 DOI: 10.3217/978-3-85125-524-9-03

A Model-Based Fault Detection, Diagnosis and Repair for Autonomous Robotics systems

Stefan Loigge[1] and Clemens Mühlbacher[1] and Gerald Steinbauer[1] and Stefan Gspandl[2] and Michael Reip[2]

***Abstract*— Autonomous robots comprise of several complex software and hardware components which interact with the environment to fulfill a certain task. Due to the non-determinism, inherent of the environment and complexity of the components one cannot expect that the robot will never show a fault. Instead one needs to deal with the occurrence of faults in the robotics system. As we focus on autonomous robots the robot should deal with faults in an automated fashion.**

In this paper, we present a model-based fault detection and diagnosis method with a simple but powerful method to repair faults. Using this method, the robot can detect and react to faults in a timely manner. Furthermore, no human intervention is necessary thus allowing the robot to be autonomous. As not every repair can be performed by the robot itself the system allows the robot also to inform the maintenance staff which repairs are necessary. Thus, this approach reduces the time for fault localization of the maintenance staff.

I. Introduction

Autonomous robots perform tasks in (partly) open environments. To perform such a task, the robot uses several complex software and hardware components which interact with each other. Due to the (partly) open environment and the complex components, one cannot assume that no fault will occur. Instead one needs to design the robotic system with faults in mind. Thus, one either add fault handling in each component or one uses a more general approach. One such general approach is the use of a model-based approach as outlined in [1]. The model is used to describe the system behavior and to allow the system to detect a fault.

The use of a model-based approach allows the robot to determine if a fault has occurred. Furthermore, the robot can determine which component most likely caused this fault. Using the information which component is faulty the robot can determine which action to perform to react to this fault. Besides the possibility that the robot detects and reacts to a fault a model-based approach also allows to separate the current system description from the fault detection and localization components. As the model is used to describe the system the fault detection and localization can be done on the model only. Thus, one can use the software to perform this reasoning for many different robots without changes. The only thing which needs to be changed for a robot is the model of the robot. As many robotic system reuse components of other robots, or have similar robot components one can often reuse parts of already existing models. Thus, further decreasing the effort to perform fault detection and localization.

In this paper, we present such a model-based diagnosis approach. The method uses several different observers to observe properties of the system. These properties are observed to detect a fault. With the help of the observed properties, the system can derive a diagnosis which component caused the fault. This allows pinpointing the fault without extra costs as the only information necessary for the diagnosis is already provided through the definition of the observations. To allow the robot to react to a detected fault a simple rule engine can be used. The rule engine allows the robot to react fast to a fault and to trigger more complex repair actions. Through this fast reaction, one can reduce the chance that a robot will endanger itself or pose a threat to its surrounding.

The remainder of the paper is organized as follows. In the next section, we will give an overview of the fault detection, diagnosis, and repair system. The proceeding section discusses the different observers which check system properties in more detail. Afterward, we discuss the diagnosis engine which is used to identify the faulty component. In Section V we discuss the rule engine and how it can be used to react to faults. In Section VI, we show a use case where the system was used on an industrial robotics system. Before we conclude the paper, we discuss some related research. Finally, we conclude the paper and point out some future work.

II. System Overview

To create a robotic system, the robot operating system (ROS) [2] is often used as a framework. With the help of ROS one can use several software components, which are called nodes, and interact with each other. This interaction can be performed with the help of publisher-subscriber principle which allows exchanging message between each ROS node. To define and identify for such communication channel ROS uses so-called topics. These are strings defining an n-to-n communication channel. Furthermore, one can use service calls to provide a service from one component to another. In the remainder of the paper, we will focus on messages exchanged by topics as these are used more often as services and allow an easy introspection.

