Integrate a framework for model-based robot diagnosis with an in-house web application for robot operation monitoring

Anna Rose Johny[1], Amine Bahlouli[2] and Chandrika Sundakampalayam Paramasivam[3]

Abstract— **For autonomous robots comprising sophisticated software and hardware components, to perform a task efficiently it has to interact with the dynamic environment. During such situations, it is plausible for the robot to fail when its operating in a nondeterministic environment. Therefore, one needs to tackle the occurrence of faults in the robotics system. One way to deal with this is to display the robot's components in a web application so that humans can monitor in a timely fashion where a diagnosis can be performed. In this work, we integrate an existing model-based diagnosis for ROS based robot systems. This model-based diagnosis software can be used to monitor, diagnose, and repair the desired parameters of the robot. The diagnosis engine processes various operations made by the model. After processing the diagnosis or observations rule engine follow a list of rules and checks for each of them to trigger events. So, the observation and diagnosis messages are published and subscribed and after some processing is stored in the database, which is then etched so that it can be displayed in the web application using the Flask web framework for diagnosis purposes.**

I. INTRODUCTION

For the autonomous robots working in a real-world environment, the ability to determine the faults and repairing those faults are beneficial. For this diagnosis, the robot needs to analyze various hardware and software components so as to make the system fault free. One cannot assume that the component showing undesired behavior may be due to defective software or hardware deadlocks or any other reasons that occur when robots operate in such a partly open environment. This situation could be because of robots operating in a dynamic environment also the sophisticated interactions within the robot modules. Therefore, for a robot to autonomously cope up with such problems, there's a necessity of monitoring the systems which can detect the fault and also simultaneously at run time can repair them. So, the robot's parameters and components can be visualized in a web application online by integrating the front end being the web page and the back end, the database having details on messages being published.

In this paper, ROS based robot system is leveraged to publish and subscribe to a set of topics from the config file. Robot Operating Systems – ROS is a famous open-source, meta operating system for our robot. It provides all the features that we expect from an operating system, also provides tools and libraries to build, write, and run codes across many computers. It uses the concept of computation graph from the computation nodes or processes that use a publisher-subscriber mechanism for communication. The major focus is on models on information about a list of nodes that are active, their properties, input and output relations of the nodes and how do these nodes communicate.

The objective of work is to perform the task of automatic diagnosis. ROS supports this approach since it can easily access information about the communication between nodes along with the data, or the content of exchanged messages and nodes involved. The recorded information is mainly which are the running nodes and their problems, patterns of communication, and data being exchanged. Details on which nodes are active are obtained from ROS, information on communication patterns is not directly available form ROS as it only provides details on exchange messages so analysis has to done to find the pattern of communication. For example, the nodes regularly communicate. Monitoring the parameters of the robot in a web application or web page is done using the Flask web framework written in Python. Flask is popular because of its' open-source platform. It has a set of libraries, modules that helps web developers to write applications without bothering on low-level details like protocol or the thread management. Flask framework has eased our work to display desired observation and diagnosis messages on the web page by interfacing the web server to the web application. The next part of the paper introduces briefly on working of the diagnosis system and the messages being published. In the next section explanation of the Flask web framework, to display the results of diagnosis in a web application so that the user can monitor the parameters of the robot online in a timely manner. In the following section the diagrammatic representation of various design descriptions. Later, it summarizes the proposed approach and discusses future work.

