symbol perception does increase the probability of the symbolic representation to be (to some degree) incorrect.

6.2 The Bayesian Approach

A simple, yet effective method of symbol interpretation is to use a modified Naive Bayes classifier [71] similar to what often has been used for the purpose of text classification such as e-mail spam filtering [72, 73]. Here, given the total set of all possible solutions S and the current set of unique observations O, we seek to find the recovery solution $s \in S$ with the highest probability of being a success:

$$p(s) = \arg\max_{s \in S} \big(p(s|O)\big) \tag{6.1}$$

Here, each observation $o \in O$ is a single unique textual representation of an observable concept and its properties (e.g., a “distant blue box on the right” is represented as “distant-blue-right-box”). The conditional probability $p(s|O)$ of s being a success, given observation O, is calculated using:

$$p(s|O) = \frac{\prod\limits_{o \in O} p(s|o)}{\prod\limits_{o \in O} p(s|o) + \prod\limits_{o \in O} \big(1 - p(s|o)\big)} \tag{6.2}$$

The conditional probability $p(s|o)$ for a solution $s \in S$ to be a success given the presence of some observation $o \in O$ is calculated using the Naive Bayes rule:

$$p(s|o) = \frac{p(o|s)p(s)}{p(o|s)p(s) + p(o|u)p(u)} \tag{6.3}$$

Here, the conditional probabilities $p(o|s)$ and $p(o|u)$ of the solution s to be successful or unsuccessful respectively, given the observation o, is calculated using the expected value of the score $\xi(s)$ to be either higher than or equal to 0.0:

$$p(o|s) = E\Big(o \wedge \big(\xi(s) > 0.0\big)\Big) \tag{6.4}$$

$$p(o|u) = E\Big(o \wedge \big(\xi(s) \equiv 0.0\big)\Big) \tag{6.5}$$

The prior probabilities $p(s)$ and $p(u)$ of any solution s to be successful or unsuccessful respectively, is calculated using the expected value of the score $\xi(s)$ being higher than 0.0, independent of the presence of any observation:

$$p(s) = E\big(\xi(s) > 0.0\big) \tag{6.6}$$

$$p(u) = 1 - p(s) \tag{6.7}$$

Here, the conditional probability $p(o|s)$, of observing o given that a solution s is a success, is corrected by the subtraction of $p(o|u)$ into $p^*(o|u)$ in Equation 6.8. Here, $p(o|u)$ is the conditional probability of observing o given that a solution s is unsuccessful. This allows the algorithm to ignore observations that correlate less with the solution s being a success.

$$p^*(o|s) = [0, p(o|s) - p(o|u), 1] \tag{6.8}$$

6.2.1 Results

The learning curves of the Bayesian approach in using the symbolic representation, can be seen in Figure 6.1 for all three test scenarios. In test scenario 1 and 2, the performance of the Bayesian approach of using the symbolic representation is similar to that of using ϵ-Greedy using Ground Truth information (Chapter 4). Interestingly, in test scenario 3, the Bayesian approach outperforms ϵ-Greedy using Ground Truth information and almost reaches a similar performance as the Non-Symbolic interpretation method (see Chapter 5).

6.3 Dynamic Representations

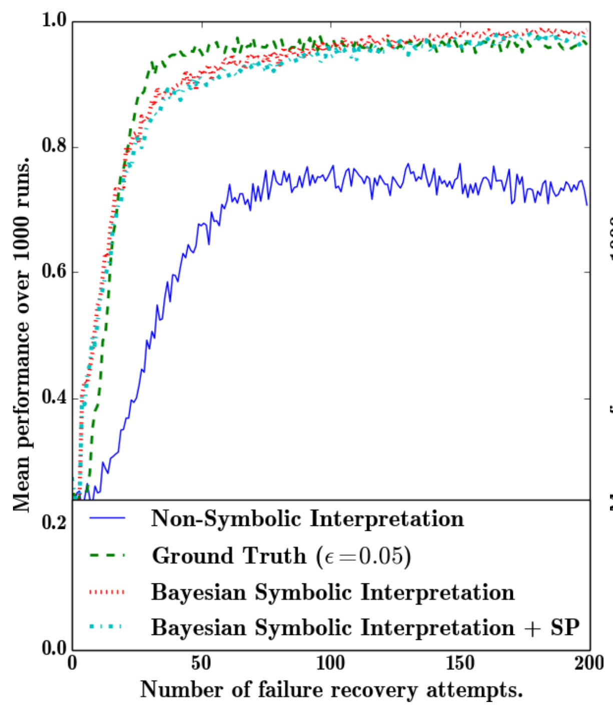
Many variations in the properties of the concepts result likewise in many variations of possible symbols when using the Bayesian approach of symbolic interpretation. This is especially true for test scenario 3 in which not only the color differs but also the location and distance of the concept. This explains why the results of the Bayesian approach of symbolic interpretation performs poorly in test scenario 3 (see Figure 6.1). This problem can be solved by transforming the original set of observations O into a new representation $r(O)$ in which there are less variations of symbols possible, meaning that there is more data available for each symbol variation, thus providing a solution to the classic "data starvation" problem as seen in many machine learning applications.

6.3.1 Ontology Generalization

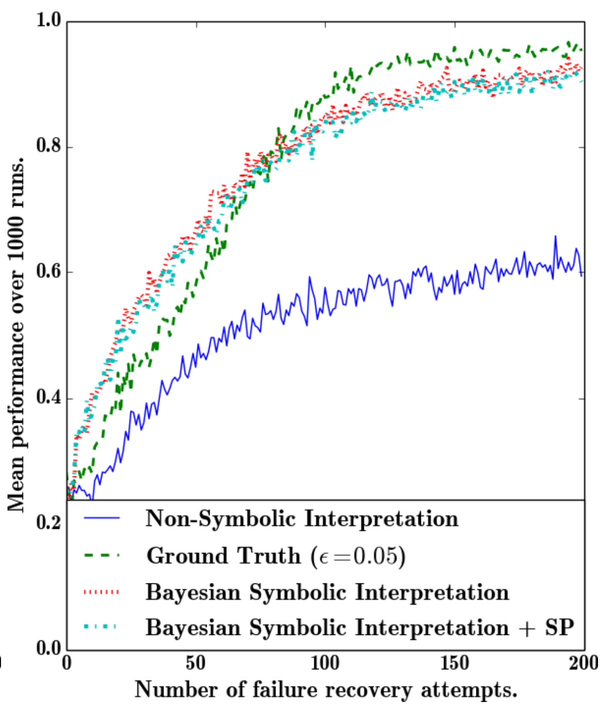
One way of forming new representations is to use general knowledge about concepts and their properties, for example, by using an ontology as seen in Figure 6.2 (see [74] for an overview of ontology evaluation techniques). The ontology allows to generalize over the different types of concepts and their properties. For example, by regarding everything as a "thing" and simplifying the location in either being "in the middle" or "on the side", the total number of symbol variations can be reduced significantly.

6.3.2 Formulas in First-Order Logic

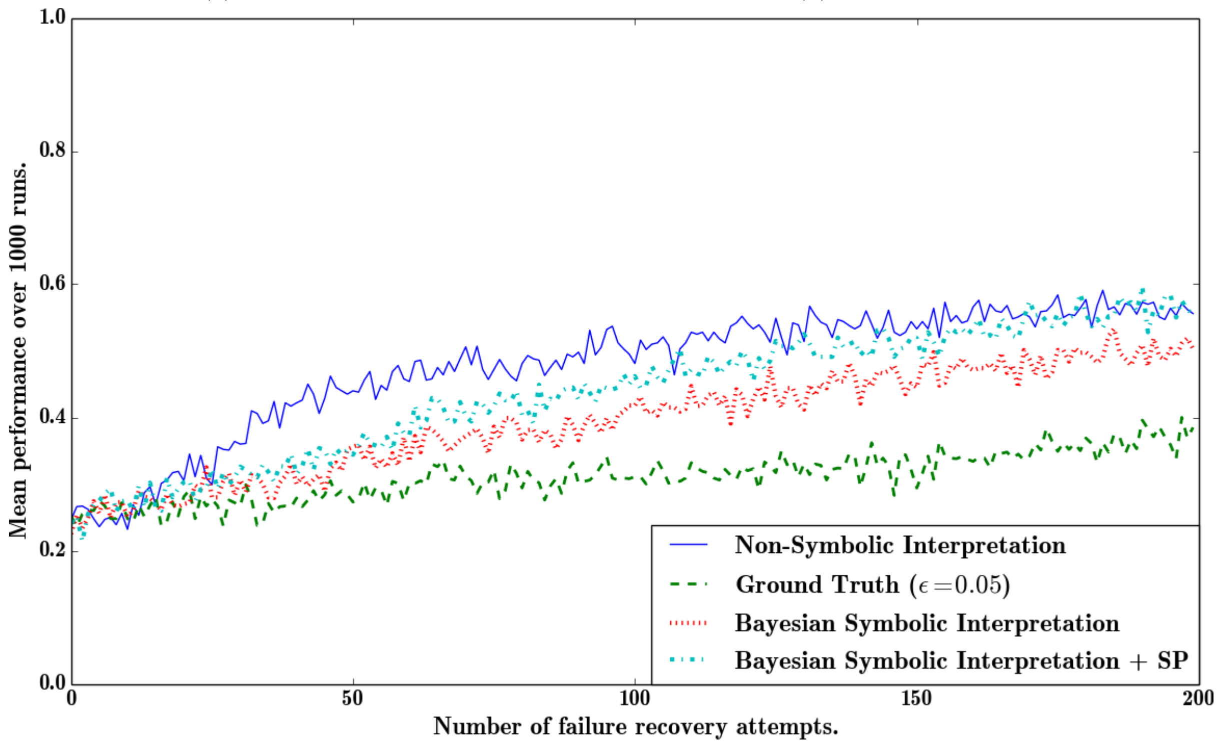
An alternative way of creating new symbolic representations is to form formulas in first-order logic and include, if the formula holds for the original representation, its unique label in the new representation. For example, a statement such as $\exists x\big(Nearby(x) \wedge Middle(x)\big)$ can be very useful in order to discard anything that is not in front of the entrance.



