Automated Learning of Communication Models for Robot Control Software

Alexander Kleiner, Gerald Steinbauer and Franz Wotawa

Post Print

Original Publication:

Alexander Kleiner, Gerald Steinbauer and Franz Wotawa, Automated Learning of Communication Models for Robot Control Software, 2008, MBS 2008 - Workshop on Model-Based Systems, 18th European Conference on Artificial Intelligence (ECAI).

Postprint available at: Linköping University Electronic Press
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-72538

Automated Learning of Communication Models for Robot Control Software

Alexander Kleiner [1] and **Gerald Steinbauer** [2] and **Franz Wotawa** [2]

Abstract.
Control software of autonomous mobile robots comprises a number of software modules which show very rich behaviors and interact in a very complex manner. These facts among others have a strong influence on the robustness of robot control software in the field. In this paper we present an approach which is able to automatically derive a model of the structure and the behavior of the communication within a component-orientated control software. Such a model can be used for on-line model-based diagnosis in order to increase the robustness of the software by allowing the robot to autonomously cope with faults occurred during runtime. Due to the fact that the model is learned form recorded data and the use of the popular publisher-subscriber paradigm the approach can be applied to a wide range of complex and even partially unknown systems.

1 Introduction

Control software of autonomous mobile robots comprises a number of software modules which show very rich behaviors and interact in a very complex manner. Because of this complexity and other reasons like bad design and implementation there is always the possibility that a fault occurs at runtime in the field. Such faults can have different characteristics like crashes of modules, deadlocks or wrong data leading to a hazardous decision of the robot. This situation can occur even if the software is carefully designed, implemented and tested. In order to have truly autonomous robots operating for a long time without or with limited possibility for human intervention, e.g., planetary rovers exploring Mars, such robots have to have the capability to detect, localize and to cope with such faults.

In [8, 7] the authors presented a model-based diagnosis framework for control software for autonomous mobile robots. The control software is based on the robot control framework *Miro* [10, 9] and has a client-server architecture where the software modules communicate by exchanging events. The idea is to use the different communication behaviors between the modules of the control software in order to monitor the status of the system and to detect and localize faults. The model comprises a graph specifying which modules communicate with each other. Moreover, the model has information about the type of a particular communication path, e.g, whether the communication occurs on a regular basis or sporadically. Finally, the model includes information about which inputs and outputs of the software modules have a functional relation, e.g, which output is triggered by which input. The model is specified by a set of logic clauses and uses a component-based modeling schema [1]. Please refer to [8, 7] for more details.

The diagnosis process itself uses the well known consistency-based diagnosis techniques of Reiter [5]. The models of the control software and the communication were created by hand by analyzing the structure of the software and its communication behavior during runtime. Because of the complexity of such control software or the possible lack of information about the system it is not feasible to do this by hand for large or partially unknown systems.

Therefore, it is desirable that such models can be created automatically either from a formal specification of the system or from observation of the system. In this paper we present an approach which allows the automatic extraction of all necessary information from the recorded communication between the software modules. The algorithm provides all information needed for model-based diagnosis. It provides a communication graph showing which modules communicate, the desired behavior of the particular communication paths and the relation between the inputs and outputs of the software modules.

These model learning approach was originally developed for and tested with the control software of the *Lurker* robots [2] used in the RoboCup rescue league. This control software uses the *IPC* communication framework [6], which is a very popular event-based communication library used by a number of robotic research labs worldwide. However, the algorithm simply can be adapt to other event-based communication frameworks, such as for instance Miro. The next section describes in more detail how the model is extracted from the observed communication.

2 Model Learning

Control systems based on IPC use an event-based communication paradigm. Software modules which wants to provide data are publishing an event containing the data. Other software modules which like to use this data, subscribe for the appropriate event and get automatically informed when such an event is available. A central software module of IPC is in charge for all aspects of this communication. Moreover, this software module is able to record all the communication details. It is able to record the type of the event, the time the event was published or consumed, the content of the event, and the names of the publishing and the receiving module.