II. RELATED SEARCH

A significant body of work on diagnosis and repair for robots exists in the literature. The approaches differ mainly in application fields such as software, hardware, integration of systems, behavior, and whether diagnosis and repair are combined. The approach is a combination of part of [3], [1]. Work in [3] illustrates on basic concepts of model based diagnosis software to monitor and diagnose ROS based systems. Also, the details on creating observers for a node, launch file for resource observations, the configuration file for resource observation, creating observers for the topic, setting up diagnosis is elaborated in [3]. Authors in [1] presented the model based approach for fault diagnosis of robot control software using the model for communication between the components. As part of the repair and hardware diagnosis, the authors of [8] summarized a mechanism for fault diagnosis along with repair at run time of the robot drives. In paper [2] an integrated approach for diagnosis and repair of robot drives during run time is illustrated. Additionally provides a control framework capable to reconfigure drive-based control functions on the detected faults. It also uses the model based approach and can deal with faults in hardware and software. For the current work in this paper, the ROS framework is used and integrated with Flask web application to monitor results on a web page. Use of ROS to establish the communication between the nodes and understanding basic concepts was adopted from [6]. In [9] a hybrid diagnosis approach is presented where continuous sensor measurements for model estimation were combined with decision tree utilizing discrete components mode. It uses the correlation of signals to produce input for the discrete input for a discrete model-based approach. Authors of [10] illustrate the autonomous systems on line diagnosis of components. It compares a couple of sensor signals to find faulty components. It's a structural model and signals are linearly correlated. In contrast to our approach where only monitoring of the observer and diagnosis messages is done in a web application.

III. Model-based diagnosis system

Model-based diagnosis is the activity of locating malfunctioning components of a system solely on the basis of its structure and behavior. The model-based system implements fault diagnosis and repair system for both hardware and software components for ROS based robot system [1]. A model-based diagnosis system consists of mainly 5 modules such as a model server, a set of observers, a diagnosis engine, a repair engine, and a hardware diagnosis board. The figure shows the basic model of the model-based diagnosis system. Now we will discuss in detail various components used in the diagnosis engine.

The first module is a set of observers. Observers are used for monitoring the ROS based system. There are different types of observers for analyzing various factors associated with a system. The observation results are then published using /observations topic. The observations are published in the form of strings. The strings can be observations such as running node, an observation made correct, and so on. The observers can be diagnostic observer, hardware observer, general observer, node observers. Diagnostic observers use existing /diagnostics topics and publish these topics on /observations. This provides a connection between the ROS based diagnosis system and the diagnosis and repair engine. Hardware observers are used for observing the power supply provided by various channels. The observations using the hardware observer is being published using /observations. A general observer is used for subscribing to a particular topic using the inputs provided by the received messages. The topics being published by this general observer node is also /observations. The general observer is mainly concerned with the frequency of the running topic. Node observers are used for observing the running nodes. Again the topic published by the node observers are /observations. The parameters passed are in the form of the string, which provides the information about a node such as running or not. There are several other observers are also. Here we are concentrated on the above set of observers.

The next module is the diagnosis model server. This is an action server that provides the diagnosis model of the system. The model server is saved in the form of a YAML file. The topic being published is /diagnosis_model. This provides the system with a model for diagnosis. The YAML file is provided with the path to the existing packages. A diagnosis engine is used in the model-based diagnosis system which uses the correct behavior and the current observations for calculating the diagnosis. The diagnosis engine generates the set of components that are faulty and the nodes that are working correctly. The system detects whether there is a fault that occurred in the system based on the observation made and the correct behavior data. The deviations in the observations are analyzed, which is then utilized for calculating the faulty components. The controller for the diagnosis engine is known as diagnosis_engine_controller. This is a client and the server is written in java which connects the client using a Transport Control Protocol(TCP).

The diagnosis repair engine is a planner based repair action server. This is written using Planning Domain Definition Language(PDDL). This planner makes the plans for various actions using problem file and domain knowledge. The repair engine gets the various diagnosis and observations from the diagnosis engine and observers respectively. This is used by the planner as the problem file. Based on the observations made by the planner, the various actions are taken by the action server. The planning based repair engine used is diagnosis_repair. This is also written in java. The client is connected using TCP.

Finally, the diagnosis board is the hardware component present in this ROS based system. This package consists of various hardware components such as board controller, board simulator, and virtual board. This module considers various hardware diagnoses that are related to the hardware sensors of the robot system. The board is connected to the outside using an Ethernet connection. The controller of the board is known as the diagnosis_board_controller. Client communication is made using TCP. the topics being published are /board measurements which give the values such as id, status(on/off), measurements on individual channels.

All these modules comprise the model-based diagnosis system. Various diagnosis messages are used by the diagnosis system. The messages can be observation, diagnosis, repair, and model server. The various diagnosis launch files are used for launching various modules of the ROS-based diagnosis system.