(a) Test Scenario 1.



(b) Test Scenario 2.



(c) Test Scenario 3.

Figure 6.1: Learning curves of the Bayesian approach in using the symbolic representation for all three test scenarios. Performance results for the Non-Symbolic Interpretation and ϵ-Greedy using Ground Truth information, are copied from Figure 5.3. Results for the Bayesian approach both without and with (Chapter 7) Symbol Perception (SP) are shown.

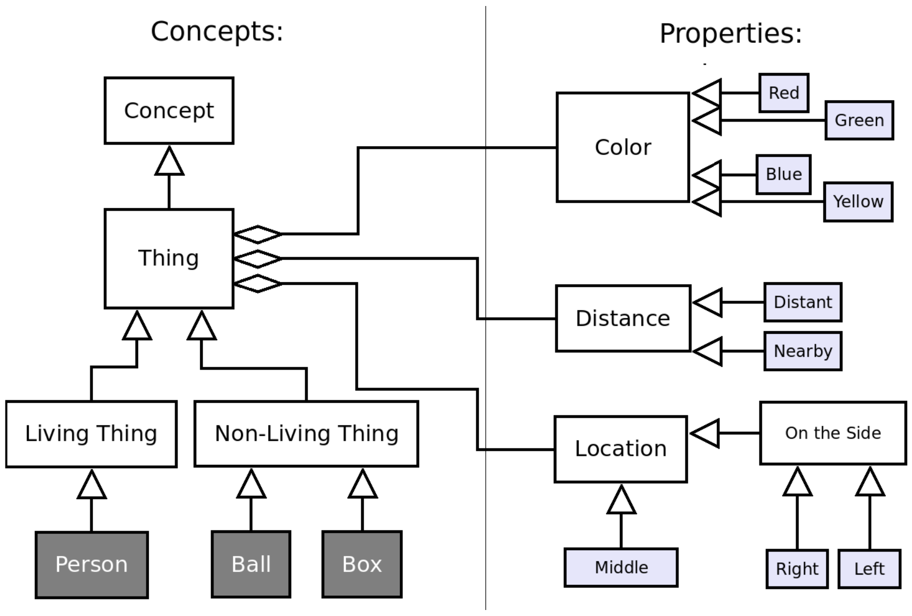


Figure 6.2: A diagram of the ontology as used in the experiments, defined using the Unified Modeling Language (UML) [75]. The symbolic representation of the environment in each failure state can be expressed in concepts (person, ball or box) and their properties (color, distance and location). The leaves of the ontology tree are the words (similar to the ones described in Figure 3.5 in Section 3.6) present in the original symbolic representation.

6.3.3 Representation Selection

New representations allow for less variations in symbols and thus increase the amount of data per symbol. This increases the reliability of the probabilities being calculated. However, this does not mean that the transformation results in a better representation. Sometimes the reason for the failure (and a given solution to be of any success) depends very much on a single detail, which is possibly lost during transformation. For this reason, the algorithm uses a set of possibly useful representations R and tries to maximize the probability of a given solution to be successful using R. To be specific, given the set of representations R and set of solutions S, the algorithm seeks to find a solution s which maximizes $p(s|r(O))$ for the current set of observations O. This essentially transforms the original equation of the Bayesian approach (Equation 6.9) into the following:

$$p(s) = \arg\max_{r \in R} \left(\arg\max_{s \in S} \Big(p(s|r(O)) \Big) \right) \tag{6.9}$$

Equation 6.8 in Section 6.2 ensures that any representation that is not representative of the current failure situation, automatically results in a lower probability value. The representation yielding the highest probability $p(s)$ for any given solution $s \in S$, is regarded as being the most meaningful and is used to select the best recovery solution.

6.3.4 Results

The performance of using dynamic (selection of) representations can be seen in Figure 6.3 for two different test scenarios. One of these is a custom test scenario, similar to test scenario 1 with 12 different failure states, but in which only the color of the concept matters to select the best recovery solution (solution “Push” works best for any concept that is either blue or green, while solution “Ask” works best for any concept that is either red or yellow). The other test scenario is test scenario 3 as described in Section 3.6.4.

In *both* test scenarios, the method uses a set of representations R, consisting of the following:

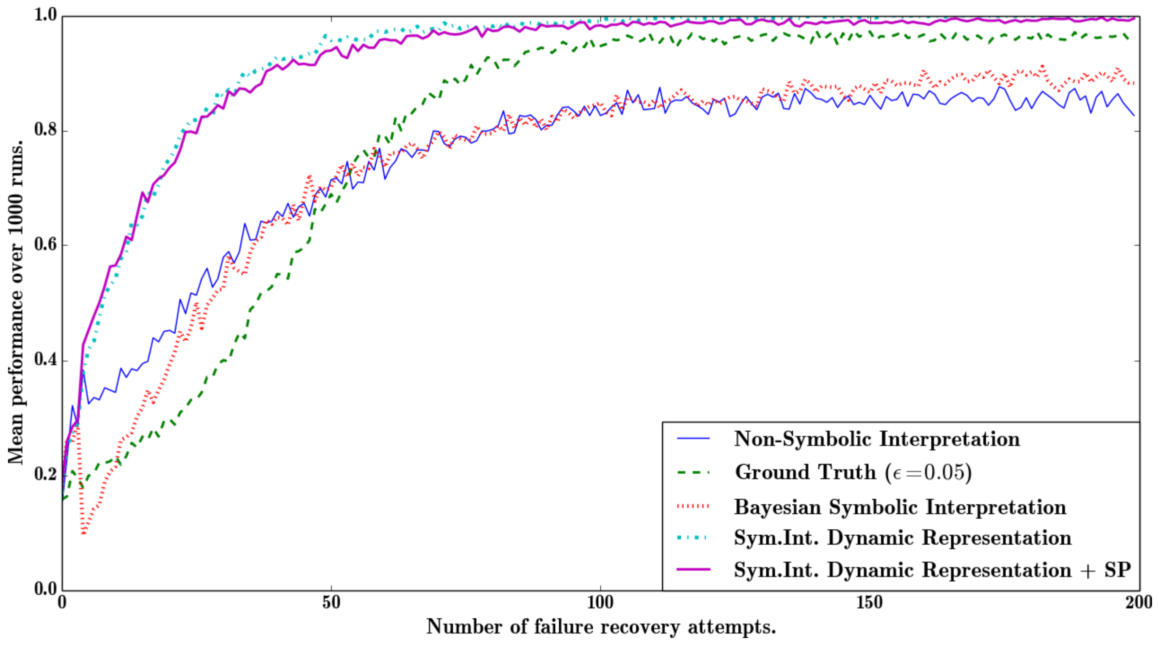
1. The original representation as also used in the original Bayesian approach (Section 6.2).
2. Using an ontology tree, a simplification based on a given concept type level and property level. Given the ontology tree (Figure 6.2), a total of two different concept type levels and two different property type levels are possible, resulting in four different representations.
3. A selection of symbols based on the logic statement: $\exists x\big(Nearby(x) \wedge Middle(x)\big)$
4. The logic statements: $\exists x\big(Blue(x) \vee Green(x)\big)$ or $\exists x\big(Red(x) \vee Yellow(x)\big)$, resulting in two different representations.

This results in a total of eight different representations. Even though each transformation results in a very different representation of the original, the algorithm is able to utilize the best representation for a given solution. This results in a significantly higher performance compared to the original Bayesian approach (see Figure 6.3).