Using ROS, a robotic system can be created which uses several software components interacting with each other. As we are interested in detecting and identifying faults and react to these faults we use the system depicted in Figure 1. The

[1]Stefan Loigge, Clemens Mühlbacher and Gerald Steinbauer are with the Institute for Software Technology, Graz University of Technology, Graz, Austria. {sloigge,cmuehlba,steinbauer}@ist.tugraz.at
This work is partly supported by the Austrian Research Promotion Agency (FFG) under grant 843468.
[2]Stephan Gspandl and Michael Reip are with incubedIT, Hart bei Graz, Austria. {gspandl,reip}@incubedit.com

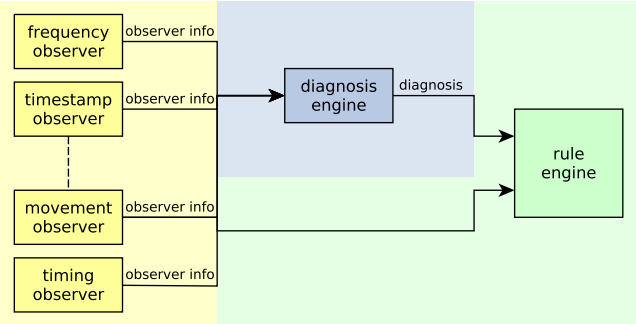

Fig. 1. Monitoring, diagnosis, and fault handling overview: observer (yellow), diagnosis engine (blue), rule engine (green), [4]

system consists of three parts. A set of observers which is used to detect a fault. A diagnosis engine which identifies the component which caused the fault. The usage of observers and a diagnosis engine for a ROS was already proposed in [3] and was extended in this paper. Finally, a rule engine is used to react to faults.

To allow the method to be applied for already existing software, it is of interest that the used software components are not needed to be altered. Thus, instead of detecting a fault in the software components directly, we use information provided by the interaction of the software components. This allows that we can detect a fault without changing existing software components. This can be simply achieved in ROS by introspection on the topics which are used for the communication. By observing properties of a topic, e.g. frequency of communication on a topic, the system can be checked if it conforms to the given model. This observation is provided using different observers where each observer is used to determine if a specific property hold. We will discuss in the next section in more detail which observers exist.

With the observations, only the robot would only be able to detect that a fault has occurred. But the robot needs also to determine which component caused the fault. This is of special interested if several malfunctions are detected at the same time. With the help of the model of the system and a reasoning process, the diagnosis engine determines which components are faulty. The reasoning performed uses a consistency-based diagnosis [5] approach which searches for a minimal set of components which are blamed for being faulty explain the observations. We discuss the diagnosis engine in more detail in Section IV.

After the robot, has determined which components might have caused the fault the robot needs to react to this fault. This is achieved with the help of a rule engine which uses the current diagnosis of the system together with the observations. By combining the diagnosis and the observations the rule engine can determine which rule should be triggered to execute a specific repair. This allows the robot to react in a timely manner. If a planning system would be used as it was described in [3] a possible high planning time may not allow such a fast reaction. Due to this reaction, the robot can bring itself into a safe state which can be used afterward to perform a more complex repair. Let's consider a simple example. The robot detects that the laser scanner used for navigation is malfunctioning. After determining this malfunction, the robot can react and stop immediately. Thus, the robot will not drive into an obstacle. After the robot, has stopped, the robot can perform a more complex reasoning which repair should be performed with another method [3]. Or it may even try to reconfigure itself to deal with the fault [4]. In this paper, we will focus only on a quick reaction to a fault and not a complex repair or reconfiguration mechanism. We will discuss the rule engine in more detail in Section V.

The complete system as it is described in this paper is public available under `http://git.ist.tugraz.at/ais/model_based_diagnosis`.

III. Observers

As outlined above we use several observers to check if a certain property of the robotic system holds. These observers are used to mediate between the concrete messages send in the robotic system and the abstract model of the system. This allows that the model of the system uses a predicate based representation of the robotic system which simplifies the diagnosis process. Furthermore, the observers can use specifically design methods to observe a certain property allowing a small computation overhead to provide the observations.

To properly supervise the system different types of observers are used. Some observers observe the behavior of a node directly where others observe the behavior or the message exchanged. To observe the behavior of the node directly two observers can be used.