[1] Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee, D-79110 Freiburg, Germany, kleiner@informatik.uni-freiburg.de

[2] Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b/II, A-8010, Austria, {steinbauer,wotawa}@ist.tugraz.at

The collected data is the basis for our model learning algorithm. Figure 1 depicts such collected data for a small example control software comprising only 5 modules with a simple communication structure. This example will be used in the following description of the model learning algorithm. The control software comprises two data path. One is the path for the self-localization of the robot. The two modules in the path *Odometry* and *SelfLoc* provide data on a regular basis. The other is the path for object tracking. The module *Vision* provides new data on a regular basis. The module *Tracker* provides data only if new data is available from the module *Vision*. The figure shows when the different events were published. Based on this recorded communication we extract the communication model step by step.

2.1 The communication graph

At a first step the algorithm extract a communication graph from the data. The nodes of the graph are the different software modules. The edges represent the different events which are exchanged between the modules. Each event is represented by at least one edge. If the same event is received by multiple modules, there is an edge to every receiving module originating from the publishing module. Figure 2 depicts the communication graph for the above example. This graph shows the communication structure of the control software. Moreover, it shows the relation of inputs and outputs of the different software modules because each node knows its connections. Such a communication graph is not only useful for diagnosis purposes, but it is also able to expressively visualize the relation of modules from a larger or partially unknown control software.

Formally the communication graph can be defined as following:

Definition 1 (CG) *A communication graph (CG) is a directed graph with the set of nodes M and the set of labeled edges C where:*

- *M is a set of software modules sending or receiving at least one event.*
- *C is a set of connections between modules, the direction of the edge points from the sending to the receiving module, the edge is labeled with the name of the related event.*

Please note that the communication graph may contain cycles. Usually such cycles emerge from acknowledgement mechanisms between two modules.

The algorithm for the creation of the communication graph is straightforward. The algorithm starts with an empty set of nodes M and edges C. The algorithm iterates trough all recorded communication events. If either the sender or the receiver are not in the set of the nodes the sender or the receiver is added to the set. If there is no edge pointing from the sending to the receiving node with the proper label, a new edge with the appropriate label is added between the two modules.

Moreover, we define the two functions $in: CO \mapsto 2^C$ which returns the edges pointing to a node and the function $out: CO \mapsto 2^C$ which returns the edges pointing from a node.

2.2 The communication behavior

In a next step the behavior or type of each event connection is determined. For this determination we use the information of the node the event connection comes from, and the recorded information of the event related to the connection, and all events related to the sending node.

We can distinguish the following types: triggered event connection (1), periodic event connection (2), bursted event connection (3) and random event connection (4). In order to describe the behavior of a connection formally we define a set of connection types $CT = \{periodic, triggered, bursted, random\}$ and a function $ctype: C \mapsto CT$ which returns the type of a particular connection $c \in C$.

The type of a event connection is determined by tests like measurements of the mean and the standard deviation of the time between the occurrence of the events on the connection, and comparison or correlation of the occurrence of two events. The criteria used to assign an event connection to one of the four categories are summarized below:

triggered A triggered event only occurs if its publishing module recently received a trigger event. In oder to determine if an event connection is a triggered event connection, the events on connection $c \in out(m)$ are correlated to the events on the set of input connection to the software module $I = in(m)$. If the number of events on connection c, which are correlated with an event on a particular connection $t \in in(m)$, exceed a certain threshold, connection t is named as trigger of connection c. The correlation test looks for the occurrence of the trigger event prior the observed event. Note each trigger event can only trigger one event. If connection c is correlated with at least one connection $t \in in(m)$ connection c is categorized as a triggered connection. Usually, such connections are found in modules doing calculations only if new data are available.

periodic On a periodic event connection the same event regularly occurs with a fixed frequency. We calculate from the time stamps of the occurrence of all events a discrete distribution of the time difference between two successive events. If there is a high evidence in the distribution for one particular time difference, the connection is periodic with a periodic time of the estimated time difference. For a pure periodic event connection one gets a distribution close to a Dirac impulse. Usually, such connections are found with modules providing data at a fixed frame rate, such as a module sending data from a video camera.

bursted A bursted event is similar to the periodic event but its regularly occurrence can be switched on and off for a period of time. A event connection is classified as bursted if there exist time periods where the criteria of the periodic event connection hold. Usually, such connections are found with modules which do specific measurements only if the central controller explicitly enable them, e.g., a complete 3d laser scan.

random For random event connections none of the above categories match and therefore no useful information about the behavior of that connection can be derived. Usually, such