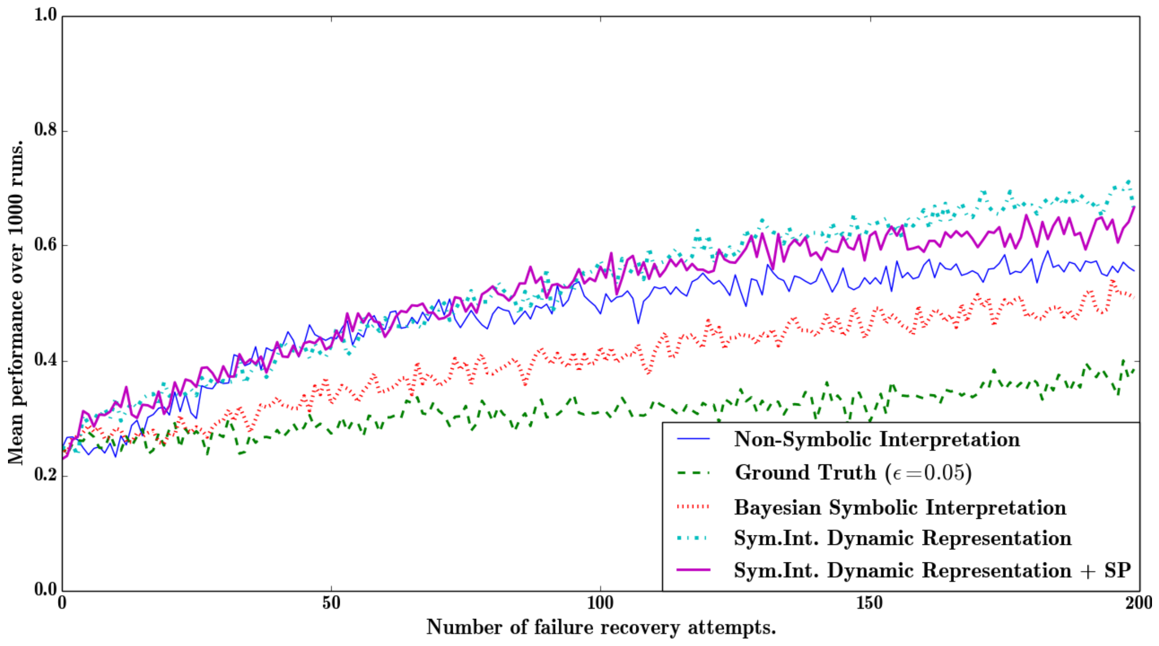
6.4 Discussion

The Bayesian approach of interpreting the symbolic representation is able to learn the best recovery solution in most cases without any additional knowledge of the environment. With the addition of dynamic representation selection, the performance can be increased significantly, thereby surpassing the performance of failure recovery when only low level sensory information is used using the non-symbolic failure recovery method. It even surpasses the performance of the Ground Truth method in which the exact failure state is given. This shows that perfect information about the failure situation is neither sufficient nor necessary, but that rather a good interpretation of this information is more important.

Using the different representations, not only allows for a higher final performance, but also allows for *faster* learning compared to the other methods. It is able to quickly remove any irrelevant data, thereby allowing it to accumulate more useful data in earlier attempts. However, it remains to be investigated as to how these representations can be extracted from the information in the ontology tree efficiently. A possible solution is to use Human Robot Interaction, in which the robot can ask for suggestions or verifications about possible hypotheses.



(a) Custom test scenario.



(b) Test scenario 3.

Figure 6.3: Learning curves of the Bayesian approach in using the *dynamic* symbolic representation for test scenario 3 (same as in Figure 6.1) and a custom test scenario (as described in Section 6.3). The dynamic representation approach allows the method to utilize a known ontology of concepts and any suggestive information provided by the user. In both the custom test scenario as well as in test scenario 3, the dynamic representation approach tests all possible representations simultaneously without knowing which one is the best. Results are shown for both without and with (Chapter 7) Symbol Perception (SP). See text for more details.

Chapter 7

Symbol Perception

The symbolic interpretation method for recovering of failures, shows promising results. However, its performance is only as good as the reliability of the original symbolic representation. Also here, the classic rule holds; garbage in $\rightarrow$ garbage out.

This section proposes a method of Symbol Perception (SP) in which the original symbolic representation, as used in Chapter 6, is deduced from the low level sensory information using a selection of machine learning methods. Except for the specific feature extraction methods being used, the method assumes as little as possible about the original data or classification to be made[1]. A complete overview of the symbol perception method is illustrated in Figure 7.1.

7.1 Learning Procedure

Using the symbolic representation of failure recovery allows the robot to utilize whatever symbol it has learned before. By the recognition of known concepts, the robot can already start interpreting the failure upon the first occurrence. While using the low-level sensory information as in the non-symbolic representation, the robot has to accumulate a lot more experience for each failure type over time before it can try resolving it.

However, this benefit is only of use if these symbols are known and learned beforehand. As such, the robot is allowed to first learn the concepts and their properties during a training procedure in which the robot is presented with the perception of different objects and persons in a special training location (without walls and a floor). In the training location, the robot is told what it sees and the properties of the concepts, for example, “There is a green distant ball on the left.”.

7.2 Specialized Perception Modules

One possibility is to use specialized perception modules to recognize the meaning of these symbols in the environment. One could, for instance, use existing recognition algorithms such as SIFT/SURF [76, 77] to recognize objects or HAAR/HOG [78, 79] to detect humans in the environment, and use the resulting combination of detections and classifications for the symbolic representation directly. However, this requires specialized training and careful selection for each of these algorithms, thereby introducing a stronger bias to the ideas of the designer. It also makes the recognition of generic properties of these symbols (i.e., adjectives to describe the location, size, color etc.) harder.

[1]Especially since the type of *referent*, as explained in Section 1.4, is assumed to be unknown.

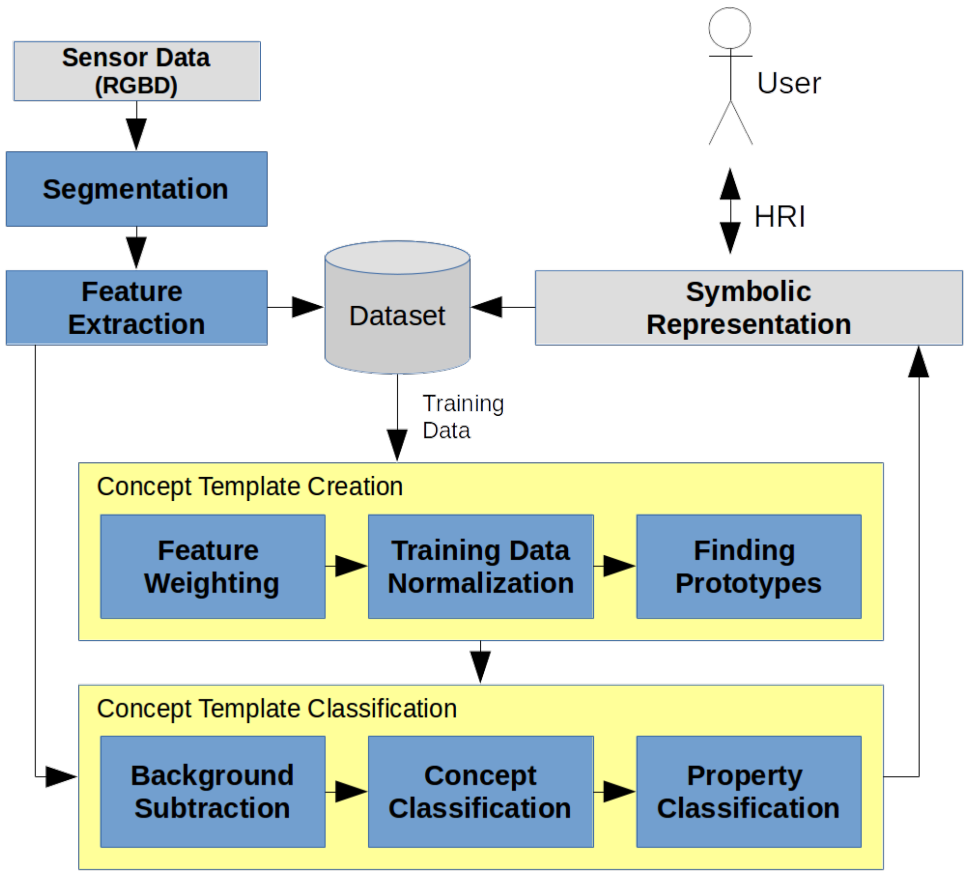


Figure 7.1: An overview of the symbol perception method. The symbolic representation as used in this thesis can either be explained to the robot by a human or recognized from low level sensory data using the symbol perception method explained in Chapter 7. During training, annotated surfaces are stored in the database after segmentation and feature extraction. Once a sufficient number of training samples have been stored for each concept (about 10 per concept (box, ball and person) and unique configuration of properties (location, color and distance)), a template is created for matching during classification. See Figure 7.2 for an Unified Modeling Language (UML) diagram of the Object-Relational Mapping (ORM) database of the dataset as used to store a single observation of a failure situation. See text for more details.