- The activated observer checks if a node is present in the robotic system. Thus, allowing to check if the system is properly configured.
- the resource observer checks if a specific node in the system uses a predefined amount of system resources, e.g. CPU. This allows checking if the node neither consumes too many resources, e.g. a memory leak causing the accumulation of memory nor the consumption of too fewer resources, e.g. no CPU usage as the node has deadlocked itself.

To observe the behavior of the message exchange in the system the following six observers can be used.

- The time-out observer checks if at least one message was sent within a specified time interval. This allows checking if a topic is used for communication and performs a watchdog functionality for a topic. Thus, allowing to revival problems which cause the communication to break down, e.g. the node which should send an information can't produce an output.
- The HZ observer checks on a topic if messages are exchanged with a given frequency. This allows checking if a communication is done on a regular basis. Thus, allowing to check if the node which provides the information is overloaded.
- The time-stamp observer checks if the timestamp of a message send is not too old. This allows checking if

old data are sent in the system thus reveal problems to produce new data.
- The timing observer checks the time difference between one message send on one topic and one message sends on another topic. This can be used to check if a node produces an expected output within the expected time frame. Thus, one can detect if a processing step takes too long.
- The score observer checks if the float value of a topic is within a range. This allows checking the calculated score, specifying the performance of an algorithm outcome.
- The movement observer uses two topics which specify the movement of the robot for correlation. This correlation can be used to check if the expected movement differs significantly, e.g. the movement measured by the IMU is different to the movement measured by the odometry.

Using the different observer types different properties of the system can be checked. As the observations, may be subject to noise one cannot simply use the raw values to perform the check. Instead one can apply different filter mechanism to process the raw values before performing a check. Thus, the raw value to check, e.g. the frequency of a topic is treated as a signal which needs to be filtered as it is common in signal processing [6].

After filtering the raw values of the observation one needs to perform a check to determine if the observed values are acceptable. This can be done by simple checks which determine the correctness using comparison with a fixed value. But it is also possible to use a more complex test which uses a statistical approach. This is done by performing a student-t-test [7] on the filtered data. Through this test one can check if the hypothesis that the observation is acceptable needs to be withdrawn. Thus, allowing to perform a check considering the statistical uncertainty.

All except one observer type check the raw value observed with a nominal value of the mode, e.g. the frequency of a topic with the expected value. The movement observer is the exception, as it correlates two values with each other. The idea is to use the redundant information in the robotic system to check for consistency. This follows the idea of residuals [8] which create an error term between redundant information in the system. To do so, we first derive from each movement measurement the resulting acceleration. Thus, if the movement is given by the current velocity the movement is differentiated to get the acceleration. Afterward, the accelerations of one input are subtracted from the other input. If no fault occurs this value is zero. Due to the noise measurement, the value follows a Gauss distribution with zero mean. With the help of the filter methods, one can estimate the mean of the distribution and use this estimation to perform a check if the value is close enough to zero.

IV. Diagnosis Engine

Using the observers one can detect if one property of the system behaves not as defined. This allows to detect a fault but does not allow to isolate the faulty component directly. Instead one needs to perform a reasoning. We use the idea of consistency-based diagnosis [5] to perform this reasoning. The reasoning uses the information about the observations taken from the system as well as the topology of the system. This allows handling fault propagation properly. To specify the system, we define a system to consists of a set $\mathcal{N}$ defining the nodes of the system. These are the software components which are running and need to be diagnosed. Additionally, the system consists of a set $\mathcal{M}$ defining the topics which are used to exchange messages between the software components. To represent the input topics to a node we use the function $input : \mathcal{N} \rightarrow 2^{\mathcal{M}}$. The output which is produced by a node is defined through $output : \mathcal{N} \rightarrow 2^{\mathcal{M}}$. Using the set $\mathcal{N}$, and the functions $input$ and $output$ one can describe the information flow of the system. This information flow is of interest as a fault can be propagated along this information flow.

To define a software component n to be faulty we use the predicate $AB(n)$. Besides the software component also a topic can be observed to be faulty thus we write $AB(m)$ that on observation indicate that the message exchange m is not as expected. Please note that we are only interested in the predicates $AB(n)$ which are used to explain a faulty behavior. Thus, we will search for a minimal set of $AB(n)$ predicates which explain the observations.