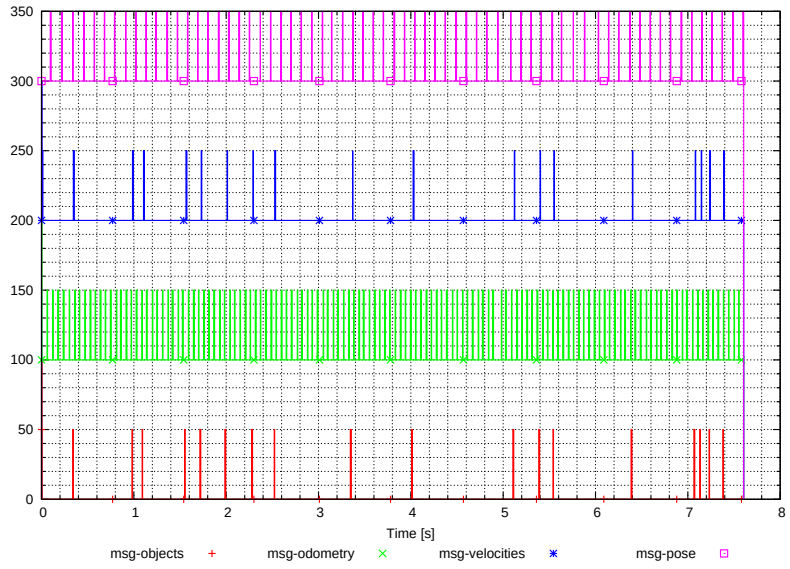

Figure 1. Recorded communication of the example robot control software. The peaks indicate the occurrence of the particular event.

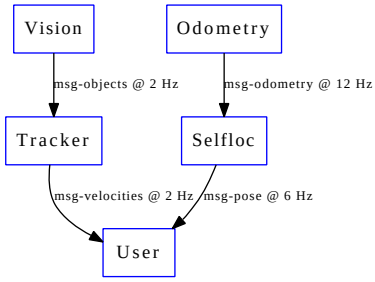

Figure 2. Communication graph learned from the recorded data of the example control software.

connections are found in modules which provide data only if some specific circumstance occur in the system or its environment.

In the case of the above example, the algorithm correctly classified the event connections *odometry*, *objects* and *pose* as periodic and the connection *velocity* as triggered with the trigger *objects*.

2.3 The observers

In order to be able to monitor the actual behavior of the control software, the algorithm instantiates an observer for each event connection. The type of the observer is determined by the type of the connection and its parameters, estimated by the methods described before. An observer rises an alarm if there is a significant discrepancy between the currently observed behavior of an event connection and the behavior learned beforehand during normal operation. The observer provides as an observation O the atom $ok(l)$ if the behavior is within the tolerance and the atom $\neg ok(l)$ otherwise. Where l is the label of the corresponding edge in the communication graph. The observations of the complete control OBS software is the union of all individual observations

$$OBS = \bigcup_{i=1}^{n} O_i$$

where n is the number of observers.

The following observers are used:

triggered This observer raises an alarm if within a certain timeout after the occurrence of a trigger event no corresponding event occurs or if the trigger event is missing prior the occurrence of the corresponding event. In order to be robust against noise, the observer uses a majority vote for a number of succeeding events, e.g, 3 votes.

periodic This observer raises an alarm if there is a significant change in the frequency of the events on the observed connection. The observer checks if the frequency of successive events does vary too much from the specified frequency. For this purpose, the observer estimates the frequency of the events within a sliding time window.

bursted This observer is similar to the observer above. It differs in the fact that this observer starts the frequency check only if events occur and does not raise an alarm if no events occur.

random This is a dummy observer which alway provides the observation $ok(l)$. This observer is implemented for completeness.

2.4 The system description

The communication graph together with the type of the connections is a sufficient specification of the communication behavior of the robot control software. This specification can be used in order to derive a system description for the diagnosis process. It is a description of the desired or nominal behavior of the system. In order to be able to be used in the diagnosis process, the system description is automatically written down as a set of logical clauses. This set can easily be derived from the communication graph and the behavior of the connections.