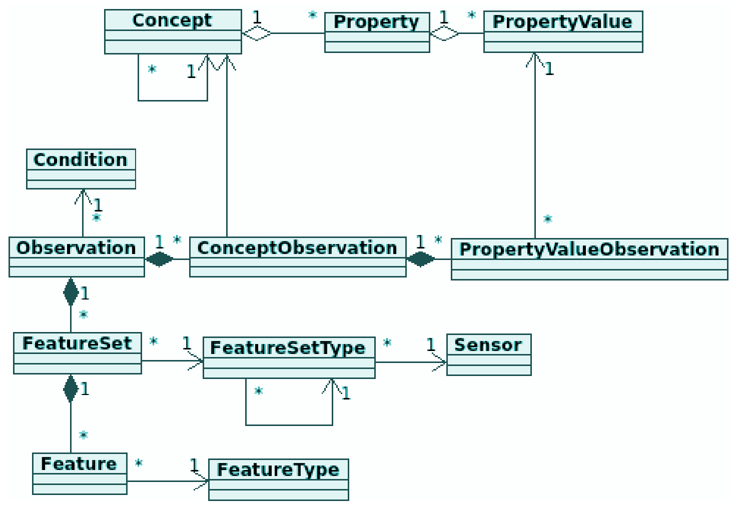


Figure 7.2: An Unified Modeling Language (UML) diagram of the Object-Relational Mapping (ORM) database of the dataset. A single observation of a possible failure situation can consist of one or more sets of features (forming multiple feature vectors) as a result of the pointcloud segmentation procedure.

7.3 Background Subtraction

The environment in which the concepts are being learned (an empty location), is distinctively different from the failures situations (in front of an entrance with walls, a floor and possibly other visible concepts) as used in the previous sections. During the detection and classification of the symbols, the key challenge is to identify the relevant information from the (unknown) background information (such as walls, floors and other (partially observable) concepts). This process of filtering relevant information from the background information can in essence be compared to the one-class classification problem [80] or outlier detection problem [81]. The Symbol Perception method is therefore designed, and the template classification procedure as discussed in Section 7.6, with this challenge in mind.

7.4 Pre-Processing

For the purpose of this research, only the top color and depth (RGBD) camera is used to extract low level information about the environment. However, unlike the feature vector extracted in Chapter 5, feature extraction for symbol grounding occurs in a significantly different manner. Unlike the previous method, in which a single feature vector is used for the classification of a failure state, the method explained here extracts a set of feature vectors which represent the presence of none, one or multiple known concepts in the scene. One or more feature vectors may originate from the presence of a single concept. Some feature vectors are part of the background scene (such as the door, walls and the floor) or belong to an unknown class of concepts.

The following sections explain the process of extracting this set of feature vectors from a single observation snapshot (either during training or at the failure situation itself). The actual classification of this set of feature vectors and its transformation to the symbolic representation, is discussed in Section 7.5 and Section 7.6. The Point Cloud Library (PCL) [82] is used for most operations required to segmentize the original pointcloud and extract features from a single segmented set of points.

7.4.1 Segmentation

Before feature extraction can occur, the pointcloud data extracted from the overlapping color and depth frames is segmented into different surfaces. The region growing algorithm[2] as provided by the Point Cloud Library (PCL) [82] is being used to segment surfaces from the original point cloud. The resulting surfaces (the different colors as shown in Figure 7.3) are segmented based on a specific smoothness and curvature threshold.

7.4.2 Feature Extraction

From each surface pointcloud (as illustrated in different colors in Figure 7.3), the following features are extracted, forming a single 28-dimensional feature vector:

1. A least-squares plane fit, resulting in three estimated plane parameters; x, y and z, together with the surface curvature.

2. A similarity measure compared to a known model such as a plane, cylinder, sphere and a line. Using the MLESAC (Maximum Likelihood Estimator SAmple Consensus) algorithm [83], which is in turn based on the RANSAC (RANdom SAmple Consensus) algorithm [84]. The similarity is calculated by dividing the inliers which correspond best to the model by the total number of points for a given model. Additional models could be added using expert knowledge about the type of concepts to be recognized. However, only primitive models are used in order to reduce the phenomenon of blindness [10].

3. The color image is merged with the pointcloud, which results in each point having a red, blue or green component in addition to the x, y, z coordinates. This is used to calculate a normalized 4-binned color histogram for each red, blue or green channel, resulting in a total of 12 features.

4. Some metric information, such as the centroid of the surface, its surface and cubic size and the width, height and depth dimensions. Principal Component Analysis (PCA) [85] is used to transform the original pointcloud to the origin in which the principal components represent the different dimensions of the surface pointcloud.

[2]`http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php`

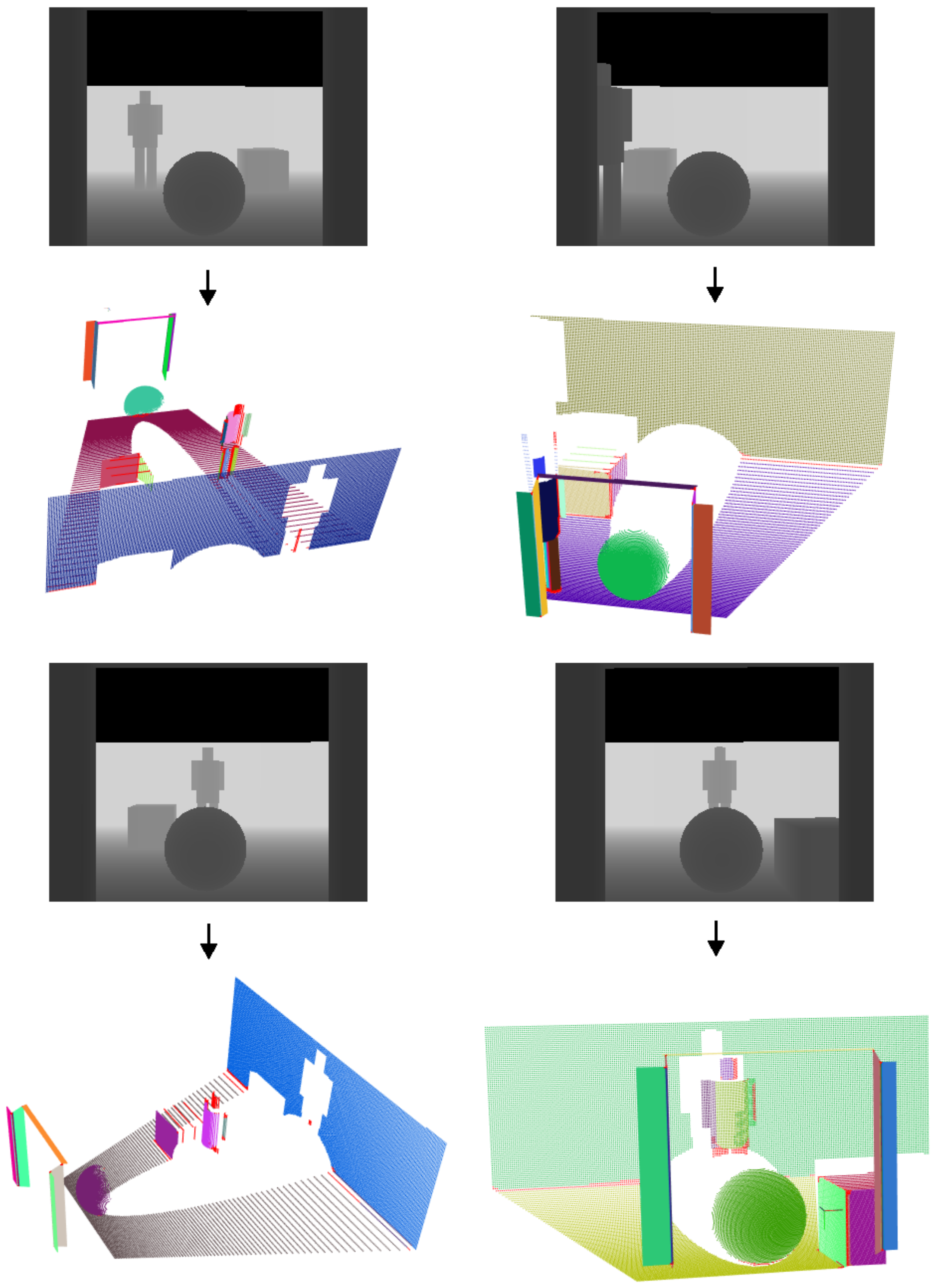


Figure 7.3: Four different example pointclouds as seen by the robot (while looking through the door), shown in different orientations. Using the region growing algorithm as provided in the Point Cloud Library (PCL), each pointcloud is segmented into different surfaces (shown in different colors). Regions shown in red are discarded and not part of the resulting set of surfaces. See text for more details.

7.5 Template Creation

Once enough concept training data has been collected (i.e., there is a sufficient number of labelled example surfaces for each concept), a template can be created to recognize a specific concept during the classification of a new scene. This section discusses the template creation algorithm as used in this research; the actual classification procedure is explained in Section 7.6.

The algorithm explained here assumes the input to be a collection of observations, in which each observation contains multiple feature vectors labelled as one or more concepts. The feature vectors originate from the feature extraction process described in Section 7.4.2. However, one could use the algorithm discussed here and in Section 7.6 using any form of perception as long as the same assumption holds.

The main purpose of the algorithm explained here, is to create a set of prototype feature vectors which best represent the original training data. Using only a few prototype feature vectors, rather than the complete training set, has the advantage of being less computationally expensive while increasing the overall generalization of the solution. The complete step-wise procedure to create these prototypes is illustrated in Figure 7.1. The following sections discuss each step in more detail.