To specify the fault propagation, we use the following logical formula which is defined for each $n \in \mathcal{N}$.

$$\forall m_o \in output(n) : AB(m_o) \rightarrow \left(AB(n) \bigvee_{m_i \in input(n)} AB(m_i) \right)$$

The formula states that if the output of a software component seems to be faulty either the component is faulty or one of its inputs where faulty. Thus, one can propagate the fault from input to output.

With the help of the above formula, we can define the fault propagation in the system per the structure of the system. Besides the structure of the system, we need also to define how the observations made a link to the components in the system. This link depends on the type of observation made. We use the following formulas to link the observations and the components of the system.

- If component n is observed with the help of an activated observer ($obs_{activated}(n)$) we state the following logical formula.

  $$\neg obs_{activated}(n) \rightarrow AB(n)$$

  As we directly observe the component we can detect that the component is faulty if the observation indicates a fault.
- If component n is observed with the help of a resource observer ($obs_{resource}(n)$) we state the following logical formula.

  $$\neg obs_{resource}(n) \rightarrow AB(n)$$

As we directly observe the component we can detect that the component is faulty if the observation indicates a fault.

- If a topic m is observed with the help of a time-out observer ($obs_{timeout}(m)$) we state the following logical formula.

$$\neg obs_{timeout}(m) \rightarrow AB(m)$$

As we only observe a topic we can only state that the topic is abnormal and use the structure to determine which component caused this fault.

- If a topic m is observed with the help of an HZ observer ($obs_{hz}(m)$) we state the following logical formula.

$$\neg obs_{hz}(m) \rightarrow AB(m)$$

As we only observe a topic we can only state that the topic is abnormal and use the structure to determine which component caused this fault.

- If a topic m is observed with the help of a time-stamp observer ($obs_{timestamp}(m)$) we state the following logical formula.

$$\neg obs_{timestamp}(m) \rightarrow AB(m)$$

As we only observe a topic we can only state that the topic is abnormal and use the structure to determine which component caused this fault.

- If two topics m_1 and m_2 are observed with the help of a timing observer ($obs_{timing}(m_1, m_2)$) we state the following logical formula.

$$\neg obs_{timing}(m_1, m_2) \rightarrow (AB(m_1) \vee AB(m_2)).$$

If the timing of the two topics does report an error one of the topics need to cause the fault. As we only observe that at least one of the topics need to be abnormal we need to use the structure to determine which component caused this fault.

- If a topic m is observed with the help of a score observer ($obs_{score}(m)$) we state the following logical formula.

$$\neg obs_{score}(m) \rightarrow AB(m)$$

As we only observe a topic we can only state that the topic is abnormal and use the structure to determine which component caused this fault.

- If two topics m_1 and m_2 are observed with the help of a movement observer ($obs_{movement}(m_1, m_2)$) we state the following logical formula.

$$\neg obs_{movement}(m_1, m_2) \rightarrow (AB(m_1) \vee AB(m_2) \vee AB(movement))$$

The formula states that if the movement is observed to be faulty then either one of the topics is abnormal or the movement relation is not valid. The movement relation may not be valid as we may observe the difference between the IMU and the odometry. If the robot now slips the odometry and the IMU do no longer agree but none of the components is faulty. Instead, the model of the environment imposing that these two sources of information are redundant does not longer hold.

With the logical formulas from above, the model of the system is described. Furthermore, the link between the observations and the model of the system is defined through the logical formulas from above. With the help of this logical formula, one can derive which set of $AB(n)$ predicates is consistent. This set represents the software components which need to be faulty to explain the observed faults. As we are interested in the most likely explanation we follow the idea of Occams razor and search for a minimal set of $AB(n)$ predicates which are consistent.