IV. Web application for robot operation monitoring

A. Observation and diagnosis messages

The model-based diagnosis system consists of sets of observers, diagnosis engine, rule engine. The messages can be observation messages and diagnosis messages.

Observers are used for getting certain information from the

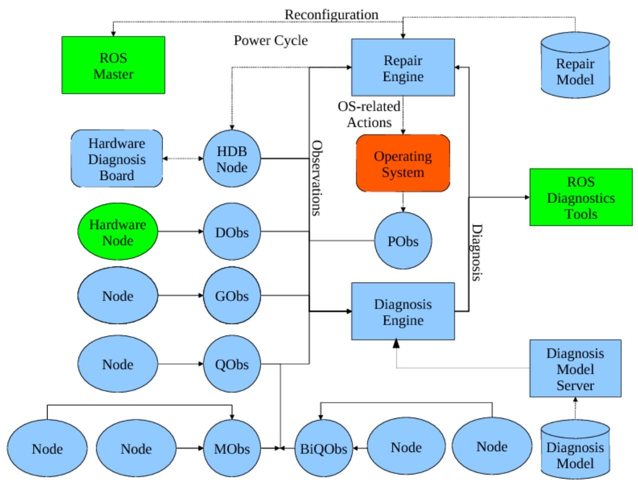


Fig. 1. Overview of model-based diagnosis and repair system [5]

system. These topics are published to /observers/info. The observations made by the observer's node are processed using the diagnosis engine. The result obtained from the diagnosis engine is published to /diagnosis. For communicating observations and diagnosis results, various messages are used. The messages can be observation messages or diagnosis messages. The observation messages can be observer_info, observation_info, observation. The observer_info message provides the observations made by the given observer. Observation_info message used for getting the observation made by a particular resource provided the type of observer. Various observation_info messages are header, type, resource, observation. The 'header' is used to specify the time of the particular observation made. The 'type' provides information about the kind of observation made. The 'resource' provides the data about the resources observed. This can be a mode name, topic name, or some observation made from the topics being published. The 'observation' provides information about the observations taken. An observation message is used for describing the observations made by the system. The observation messages can be observation_msg, verbose_observation_msg, observation. Observation_msg gives an idea of the message observations taken by the system. Verbose_msg provides the verbose description of the observations taken. This shows the description of the observations taken. Observation is a message which gives the number associated with the observations taken. The numbers less than zero are considered faults. The number that is greater than or equal to zero shows the normal behavior of the system.
The diagnosis messages include diagnosis_set and diagnosis. The diagnosis_set stores the values calculated using the diagnosis engine. The diagnosis_set messages can be header, resource, type, diagnoses. The 'header' provides the time when observations are made, which helps in calculating the diagnosis. The 'type' gives information about how the diagnosis is calculated. The 'diagnoses' message is a set of diagnoses calculated by the system. Diagnosis messages used for specifying the mode assignment for the resources in the system. The diagnosis engine uses this resource nodes for the correct diagnosis. The 'resource' message specifies the name of certain resources. [4]

B. Database using sqlite3

The sqlite3 is a program that allows the user to execute the SQL statement and manually enter against an SQLite database. Sqlite3 can be started by typing "sqlite3 diagnoses.db" in the terminal. This will create an SQLite database(if the database with this name is not present, then a database is created automatically.