7.5.1 Feature Weighting

Each type of feature as discussed in Section 7.4.2 ought to be useful. However, some might be more relevant (such as the dimensions of the surfaces) than others (such as the color). Therefore, feature weighting or selection is preferred, see [86, 87] for an overview of feature weighting and selection methods seen in literature.

It is natural to assume that features whose values are unique among all concept classes are more relevant than features of which the values are more common to all concept classes. In essence, the smaller the overlap between the values of one probability density function $p(f, c, x)$ of class $c \in C$ and all other classes $C \setminus \{c\}$, the higher the relevance (or weight w) for a given feature type f.

The weight $w(f, c)$ for any feature f and class c is calculated using the following:

$$w(f,c) = 1 - \int\limits_{min(f)}^{max(f)} \left(p(f,c,x) \frac{\sum\limits_{o \in C \setminus \{c\}} p(f,o,x)}{N-1} \right) dx \tag{7.1}$$

One simple and effective way to calculate the probability density function, is to calculate the mean value and its standard deviation for a given feature type value. However, this assumes the underlying distribution to be unimodal and symmetric. This is, however, often not the case since each concept has been learned in many different orientations and conditions. The overlap is therefore instead calculated using Kernel Density Estimation (KDE) also known as the *Parzen-Rosenblatt* window method [88, 89], using a Gaussian kernel. The SciKit Learn library [90] is used for the implementation of the Kernel Density Estimation.

7.5.2 Training Data Normalization

After the calculation of the weights for each class, one could modify the distance function required for the creation of prototypes and the classification of concepts. However, this would imply modifying all methods which normally assume a standard euclidean distance. The complete training set is therefore normalized prior to finding the prototypes, while any new feature vectors are normalized prior to classification (Section 7.6) using the feature weights calculated during training.

7.5.3 Finding Prototypes

After normalization, a k-means clustering algorithm [69] as provided by [90] is used to find the best prototypes representing a given class. The algorithm is allowed to run for a maximum of 300 iterations with a minimum tolerance of 10^{-4}. The number of prototypes per class depends on the mean number of feature vectors in all observations during the training procedure. For each prototype, the standard deviation of the distances of all matching feature vectors to the prototype are calculated.

7.6 Concept Template Classification

After training and template creation (Section 7.5), the template can be used to recognize the presence of one or more concepts in a single observation. Each feature vector present in the observation, is matched with the closest prototype. The distance from a given feature vector to the prototype is calculated as the total amount of standard deviations (as calculated from all matching training feature vectors). This is done in order to account for a difference in variance among different classes. Feature vectors whose distance is too large from any prototype, are discarded and assumed to be part of the background or an unknown class of concepts.

For each concept class, the number of matching feature vectors is used to determine whether or not the given concept is present in the observation. In order for a given concept class to be added to the symbolic representation, the number of matching feature vectors must be equal to or higher than the mean number of expected feature vectors minus twice its expected standard deviation.

7.6.1 Property Classification

After classification of the concept, the ontology tree (as illustrated in Figure 6.2) is used to identify the properties to be classified (color, location and distance). A k-Nearest Neighbor algorithm [67] (with $k = 3$) is used to classify each type of property in which only the training samples corresponding to the same concept class is being used.

7.7 Results

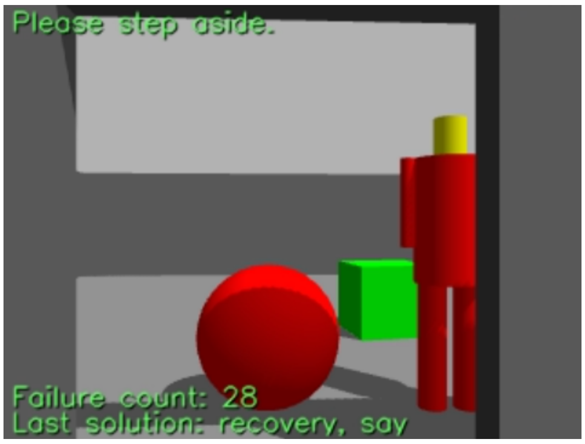
During the training procedure the robot is provided with 10 different training samples for each concept (box, ball, person) in each variation of color (red, green, yellow and blue), location (left, middle and right) and distance (distant and nearby), resulting in a total of 720 training samples. In each training sample, some random uniform noise is applied to the orientation and location of the concepts. During training, all concepts were placed in an empty environment without any background information such as a floor and walls.

Testing the classification on the complete dataset as specified in Section 3.8 (a total of 2340 samples), results in a mean total performance of 0.71. This means that in 71% of all classifications, the classified symbolic representation *exactly* matches the ground truth representation as extracted during dataset generation. In all other cases, the Symbol Perception module is either providing incorrect, insufficient or too much information in the symbolic representation. Figure 7.4 and Figure 7.5 provide examples of classification results for different failure situations.

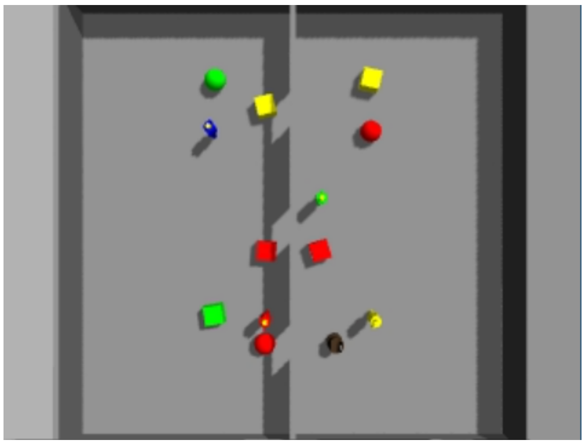
The performance of the symbol perception module (using the same training set as mentioned above) is tested in combination with the symbolic recoverer (see Chapter 6). The recovery performance results for all test scenarios in combination with Symbol Perception (SP) are shown in the previous section in Figure 6.1 and Figure 6.3.

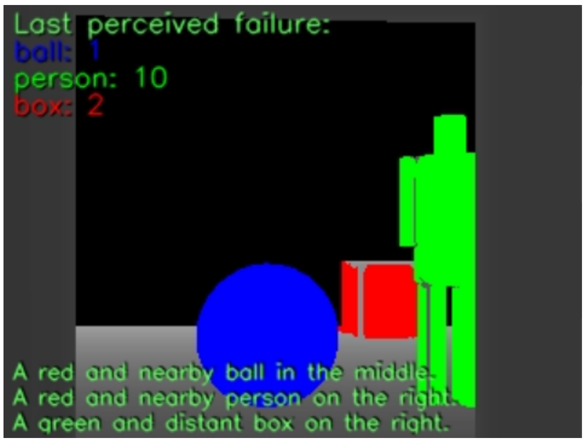
7.8 Discussion

With a total mean classification performance of 0.71, the Symbol Perception module is unable to classify the symbolic representation correctly at all times. However, as can be seen in Figure 6.1 and Figure 6.3, the actual recovery performance using the symbolic recoverer in combination with the Symbol Perception module, is almost equal to the recovery performance using the symbolic representation extracted from the ground truth information. This suggests that the symbolic recoverer is robust against misclassifications in the symbolic representation, while the Symbol Perception module does provide sufficient information despite its suboptimal performance.


Please step aside.
Failure count: 28
Last solution: recovery, say

(a) The color image as seen by the robot. The text shown at the very top indicates what the robot is saying. The text shown at the very bottom indicates the last failure recovery solution executed by the robot.


(b) A top-down view of the arena as seen in simulation. Some uniform noise is applied to the location and orientation of the objects and persons. The environment changes the moment the robot moves from one room to the other.


Last perceived failure:
ball: 1
person: 10
box: 2

A red and nearby ball in the middle.
A red and nearby person on the right.
A green and distant box on the right.

(c) The classification result of each surface annotated in the original gray-scale depth image. The number of detected surfaces for each type of concept is shown at the very top. At the very bottom, a textual form of the resulting symbolic representation is shown. See also figure 7.5.

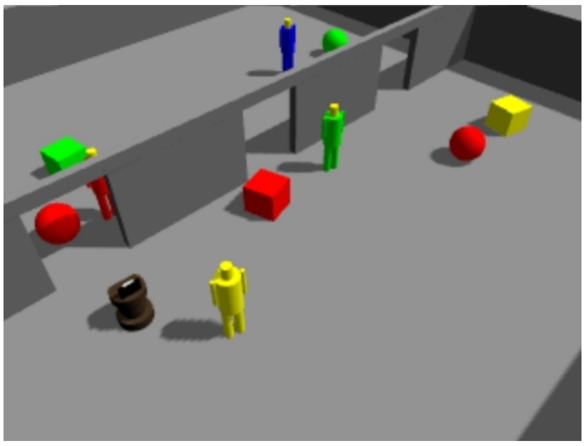

(d) A side view of the arena as seen in simulation.

Figure 7.4: An example snapshot of the demonstrated failure state interpretation and recovery as seen in simulation. Each screenshot is taken at the same time at the occurrence of a single failure situation while the robot tries to move from one room to the other. See text for more details.