To find this minimal set we use a minimal hitting set algorithm. The algorithm uses a sat solver to derive if a set of $AB(n)$ predicates is consistent. If the set of predicates is consistent the algorithm has found a diagnosis. Otherwise, the algorithm uses the predicates $AB(n)$ which are part of the conflict in the checked set of $AB(n)$ predicates to choose the next $AB(n)$ to add to the set to avoid this conflict. Due to this conflict-driven search, the algorithm can derive a minimal set in an efficient manner [5]. To perform the necessary calculations of the algorithm we use the implementation of [9].

V. Rule Engine

After detecting a fault and identifying the faulty components the robot needs to react to this fault. To deal with faulty components the robot needs either to perform a repair action [3] or change the configuration of the robotic system [4] to deal with this problem. In either case, it takes some time to deal with the fault properly. This can cause the robot to operate in an unknown state in an unsafe manner. Thus, the robot needs first to react swiftly to bring the robotic system in a known a safe state. This imposes that the robotic system will not harm itself or its environment. Additionally, often such a reaction is sufficient as some faults cannot be fixed by the robot itself, e.g. a broken wheel.

To allow the robot to perform a fast reaction we propose a simple but powerful rule engine. The simplicity of the rule engine is not only due to the simple model how the robot should react but also due to the limited reasoning which is performed to choose the reaction. This restricts the possible reactions of a robot but allows to perform the reactions fast without a large computation overhead. The reaction triggered by the rule engine is a kind of reflex of the robot. Thus, only preventing it from further harm if possible.

To perform the reaction, the rule engine uses a set Obs of the observations made so far. The set is updated with each incoming observation to ensure that only one observation per component/topic for a specific type is present. This update also ensures that only the newest information is used. To trigger the rules an additional set is used, the set $PosAb$ of components which are possibly faulty. The set defines those components which are part of a minimal diagnosis. Thus, if one has two diagnoses $\{\{m_1\}, \{m_2\}\}$ the set of possibly faulty components consist of the elements of both diagnosis ($\{m_1, m_2\}$. This set simplifies reasoning as one

does not reason over different diagnosis but only over the set of components which may be faulty. The components which may be the faulty need either to be observed more closely or need to be repaired. Additionally, one cannot assume that this component works properly with the information given so far. Thus, this set is sufficient to decide which action to execute.

The rule engine consists of a set of rules $\mathcal{R}$ where each rule r is a tuple comprising the following elements.

- A set $posObs$ defining observations which should have been made
- A set $negObs$ defining observations which should not have been made
- A set $posPosAb$ which is a set of components which should have been diagnosed as possibly faulty
- A set $negPosAb$ which is a set of components which should not have been diagnosed as possibly faulty
- α an action to execute.

Due to the use of the sets, one can simply perform the reasoning by intersecting the sets to determine if the rule should be triggered. As some observations, may be missing one may face the problem that neither $obs_{resource} \in Obs$ nor $\neg obs_{resource} \in Obs$ holds, thus one cannot take a decision if the observation of the resource is true or false. If one would strictly perform the reasoning a rule may not triggered because $obs_{resource} \notin Obs$ holds although $\neg obs_{resource} \notin Obs$ holds. This is of special interest as not every observer may state regularly which observations are true but only state which observations are false. To deal with this problem we trigger a rule if no contradicting information is observed. This is achieved by the following simple procedure.

Trigger the rule if neither of the following holds.

- $\overline{posObs} \cap Obs \neq \emptyset$, where $\overline{posObs} = \{\neg po | po \in posObs\}$
- $negObs \cap Obs \neq \emptyset$
- $\overline{posPosAb} \cap PosAb \neq \emptyset$, where $\overline{posPosAb} = \{\neg posAb | posAb \in posPosAb\}$
- $negPosAb \cap PosAb \neq \emptyset$

As only set operations are performed one can perform an efficient reasoning which allows a fast reaction. Especially as one can assume that the sets $posObs$, $negObs$, $posPosAb$ and $negPosAb$ are small. Thus, one can perform this checks in $\mathcal{O}(|posObs| * log(Obs) + |negObs| * log(Obs) + |posPosAb| * log(PosAb) + |negPosAb| * log(PosAb))$ which allows a fast reaction even in case of many observations or many possible faulty components.