The algorithm to derive the system description starts with an empty set SD. For every event connection in two steps, clauses are added to the system description. In the first step, a clause for forward reasoning is added. The clause specifies if a module works correct and all related inputs and outputs behave as expected. Depending on the type of the connection, we add the following clause to the SD. If connection c is

triggered, we add the clause

$$\neg AB(m) \bigwedge_{t \in trigger(c) \wedge t \in in(m)} ok(t) \rightarrow ok(c)$$

and the clause

$$\neg AB(m) \rightarrow ok(c)$$

otherwise. $\neg AB(m)$ means that the module m is not abnormal and the module works as expected. The atom $ok(c)$ specifies that the connection c behaves as expected.

In a second step, a clause for backward reasoning is added. The clause specifies if all output connections c' of module m behave as expected the module itself has to behave as expected. We add the clause

$$\bigwedge_{c' \in out(m)} ok(c') \rightarrow \neg AB(m)$$

Figure 3 depicts the system description obtained for the above example control software.

3 Model-based diagnosis

For the detection and localization of faults we use the consistency-based diagnosis technique of [5]. A fault detectable by the derived model causes a change in the behavior of the system. If such an inconsistency between the modeled and observed behavior emerges, a failure has been detected. Formally, we define this by:

$$SD \cup OBS \cup \{\neg AB(m) | m \in M\} \models \bot$$

where the latter set says that we assume that all modules work as expected.

In order to localize the module responsible for the detected fault, we have to calculate a diagnosis Δ. Where Δ is a set of modules $m \in M$ we have to declare as faulty (change $\neg AB(m)$ to $AB(m)$) in order to resolve the above contradiction. We use our implementation [3] of this diagnosis process for the experimental evaluation of the models. Please refer to [8, 7] for the detail of the diagnosis process.

4 Experimental Results

In order to show the potential of our model learning approach, the approach has been tested on three different types of robot control software. We evaluated whether the approach is able to derive an appropriate model reflecting all aspects of the behavior of the system. The derived model was evaluated by the system engineer who has developed the system. Moreover, we injected artificial faults like module crashes in the system, and evaluated if the fault can be detected and localized by the derived model.

4.1 A small example control software

The example software from the introduction comprises five modules. The module *Odometry* provides odometry data at a regular basis. This data is consumed by the module *SelfLoc*, which does pose tracking by integrating odometry data, and providing continuously a pose estimate to a visualization module *User*. The module *Vision* provides position measurements of objects. The module *Tracker* uses this measurements to estimate the velocity of the objects. New velocity estimations are only generated if new data is available. The velocity estimates are also visualized by the GUI. Figure 1 shows the recorded communication of this example. Figure 2 depicts the communication graph extracted from the recorded data. It correctly represents the actual communication structure of the example, and shows the correct relation of event producers and event consumers.

Moreover, the algorithm correctly identified the type of the event connections. This can be seen by the system description the algorithm has derived which is depicted in Figure 3. It also instantiated the correct observer for the four event connections. A periodic event observer was instantiated for *odometry*, *objects* and *pose*, and a triggered event observer was instantiated for *velocities*.

1. $\neg AB(Vision) \rightarrow ok(objects)$
2. $\neg AB(Odometry) \rightarrow ok(odometry)$
3. $\neg AB(Tracker) \wedge ok(objects) \rightarrow ok(velocities)$
4. $\neg AB(Selfloc) \rightarrow ok(pose)$
5. $ok(objects) \rightarrow \neg AB(Vision)$
6. $ok(odometry) \rightarrow \neg AB(Odometry)$
7. $ok(velocities) \rightarrow \neg AB(Tracker)$
8. $ok(pose) \rightarrow \neg AB(Selfloc)$

Figure 3. The system description automatically derived for the example control software.

Figure 3 depicts the extracted system description. Clauses 1 to 4 describe the forward reasoning. Clauses 5 to 8 describe the backward reasoning. Clause 3 states that the module *Tracker* works correctly only if a velocity event occurs only after trigger event. For instance, Clause 6 states that if all output connections of module *Odometry* work as expected, consequently the module itself works correct. This automatically generated system description was used in some diagnosis tests. We randomly shutdown modules and evaluate if the fault was correctly detected and localized. For this simple example the faults were always proper identified.

4.2 Autonomous exploration robot Lurker

In a second experiment we recorded the communication of the control software of the rescue robot Lurker [2] while the robot was autonomously exploring an unknown area. The robot is shown in Figure 4.