C. Web interface using Flask framework

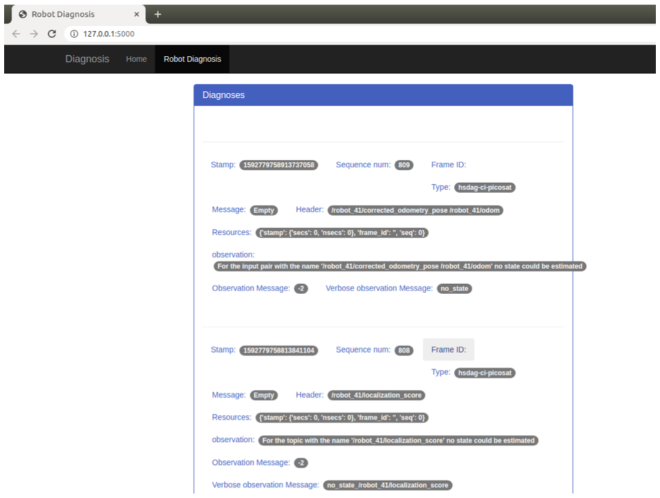
Flask is a web framework written in python. This means it provides tools, libraries, and technologies that allow us to build web applications. The web page has no dependencies on external libraries. It has both merits and demerits, merits being that framework is a light, little dependency to update and look for security bugs and demerits being to work more on increasing the dependency list by adding the plugins. Flask allows users to add some extra functionality to make the web page generation easy and flexible. Flask dependencies are:

- Werkzeug a WSGI utility library
- jinja2 a template engine

Werkzeug is a WSGI (Web Server Gateway Interface), toolkit to implement concepts such as requests, response objects, or other utility functions. It enables the building of a web framework. WSGI is a standard for Python web development. Jinja2 is a popular templating engine for python. When templating engine task is to combine template(HTML syntax) with a certain data source to render some dynamic web pages. Templates are files that contain data that is rendered with desired data to generate the final document. The global variables that are available in jinja2 templates to ease web page implementation by default are flask.config, flask.request, flask.session,flask.g, flask.url_for () function, flask.get_flashed_messages() function. Some features of Flask are namely development server and debugger, Integrated support for unit testing, uses jinja templating, support for secure cookies, extensive documentation, Google App Engine compatibility, etc., Pre-requisites for FLASK installation in python are python 2.7/3 and install of the virtual environment(so that separately files can run with no compatibility issues on environment and versions).

D. Interfacing web server with web application

The web application using the Flask framework is implemented using python. In which observation messages and diagnosis message parameters are displayed on the webpage which updates as launching some files. Fig 2. shows the final web page using the Flask framework for ROS based robot system. The diagnosis messages are stamp, frame_id, msg_type, and msg_diagnosis and the observation messages are header, resources, observation_msg, and verbose_observation_msg. The configuration files of different robots are used to configure the robot and make initial settings of values to the parameters. So, these configuration files are first launched in the ROS environment on which diagnosis is performed by publishing and subscribing to these messages. The diagnosis output is stored in a database that has two columns each for storing diagnosis messages and observation messages by creating a table. We wrote a python program so that it can subscribe to these observations and diagnosis messages. To subscribe to observation messages first the ROS message is converted to the dictionary and is stored sequentially, these are then inserted to the database after this the messages are subscribed. Another python program is written to display these observations and diagnosis messages on the web page using the FLASK framework.



Stamp: 1592779758913737058 Sequence num: 809 Frame ID:
Type: hsdag-ci-picosat
Message: Empty Header: /robot_41/corrected_odometry_pose /robot_41/odom
Resources: {'stamp': {'secs': 0, 'nsecs': 0}, 'frame_id': '', 'seq': 0}
observation: For the input pair with the name '/robot_41/corrected_odometry_pose /robot_41/odom' no state could be estimated
Observation Message: -2 Verbose observation Message: no_state

Stamp: 1592779758813841104 Sequence num: 808 Frame ID:
Type: hsdag-ci-picosat
Message: Empty Header: /robot_41/localization_score
Resources: {'stamp': {'secs': 0, 'nsecs': 0}, 'frame_id': '', 'seq': 0}
observation: For the topic with the name '/robot_41/localization_score' no state could be estimated
Observation Message: -2
Verbose observation Message: no_state_/robot_41/localization_score