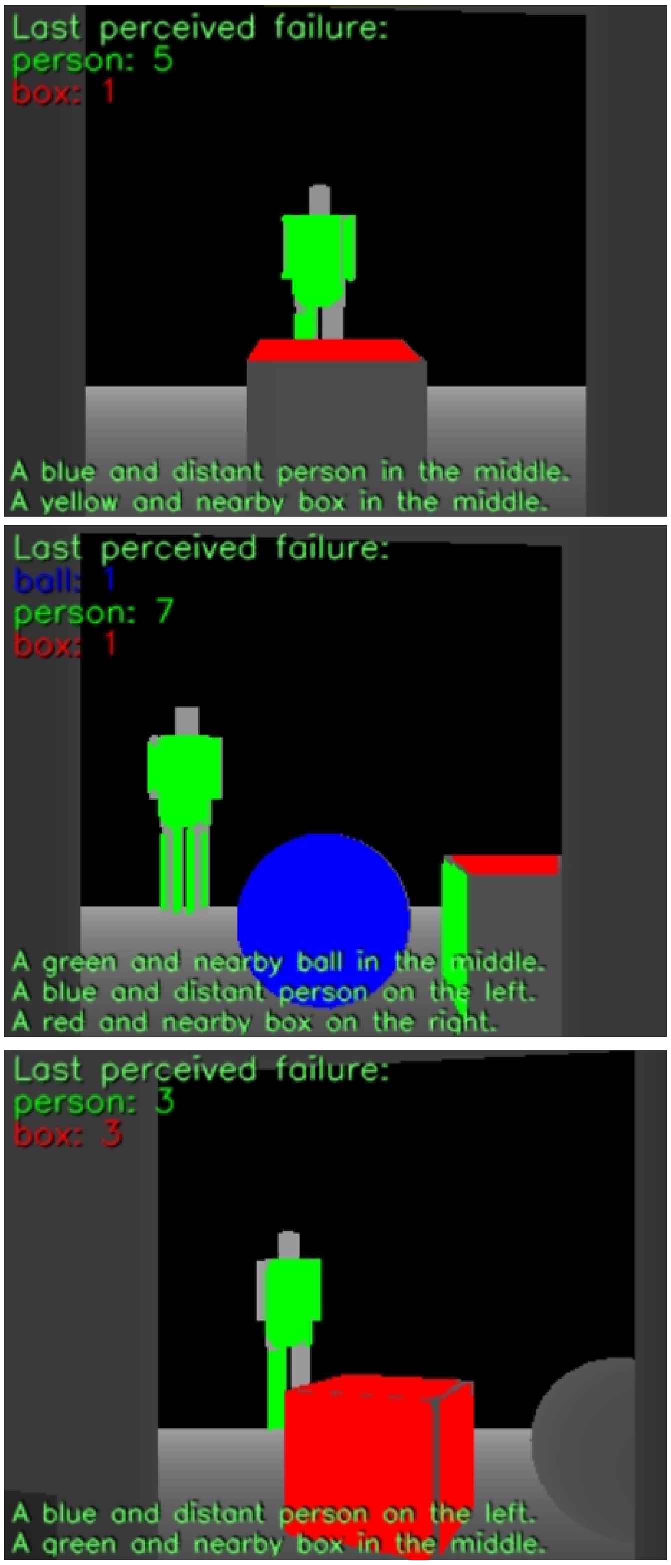

Last perceived failure:
person: 5
box: 1
A blue and distant person in the middle.
A yellow and nearby box in the middle.
Last perceived failure:
ball: 1
person: 7
box: 1
A green and nearby ball in the middle.
A blue and distant person on the left.
A red and nearby box on the right.
Last perceived failure:
person: 3
box: 3
A blue and distant person on the left.
A green and nearby box in the middle.

Figure 7.5: Example classification results in three different failure situations in which the symbol perception method is unable to classify all surfaces correctly. However, the resulting symbolic representation, shown in textual form at the very bottom of each image, is still valid despite this misclassification.

Chapter 8

Conclusion

The results discussed in this thesis have shown that, for a domestic service robot, efficient failure recovery is possible using both a non-symbolic and symbolic representation of the environment. Efficient failure recovery is possibly even if the robot has no prior knowledge about the number or different types of failures that may occur. This eliminates the strict need for the designer to account for all possible failures (on a behavioral level) prior to the actual application of a domestic service robot.

In most cases, the performance of the method of symbolic interpretation of failures surpasses the performance of the non-symbolic method. In cases where there is a lot of variation possible in the failure state, the symbolic representation suffers from the classical data starvation problem. This problem can be solved by transforming the original symbol observations to a new more abstract representation. The method is efficient at selecting the best representation to use for a given failure state, thereby in essence describing and explaining the failure in its most abstract explanatory form.

The Symbol Perception module has proven to be adequate to classify the symbolic representation, thereby allowing the symbolic failure recoverer to surpass the recovery performance of the non-symbolic recoverer, even though the same sensory information is used. This while the symbolic recoverer also allows to express the failure situation in a more human comprehensible form.

With the results shown in this thesis, all four hypotheses, as described in Section 1.4, can be confirmed, thereby making a step forward in achieving our goal of designing domestic service robots capable of meeting the changing demands of their users while dealing with the complex dynamics of ever-changing domestic environments.

8.1 Limitations

The symbolic interpretation method using symbol perception is efficient at selecting the best representation to use for a given failure state as can be seen in the custom test scenario (see Figure 6.3). Here, it has a significantly higher final mean score (0.985) compared to both the recoverer using solely ground truth information (a mean score of 0.795, with a two-tailed p-value of 5.6×10^{-10} as calculated using T-test on the means of the two independent samples) and the non-symbolic recoverer (a mean score of 0.74 with a two-tailed p-value of 2.25×10^{-13}).

However, in test scenario 3, the symbolic interpretation method using symbol perception does have a significantly higher mean score (0.54) than the ground truth recoverer (0.315, with a two-tailed p-value of 4.3×10^{-6}), but not a significantly higher score than that of the non-symbolic recoverer (0.535, with a two-tailed p-value of 0.92). Furthermore, it remains to be investigated as to how these representations can be extracted more efficiently. This can be achieved either with a more in depth investigation of the ontology, or by interaction with the user.

Although the experiments conducted in this research have been designed to reflect a domestic environment as well as possible in a simulated environment, future research should focus on the verification of the methods explained in this thesis in a real world environment using real robots.

8.2 Future Perspective

For a service robot to operate successfully in a domestic environment, we should accept the idea that the constant anticipation, recognition and recovery of failures is the default state in which it operates. Not all potential failure situations can be accounted for during the design and development of a domestic service robot. This implies that a domestic service robot should demonstrate behavior and have a situational awareness of its environment which goes beyond its initial programming.

The field of Autonomous Mental Development [91, 12] and Developmental Robotics [92, 93, 94] might offer solutions for the development of such systems in which cognition is scaffolded through the robot's interaction with the environment. However, this raises the question as to how to make a robot accountable for its actions, in which it is able to share knowledge with humans, explain its reasoning and provides arguments for its decisions. In contrast, methods related to (classical) Cognitivism offer easier solutions for backward reasoning and sharing knowledge while increasing the explanatory capabilities of the system.

Clearly, a combination of both is required if we desire to develop robots, which not only learn and perform complex tasks in a domestic environment, but also operate *safely* and responsibly in which their actions are accountable (in court).

Bibliography

[1] J. Forlizzi and C. DiSalvo, “Service robots in the domestic environment: a study of the roomba vacuum in the home,” in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 258–265.

[2] L. Iocchi, J. Ruiz-del Solar, and T. van der Zant, “Domestic service robots in the real world,” *Journal of Intelligent & Robotic Systems*, vol. 66, no. 1, pp. 183–186, 2012.

[3] S. Schneider, F. Hegger, A. Ahmad, I. Awaad, F. Amigoni, J. Berghofer, R. Bischoff, A. Bonarini, R. Dwiputra, G. Fontana *et al.*, “The RoCKIn@ home challenge,” in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of.* VDE, 2014, pp. 1–7.

[4] M. J. Matarić, “Situated robotics,” *Encyclopedia of cognitive science*, 2002.

[5] B. Rechel, E. Grundy, J.-M. Robine, J. Cylus, J. P. Mackenbach, C. Knai, and M. McKee, “Ageing in the european union,” *The Lancet*, vol. 381, no. 9874, pp. 1312–1322, 2013.

[6] J. M. Ortman, V. A. Velkoff, H. Hogan *et al.*, “An aging nation: the older population in the United States,” *Washington, DC: US Census Bureau*, pp. 25–1140, 2014.

[7] J. Carlson and R. R. Murphy, “Reliability analysis of mobile robots,” in *Robotics and Automation, 2003. Proceedings. ICRA’03. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 274–281.

[8] Y.-H. Wu, V. Cristancho-Lacroix, C. Fassert, V. Faucounau, J. de Rotrou, and A.-S. Rigaud, “The attitudes and perceptions of older adults with mild cognitive impairment toward an assistive robot,” *Journal of Applied Gerontology*, vol. 35, no. 1, pp. 3–17, 2016.