The control software of this robot is far more complex as in the example above, since it comprises all software modules enabling a rescue robot to autonomously explore an area after a disaster. Figure 5 shows the communication graph derived from the recorded data, clearly showing the complex structure of the control software.

From the communication graph and the categorized event connections a system description with 70 clauses with 51 atoms and 35 observers was derived. After a double check with the system engineer of the control software it was confirmed that the automatically derived model maps the behavior of the system.

[3] The implementation can freely be downloaded at http://www.ist.tugraz.at/mordams/.

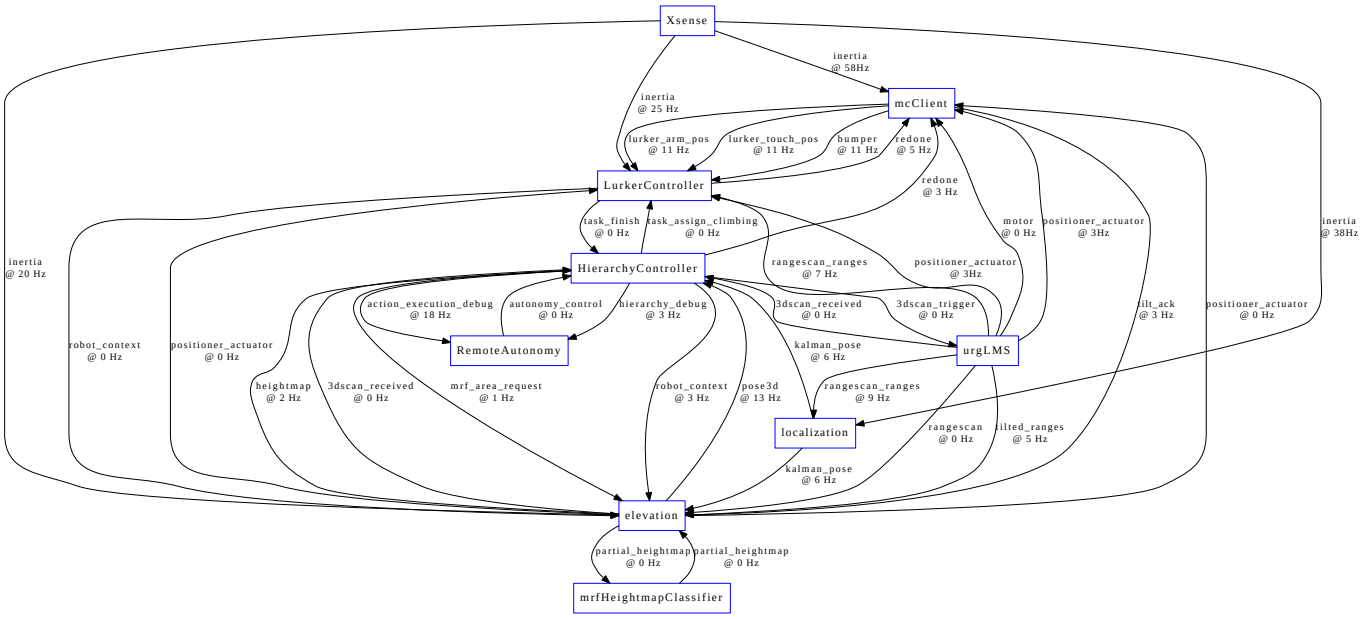

Figure 5. Communication graph Lurker robot.

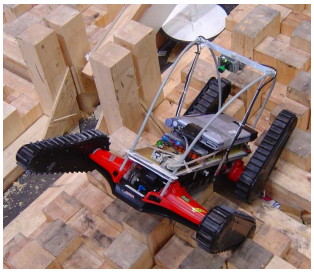

Figure 4. The autonomous rescue robot Lurker of the University of Freiburg.

4.3 Teleoperation Telemax robot.

In a final experiment we record data during a teleoperated run with the bomb-disposal robot Telemax. The robot Telemax is shown in Figurer 6.

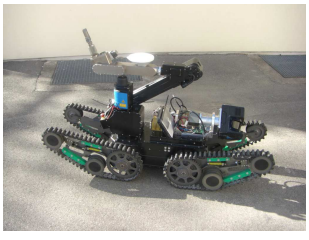

Figure 6. The teleoperated robot Telemax.