Fig. 2. Web page displaying diagnosis results

V. Experimental results

For evaluating the performance of the web page, various analysis has been done. This involves a series of steps to follow for displaying the diagnosis messages on the web page. Initially, we need to run the command "roscore" in the terminal. This helps the ROS nodes to communicate. After this various launch files needed to be launched using the command "roslaunch" command. The launch files launched here are test.launch, diagnosis_anna.launch, observer_cpp_anna.launch. Roslaunch command helps in launching multiple ROS nodes simultaneously. Roslaunch command takes one or more XML configuration files with .launch extensions. This command specifies the parameters to set and nodes to launch. In our case, these launch files were already created by the authors of model-based diagnosis system. After launching these files we found the parameters being published and subscribed. The values being published are mainly observation messages and diagnosis messages. These ROS messages have been stored in the database using sqlite3. The database is created because the ROS messages can be directly displayed on the web page. In order to obtain a proper visualization of the published messages, we chose the sqlite3 database for storing the data. The database will be updated after each ROS message from the diagnosis node or observation node is published. After the database has been created, the python script "rosCommunicator.py" provides the subscriber function for the observer and diagnosis messages. Which is then displayed on the web page using the Flask framework. Finally the python file "run.py" creates a link to the web page in which various diagnosis values have been displayed. The figure 3 represents the overall process described above.

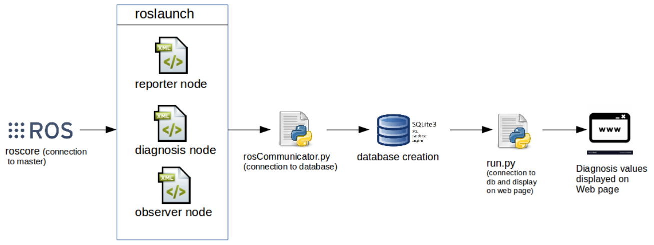

ROS
roscore (connection to master)
roslaunch
reporter node
diagnosis node
observer node
rosCommunicator.py (connection to database)
SQLite3
database creation
run.py (connection to db and display on web page)
www
Diagnosis values displayed on Web page

Fig. 3. Process flow diagram

VI. Conclusion

This work implemented for the software development project course. The paper presents a framework for model-based diagnosis with an in-house web application for robot operation monitoring using the ROS-based system and using the Flask framework. The paper uses the model-based diagnosis system for getting various diagnosis results from the robot which can be used for evaluating the performance of an autonomous robot. This evaluation of the diagnosis values helps the robot to identify the faults and thereby repair the system if faults are detected. The diagnosis values are displayed on the web page using the Flask framework, which updates the values based on the evaluations. Future work can be implementing all the values related to the robot on the web page.

References

[1] Zaman, S Steinbauer, G 2013, "Automated Generation of Diagnosis Models for ROS-based Robot Systems". in International Workshop on Principles of Diagnosis. ., Carcassonne, Frankreich, 2004.

[2] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej and S. Uran, "An integrated model-based diagnosis and repair architecture for ROS-based robot systems," 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, 2013, pp. 482-489, doi: 10.1109/ICRA.2013.6630618.

[3] `http://www.ist.tugraz.at/ais-wiki/model_based_diagnosis`

[4] `http://www.ist.tugraz.at/steinbauer/IST_ROS_Model_Based_Diagnosis_Stack`

[5] `http://wiki.ros.org/tug_ist_model_based_diagnosis`

[6] Alexander Kleiner, Gerald Steinbauer, and Franz Wotawa. Towards Automated Online Diagnosis of Robot Navigation Software. In First International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2008), volume 5325 of Lecture Notes in Computer Science, pages 159–170. Springer, 2008.

[7] F. Zhao, X. Koutsoukos, H.Haussecker, J. Reich, and P. Cheung. Distributed monitoring of hybrid systems: A model-directed approach. International Joint Conf on Artificial Intelligence (IJCAI01), 2001.

[8] Mathias Brandstötter, Michael Hofbaur, Gerald Stein- bauer, and Franz Wotawa. Model-based fault diagnosis and reconfiguration of robot drives. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Diego, CA, USA, 2007.

[9] F. Zhao, X. Koutsoukos, H.Haussecker, J. Reich, and P. Cheung. Distributed monitoring of hybrid systems: A model-directed approach. International Joint Conf on Artificial Intelligence (IJCAI01), 2001.

[10] Eliahu Khalastchi, Meir Kalech, and Lior Rokach. Sensor fault detection and diagnosis for autonomous systems. In The 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS2013), 2013.