[9] ISO, “Robots and robotic devices-safety requirements for personal care robot,” International Organization for Standardization, Geneva, Switzerland, ISO 13482, 2014.

[10] T. Winograd and F. Flores, *Understanding computers and cognition: A new foundation for design*. Intellect Books, 1986.

[11] Y. Wang, X. Wu, and J. Weng, “Skull-closed autonomous development,” in *Neural Information Processing*. Springer, 2011, pp. 209–216.

[12] D. Vernon, G. Metta, and G. Sandini, “A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents,” *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 2, pp. 151–180, 2007.

[13] T. Fong, I. Nourbakhsh, and K. Dautenhahn, “A survey of socially interactive robots,” *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 143–166, 2003.

[14] M. A. Goodrich and A. C. Schultz, “Human-robot interaction: a survey,” *Foundations and Trends in Human-Computer Interaction*, vol. 1, no. 3, pp. 203–275, 2007.

[15] S. Harnad, “The symbol grounding problem,” *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 335–346, 1990.

[16] M. Taddeo and L. Floridi, “Solving the symbol grounding problem: a critical review of fifteen years of research,” *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 17, no. 4, pp. 419–445, 2005.

[17] P. Vogt, “The physical symbol grounding problem,” *Cognitive Systems Research*, vol. 3, no. 3, pp. 429–457, 2002.

[18] C. K. Ogden, I. A. Richards, S. Ranulf, and E. Cassirer, “The meaning of meaning. a study of the influence of language upon thought and of the science of symbolism,” 1923.

[19] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker, “Robotic fault detection and fault tolerance: A survey,” *Reliability Engineering & System Safety*, vol. 46, no. 2, pp. 139–158, 1994.

[20] D. Crestani and K. Godary-Dejean, “Fault tolerance in control architectures for mobile robots: Fantasy or reality?” in *CAR’2012: 7th National Conference on Control Architectures of Robots*, 2012.

[21] E. Papadopoulos and S. Dubowsky, “Failure recovery control for space robotic systems,” in *American Control Conference, 1991.* IEEE, 1991, pp. 1485–1490.

[22] C. Angle and R. Brooks, “Small planetary rovers,” in *Intelligent Robots and Systems’ 90.’Towards a New Frontier of Applications’, Proceedings. IROS’90. IEEE International Workshop on.* IEEE, 1990, pp. 383–388.

[23] C. Ferrell, “Robust agent control of an autonomous robot with many sensors and actuators,” 1993.

[24] ——, “Failure recognition and fault tolerance of an autonomous robot,” *Adaptive behavior*, vol. 2, no. 4, pp. 375–398, 1994.

[25] P. Runeson, “A survey of unit testing practices,” *Software, IEEE*, vol. 23, no. 4, pp. 22–29, 2006.

[26] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, “Fault injection techniques and tools,” *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[27] T. Wilfredo, “Software fault tolerance: A tutorial,” 2000.

[28] L. Chen and A. Avizienis, “N-version programming: A fault-tolerance approach to reliability of software operation,” in *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*, 1978, pp. 3–9.

[29] T. G. Dietterich, “Ensemble methods in machine learning,” in *Multiple classifier systems*. Springer, 2000, pp. 1–15.

[30] N. Akhtar, A. Kuestenmacher, P. G. Ploger, and G. Lakemeyer, “Simulation-based approach for avoiding external faults,” in *Advanced Robotics (ICAR), 2013 16th International Conference on.* IEEE, 2013, pp. 1–8.

[31] A. Küstenmacher, N. Akhtar, P. G. Plöger, and G. Lakemeyer, "Unexpected situations in service robot environment: Classification and reasoning using naive physics," in *RoboCup 2013: Robot World Cup XVII*. Springer, 2014, pp. 219–230.

[32] P. Traverso, L. Spalazzi, and F. Giunchiglia, *Reasoning about acting, sensing and failure handling: a logic for agents embedded in the real world*. Springer, 1996.

[33] D. W. Payton, D. Keirsey, D. M. Kimble, J. Krozel, and J. K. Rosenblatt, "Do whatever works: A robust approach to fault-tolerant autonomous control," *Applied Intelligence*, vol. 2, no. 3, pp. 225–250, 1992.

[34] L. E. Parker, "Alliance: An architecture for fault tolerant multirobot cooperation," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 2, pp. 220–240, 1998.

[35] E. D. Sacerdoti, "A structure for plans and behavior," DTIC Document, Tech. Rep., 1975.

[36] Y.-S. Lee and S.-B. Cho, "A hybrid system of hierarchical planning of behaviour selection networks for mobile robot control," *International Journal of Advanced Robotic Systems*, vol. 11, 2014.

[37] K.-H. Chang, H. Han, and W. B. Day, "A comparison of failure-handling approaches for planning systems—replanning vs. recovery," *Applied Intelligence*, vol. 3, no. 4, pp. 275–300, 1993.

[38] C. H. Bennett, "Notes on landauer's principle, reversible computation, and maxwell's demon," *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, vol. 34, no. 3, pp. 501–510, 2003.

[39] T. Yokoyama, "Reversible computation and reversible programming languages," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 6, pp. 71–81, 2010.

[40] H. B. Axelsen and R. Glück, "What do reversible programs compute?" in *FOSSACS*. Springer, 2011, pp. 42–56.

[41] U. P. Schultz, J. S. Laursen, L.-P. Ellekilde, and H. B. Axelsen, "Towards a domain-specific language for reversible assembly sequences," in *Reversible Computation*. Springer, 2015, pp. 111–126.

[42] R. A. Knepper, S. Tellex, A. Li, N. Roy, and D. Rus, "Recovering from failure by asking for help," *Autonomous Robots*, vol. 39, no. 3, pp. 347–362, 2015.

[43] S. A. Tellex, T. F. Kollar, S. R. Dickerson, M. R. Walter, A. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," 2011.

[44] M. Salem, G. Lakatos, F. Amirabdollahian, and K. Dautenhahn, "Would you trust a (faulty) robot?: Effects of error, task type and personality on human-robot cooperation and trust," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2015, pp. 141–148.

[45] P. Gieselmann, "Comparing error-handling strategies in human-human and human-robot dialogues," in *Proc. 8th Conf. Nat. Language Process.(KONVENS). Konstanz, Germany*, 2006, pp. 24–31.

[46] D. Holz, L. Iocchi, and T. van der Zant, "Benchmarking intelligent service robots through scientific competitions: The robocup@ home approach." in *AAAI Spring Symposium: Designing Intelligent Robots*, 2013.

[47] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the first international conference on Autonomous agents*. ACM, 1997, pp. 340–347.

[48] B. Hickendorff, "Enhanced human-robot interaction by reason-based behavior adaptation," *Master Thesis*, 2011.

[49] V. Richthammer, F. Schimbinschi, M. Schutten, A. Shantia, R. Snijders, E. van der Wal, and T. van der Zant, "Borg-the robocup@ home team of the university of groningen team description paper."

[50] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[51] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[52] S. Thrun and J. J. Leonard, "Simultaneous localization and mapping," in *Springer handbook of robotics*. Springer, 2008, pp. 871–889.

[53] C. Stachniss and G. Grisetti, "Gmapping project at openslam. org," 2007.

[54] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," *Advances in Neural Information Processing Systems (NIPS)*, pp. 26–32, 2001.

[55] P. Lamere, P. Kwok, E. Gouvea, B. Raj, R. Singh, W. Walker, M. Warmuth, and P. Wolf, "The cmu sphinx-4 speech recognition system," in *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*, vol. 1. Citeseer, 2003, pp. 2–5.

[56] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using hidden markov models," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 11, pp. 1641–1648, 1989.

[57] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[58] R. S. Sutton, "Temporal credit assignment in reinforcement learning," 1984.

[59] S. Mannor, "k-armed bandit," in *Encyclopedia of Machine Learning*. Springer, 2010, pp. 561–563.

[60] D. A. Berry and B. Fristedt, *Bandit problems: sequential allocation of experiments (Monographs on statistics and applied probability)*. Springer, 1985.

[61] A. L. Strehl and M. L. Littman, "An empirical evaluation of interval estimation for markov decision processes," in *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. IEEE, 2004, pp. 128–135.

[62] M. A. Wiering, "Explorations in efficient reinforcement learning," *PhD Thesis, University of Amsterdam*, 1999.

[63] T. Van der Zant, M. Wiering, and J. Van Eijck, “On-line robot learning using the interval estimation algorithm,” in *Proceedings of the 7th European Workshop on Reinforcement Learning. CiteSeer*, 2005, pp. 11–12.

[64] L. P. Kaelbling, *Learning in embedded systems.* MIT press, 1993.

[65] M. M. Deza and E. Deza, *Encyclopedia of distances.* Springer, 2009.

[66] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification.* John Wiley & Sons, 2012.

[67] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[68] R. Kohavi *et al.*, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings IJCAI*, vol. 14, no. 2, 1995, pp. 1137–1145.