As rules, should only be used to allow the robot to react to faults, instead of continuously checking the rules, they are only checked if the set of observations or possible faulty components changes. This allows to save resources but also prohibits to trigger a rule multiple times without any change in the system.

After deciding that a rule should be triggered one needs to execute the action α which is defined for this rule. The actions range from printing a message to the console or to a log file over changing parameters to triggering the execution of an external script. Thus, one can trigger nearly arbitrary behavior to react to a fault.

VI. Use Case

Before we discuss related research, we will show a simple use case of the system. The use case is the simplified odometry calculation of a robot which delivery good in a warehouse, see [10] for a detailed description of the robot. The odometry is calculated using the wheel encoders and an IMU is used to improve this odometry. The IMU is fused with the calculation by using the rotation of the IMU instead of the calculated rotation. Thus, if the IMU is fault free the odometry is improved. To show the impact of the proposed system three faults is simulated. The IMU can either be stuck to zero after some time, it can overestimate the rotation by 20% or issue that there is no rotation after rotating a certain amount of time.

To evaluate the impact of the system the robot was commanded to move between six waypoints in the environment, for three minutes. During the movement, the wheel encoder the IMU measurements and the real position of the robot were determined. The real position of the robot was determined with the help of an OptiTrack system. After the movement of the robot was recorded the odometry is calculated using the wheel encoders and the IMU. Additionally, one observer is used which checks if the calculated rotation of the wheel encoder and the IMU correlate. This allows detecting a fault of the IMU. Using this fault detection, the diagnosis can calculate that the IMU is faulty. In such a case the rule engine changes a parameter to ensure that the IMU is no longer used for the odometry calculation.

The evaluation compares the error between the ground truth and the calculated odometry which always uses the IMU and the calculated odometry which only use the IMU if it is not diagnosed to be faulty. In case the IMU was stuck to zero after several seconds the mean error was reduced by 28.1% and the root mean squared error (RMS) was reduced by 39.9%. In case the IMU was overestimating the rotation by 20% the mean error was reduced by 25.6% and the root mean squared error (RMS) was reduced by 35.6%. In case the IMU was reporting zero rotation after one second of rotation the mean error was reduced by 39.3% and the root mean squared error (RMS) was reduced by 50.9%. Thus, the use of the diagnosis system could react quickly enough to improve the odometry calculation drastically. The evaluation was performed on an intel i5-2430M with 8 GB of RAM and took less than 2 % of the CPU.

VII. Related Research

We begin our discussion of related research with the method proposed in [11]. The method adds to each software module so-called software sensors. These sensors supervise the execution of a software component which is treated as a black box. Thus, the software component can be developed and tested independently from the sensors. During the execution, the software sensor checks for faults and report these faults on a diagnosis port. To ease the reuse of the

sensors these sensors uses interfaces which are specific to the type of information they are interested in, e.g. a state change in the component. The information provided by these sensors on the diagnosis port can afterward be used by a monitor. The monitor allows to view the sensing result and thus show which faults are present in the system. This contrasts with the method we propose in this paper as we use the information provided by the observer to calculate a diagnosis. Additionally, our observers allow checking for properties which need to hold between different components, e.g. the movement measured by the wheel encoder and by the IMU.

Another method to observe a robotic system was proposed in [12]. Each module in the system is accompanied with a detection module which checks if the module works as expected. This check is performed with the help of a residual calculation. If the residual is not zero a fault is detected. All the detected faults are gathered in a fault signature and used for fault identification. This identification is performed with the help of an incidence matrix. The matrix describes in a static manner which fault causes which observations. This contrasts with our approach as we do not assume that we can simply enumerate all possible observation and faults in a matrix. To react to a fault, the method presented in [12] reacts on the high-level which uses defined recovery actions, which are chosen per the severity of the fault. This is like our approach which use a simple rule engine to perform a reaction but delegates the fault handling to more complex reasoning whereas the rule engine allows a fast reaction.