Figure 7 depicts the communication graph derived from the recorded data. It clearly shows that the control software for teleoperation shows a far less complex communication structure than in the autonomous service. From the communication graph and the categorized event connections a system description with 44 clauses with 31 atoms and 22 observer was derived.

5 Related Research

There are many proposed and implemented systems for fault detection and repair in autonomous systems. Due to lack of space we refer only a few. The Livingstone architecture by Williams and colleagues [4] was used on the space probe *Deep Space One* to detect failures in the probe's hardware and to recover from them. Model-based diagnosis also has been successfully applied for fault detection and localization in digital circuits and car electronics and for software debugging of VHDL programs [1]. In [3] the authors show how model-based reasoning can be used for diagnosis for a group of robots in the health care domain. The system model comprises interconnected finite state automata. All these methods have in common that the used models of the system behavior are generated by hand.

6 Conclusion and Future Work

In this paper we presented an approach which allows the automated learning of communication models for robot control software. The approach uses recorded event communication. The approach is able to automatically extract a model of the behavior of the communication within a component-orientated control software. Moreover, the approach is able to derive a system description which can be used for model-based diagnosis. The approach was successfully tested on *IPC*-based

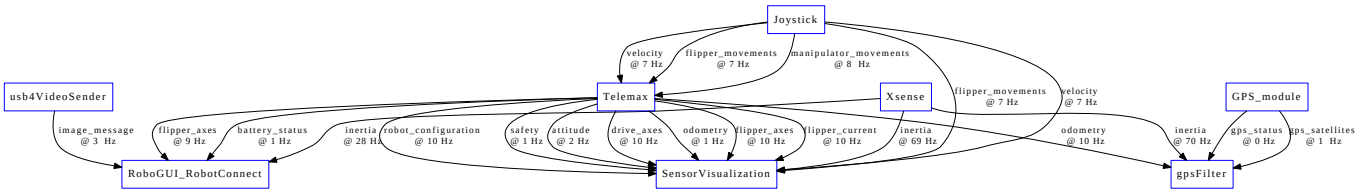

Figure 7. Communication graph Telemax robot.

robot control software like the rescue robot Lurker. *IPC* is a widely used basis for robot control software. Therefore, our approach is instantly usable on many different robot systems.

Currently, we are working on a port for *Miro*-based systems. This even will increase the number of potential target systems of our approach. Moreover, we work on the recognition of additional event types in order to enrich the generated models.

We believe that the consideration of the content of the events will lead to significantly better models and diagnosis. For the modeling the techniques of Qualitative Reasoning seem to be promising. But it is an open question how such qualitative models can be automatically learned from recorded data.

REFERENCES

[1] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa, 'Model-based diagnosis of hardware designs', *Artificial Intelligence*, **111**(2), 3–39, (1999).

[2] Alexander Kleiner and Christian Dornhege, 'Real-time Localization and Elevation Mapping within Urban Search and Rescue Scenarios', *Journal of Field Robotics*, (2007).

[3] Roberto Micalizio, Pietro Torasso, and Gianluca Torta, 'Online monitoring and diagnosis of a team of service robots: A model-based approach', *AI Communications*, **19**(4), 313 – 340, (2006).

[4] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams, 'Remote agent: To boldly go where no AI system has gone before', *Artificial Intelligence*, **103**(1-2), 5–48, (August 1998).

[5] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[6] Reid Simmons, 'Structured Control for Autonomous Robots', *IEEE Transactions on Robotics and Automation*, **10**(1), (1994).

[7] Gerald Steinbauer, Martin Mörth, and Franz Wotawa, 'Real-Time Diagnosis and Repair of Faults of Robot Control Software.', in *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pp. 13–23. Springer, (2006).

[8] Gerald Steinbauer and Franz Wotawa, 'Detecting and locating faults in the control software of autonomous mobile robots.', in *16th International Workshop on Principles of Diagnosis (DX-05)*, pp. 13–18, Monetrey, USA, (2005).

[9] Hans Utz, *Advanced Software Concepts and Technologies for Autonomous Mobile Robotics*, Ph.D. dissertation, University of Ulm, Neuroinformatics, 2005.

[10] Hans Utz, Stefan Sablatng, Stefan Enderle, and Gerhard K. Kraetzschmar, 'Miro – middleware for mobile robot applications', *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, **18**(4), 493–497, (August 2002).