[69] J. MacQueen *et al.*, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[70] S. C. Johnson, “Hierarchical clustering schemes,” *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[71] I. Rish, “An empirical study of the naive Bayes classifier,” in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22. IBM New York, 2001, pp. 41–46.

[72] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A Bayesian approach to filtering junk e-mail,” in *Learning for Text Categorization: Papers from the 1998 workshop*, vol. 62, 1998, pp. 98–105.

[73] P. Graham, “Better bayesian filtering,” in *Proceedings of the 2003 spam conference*, vol. 11, 2003, pp. 15–17.

[74] J. Brank, M. Grobelnik, and D. Mladenic, “A survey of ontology evaluation techniques,” in *Proceedings of the conference on data mining and data warehouses (SiKDD 2005)*, 2005, pp. 166–170.

[75] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The.* Pearson Higher Education, 2004.

[76] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[77] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer vision–ECCV 2006.* Springer, 2006, pp. 404–417.

[78] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[79] R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 1. IEEE, 2002, pp. I–900.

[80] S. S. Khan and M. G. Madden, “A survey of recent trends in one class classification,” in *Artificial Intelligence and Cognitive Science*. Springer, 2009, pp. 188–197.

[81] V. J. Hodge and J. Austin, “A survey of outlier detection methodologies,” *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[82] R. B. Rusu and S. Cousins, “3d is here: Point cloud library (pcl),” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[83] P. H. Torr and A. Zisserman, “Mlesac: A new robust estimator with application to estimating image geometry,” *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 138–156, 2000.

[84] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[85] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[86] D. Wettschereck, D. W. Aha, and T. Mohri, “A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms,” *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 273–314, 1997.

[87] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[88] M. Rosenblatt *et al.*, “Remarks on some nonparametric estimates of a density function,” *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.

[89] E. Parzen, “On estimation of a probability density function and mode,” *The annals of mathematical statistics*, pp. 1065–1076, 1962.

[90] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in Python,” *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[91] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, “Autonomous mental development by robots and animals,” *Science*, vol. 291, no. 5504, pp. 599–600, 2001.

[92] M. Asada, K. F. MacDorman, H. Ishiguro, and Y. Kuniyoshi, “Cognitive developmental robotics as a new paradigm for the design of humanoid robots,” *Robotics and Autonomous Systems*, vol. 37, no. 2, pp. 185–193, 2001.

[93] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida, “Cognitive developmental robotics: a survey,” *Autonomous Mental Development, IEEE Transactions on*, vol. 1, no. 1, pp. 12–34, 2009.

[94] J. C. Bongard, “Evolutionary robotics,” *Communications of the ACM*, vol. 56, no. 8, pp. 74–83, 2013.

Appendix A

Sliding Window Score Distribution Cut-off Algorithm

In case of using the score distribution as a selection criterium (see Chapter 5), the algorithm uses the following steps:

1. Select an initial window of size m most similar experiences (ordered in terms of dissimilarity).
2. Using this window, create a vector $\mathbf{w_i}$ in which each element represents the mean reward for each type of recovery action.
3. Using the total training set, create a vector $\mathbf{w_t}$ in which each element represents the mean reward for each type of recovery action.
4. Calculate the euclidean distance d_{it} between $\mathbf{w_i}$ and $\mathbf{w_t}$.
5. $i = 0$
6. Until $d_n > 1.0$ (and at most till the end of the training set):
   (a) Move the window with one position.
   (b) Using the window, create a vector $\mathbf{w_c}$ in which each element represents the mean reward for each type of recovery action.
   (c) Calculate the euclidean distance d_{ic} between $\mathbf{w_i}$ and $\mathbf{w_c}$.
   (d) Normalize using: $d_n = \frac{d_{ic}}{d_{it}}$
   (e) $i = i + 1$
7. $k = m + i$
8. Use the k most similar experiences of the training set to determine the best recovery action as discussed in Chapter 5.

Appendix B

ROS Packages

The following sections provides a list of ROS packages as used in the project[1].

B.1 RITA Platform

The following list of ROS packages are being used in order to make use of the RITA robotic platform.

en_behavior:
The (modified) core of the behavior architecture.

en_platform:
Required to control (the movements of) the RITA robot outside simulation.

en_nav:
Required for navigating the RITA in a mapped environment (which in turn uses the move_base ROS package).

en_description:
The URDF model of the RITA robot to be used in the Gazebo simulator.

en_gazebo:
Additional utilities for the Gazebo simulator (such as world and map files) for using the RITA robot in simulation.

[1]See the project webpage for more (up-to-date) technical (API) documentation: `http://cno.nu/msc`.

B.2 Project Related

The following list of ROS packages are explicitely made for the purpose of the project described in this thesis.

rs_behavior_recovery:
The specific behavior recoverers which extends the basic mechanism in the original architecture.

rs_symbol_grounding:
Anything related to the autonomous perception of symbols/concepts in the environment.

rs_exp:
The main package related used to setup the experiments. Includes utility libraries, scripts and Gazebo models.

rs_msgs:
Custom ROS messages related to the project.

rs_util:
Utility libraries.

B.3 ROS Graph

The diagram below illustrates the connection between all ROS nodes as used during experimentation. The behavior architecture and recovery mechanisms are run inside the "brain" node (part of the en_behavior package).

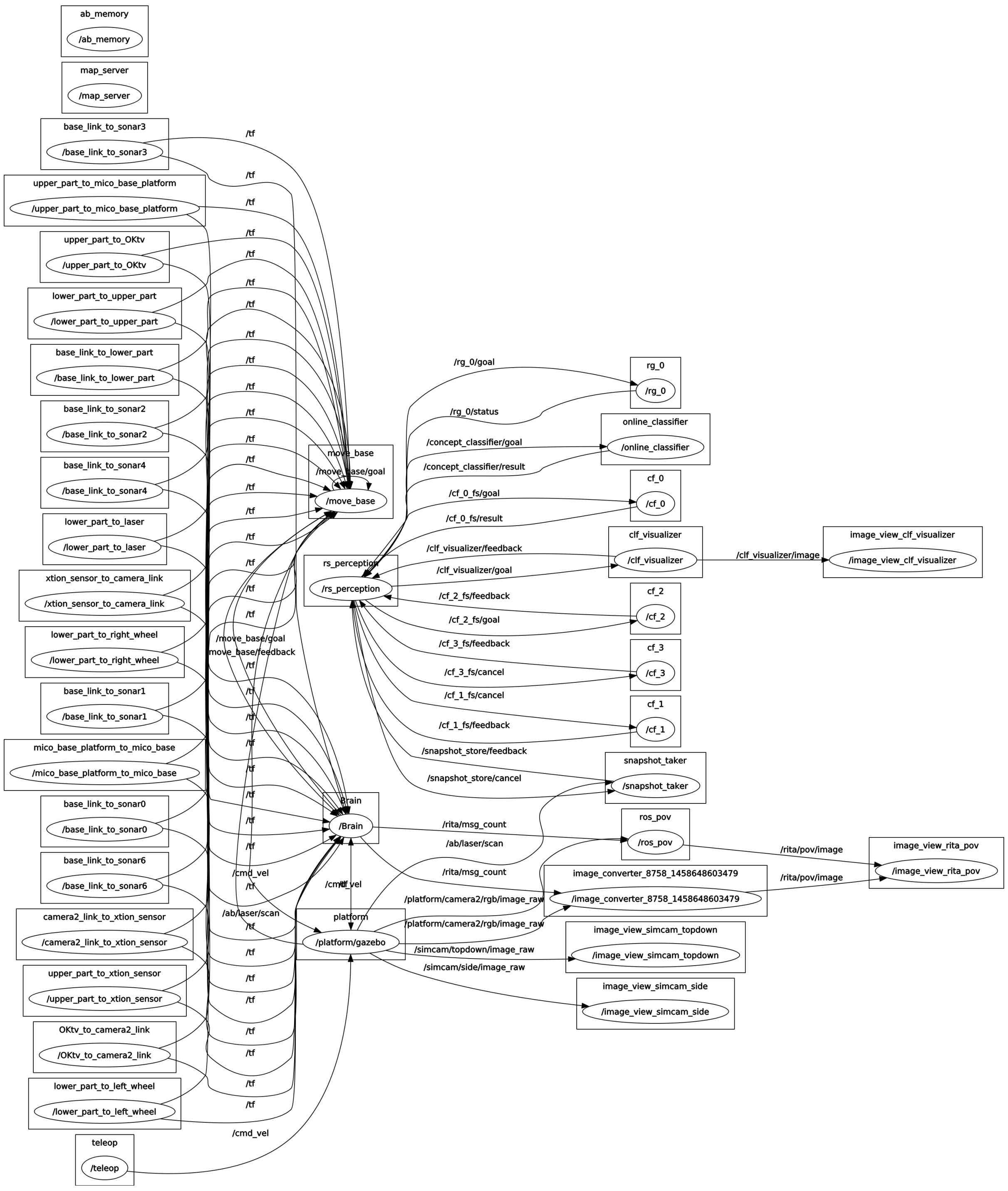


Figure B.1: The ROS graph during experimentation as generated by the rqt_graph ROS package.