A method which uses a rule system for observations was presented in [13]. The system defines safety rules which are checked during runtime. To define this rules a domain specific languages is used which allows defining conditions for the rules and which actions to trigger if a condition holds. The rules use information which is provided on different topics to define a safety rule. The actions are afterward executed on the robotic hardware and can be defined in the framework separately. The main difference to our system is that we separate the detection and the reaction to a fault. This allows us to use several observations to determine which component is faulty and afterward react depending on the faulty component.

As we have briefly outlined above our method is based on the method presented in [3]. The method presented in [3] also uses observers to detect a fault and a diagnosis engine to identify the fault component. Additionally, a planning system is used to repair if a fault is detected. Instead of using a planning system to find a proper repair we use a simple rule engine to allow the robot a fast reaction but also restricts the possible repairs which can be performed. To allow a fast reaction and a proper repair one can combine both methods and first react with the rule engine and afterward trigger a planning step for a proper repair. The other difference between the method presented in this paper and the method presented in [3] is the underlying implementation. The underlying implementation presented in this paper use plugin-based observers which are more efficient than the implementation of the observers used in [3].

VIII. Conclusion and Future Work

Autonomous robots perform tasks in a (partly) unknown environment. This is done by using several complex software and hardware components. These components need to proper function and properly interact with each other to allow the robot to achieve its task. Due to the complexity of the components and the (partly), unknown environment one cannot expect that the robot will perform its task without a fault. Instead one needs to address the problem of fault occurrence in the robotic system.

In this paper, we presented a model based approach which allows that the robot detects and identifies a fault. This is achieved by observing the communication between the components and checking this communication for specific properties. These properties are derived from the system and specify the proper function of the system. If a property indicates a fault a diagnosis engine is used to determine the minimal set of components which is faulty. Using the result of this diagnosis engine a simple rule engine can be used to allow the robot to react to a fault. This reaction can be used to repair the fault or to bring the robot in a safe state to perform a more complex repair action.

The current approach uses static properties of the system to determine if a fault has occurred. It is left for future work to extend this approach to also consider dynamic changes of the properties. This would allow to detect a malfunction in the dynamic behavior of the system as well as to determine a malfunction of a component which changes its static behavior per a defined system state.

References

[1] G. "Steinbauer and C. Mühlbacher, "Hands off - a holistic model-based approach for long-term autonomy," in *Workshop on AI for Long-Term Autonomy, 2016 IEEE International Conference on Robotics and Automation (ICRA)*.

[2] 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.

[3] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej, and S. Uran, "An integrated model-based diagnosis and repair architecture for ros-based robot systems," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 482–489.

[4] S. Loigge, "Unified and dependable robot control architecture based on ros," Master's thesis, Faculty of Computer Science and Biomedical Engineering, Graz University of Technology, 2016.

[5] R. Reiter, "A theory of diagnosis from first principles," *Artificial intelligence*, vol. 32, no. 1, pp. 57–95, 1987.

[6] A. V. Oppenheim, *Discrete-time signal processing*. Pearson Education India, 1999.

[7] Student, "The probable error of a mean," *Biometrika*, pp. 1–25, 1908.

[8] J. Gertler, *Fault detection and diagnosis in engineering systems*. CRC press, 1998.

[9] T. Quartisch and I. Pill, "Pymbd: A library of mbd algorithms and a light-weight evaluation platform." in *25th International Workshop on Principles of Diagnosis (DX-2014)*, 2014.

[10] C. Mühlbacher, S. Gspandl, M. Reip, and G. Steinbauer, "Improving Dependability of Industrial Transport Robots Using Model-Based Techniques," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[11] A. Lotz, A. Steck, and C. Schlegel, "Runtime monitoring of robotics software components: Increasing robustness of service robotic systems," in *Advanced Robotics (ICAR), 2011 15th International Conference on*. IEEE, 2011, pp. 285–290.

[12] D. Crestani, K. Godary-Dejean, and L. Lapierre, "Enhancing fault tolerance of autonomous mobile robots," *Robotics and Autonomous Systems*, vol. 68, pp. 140–155, 2015.

[13] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz, "Towards rule-based dynamic safety monitoring for mobile robots," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 207–218.