A Robotic Control System for Fault Tolerance and Safety using Human Robot Interaction

Luis J. Figueroa Rivera
Department of Electrical & Computer Engineering
Florida Polytechnic University
Lakeland, USA
aramos@floridapoly.edu

Ariel Y. Ramos Ruiz
Department of Electrical & Computer Engineering
Florida Polytechnic University
Lakeland, USA
lfigueroa@floridapoly.edu

Balasubramaniyan Chandrasekaran Ph.D.
Department of Electrical & Computer Engineering
Florida Polytechnic University
Lakeland, USA
bchandrasekaran@floridapoly.edu

***Abstract*— In Robotics, fault tolerant control is an essential aspect for the robot to be fully autonomous without running into errors. One error could lead to another thereby quickly cascading out of control causing the system to behave anomaly. Fault tolerant and self-healing robotic control systems help detect the errors and based on the type of the error the robot automatically can self-heal or self-correct itself. The current paper is a work in progress and provides an outline of such a robotic systems mentioned in literature. A Robotic Control System using human in the loop is proposed to facilitate self-adjusting capability within the robot. Such as system will also help with the safety of the robot and the environment where it is deployed.**

Keywords*— *Human Robot Interaction, Fault Tolerant, Self healing, Graceful degradation, Error Detection, Robotic Control System

I. INTRODUCTION

Fault-Tolerant is vital within a robot system. The term fault tolerant consists of creating programs that are free of faults. Many devices are riddled with bugs, errors, and problems that could degrade the performance of the device, which the systems fall into cascade errors, which one problem falls into another one, which at the end could lead the device inoperable. Such importance of such a system involves on the extended continuity or maintaining functionality in any event of a system failure. A failsafe design will help for the component if it falls into a failure; another component could cover for the malfunctioned component [1]. Reliability is an essential factor to fault tolerant is the prospect that the system will maintain operationally for the duration of the mission [1] [2]. Many errors come from dynamic operation conditions; thus automatic failure detection and self-healing is required at runtime. To achieve such capabilities, the system must identify the causes of the failure to execute a recovery policy. For this instance, an adaptive layered style robotic system is more relevant. The system's components are explicitly layered, meaning each component are layered different in hierarchy form which implies that components on the given layer invoke services at the layer below [3][4] (and are prohibited from invoking services of the component from the layers above). This type of system help in many ways such as:

- Corrects the operation of the layer below (fault tolerance)
- Modifies the layer below (recovery planning)
- Append new capabilities to the layer below (repair execution)
- Relocate components to improve its functionality, implement other self-adaptive services.

This adaptive system is designed and implemented regarding components, which control the robot, and meta-level components that implement fault tolerance, monitoring, diagnostic, recovery planning, repair execution, and a knowledge base [3][4].

This paper is organized as follows. Section II introduces the ROS software framework. A few of the architectures and meta-level components mentioned in literature are covered in Section III. Section IV introduces the proposed robotic system architecture. As the research progresses more components and elements will be added as needed. Section V discusses the conclusion and the possible progress of the research work followed by reference section.

II. BACKGROUND

A. Robot Operating System

The main software framework ROS [5] will be implemented which works with single or multi-root system development. It provides abstraction at the hardware level, low-level device control, provides commonly used functionality, message passing, and manages packages. It uses a particular process called nodes, which performs all the computations. These nodes combine to form a graph and communicate with each other [5]. A robot control system will usually manage many nodes. An example, one node will be comprised of the motor; another node could be used for a sensor, an additional node for the robot camera, and so on. This type of control system brings benefits because the nodes are treated individually, meaning that if one node fails, it will

978-1-7281-0554-3/19/$31.00©2019 IEEE

not affect the other nodes, which implement the fault-tolerance.

A Robotics System Toolbox will be implemented using Matlab/Simulink [6] in conjunction with the Robot Operating System (ROS). It provides algorithms and hardware connectivity for developing autonomous robotic applications, which includes path planning, obstacle avoidance, and state estimation [5][6]. Also, it generates ROS nodes from a Simulink model and automatically deploys it to ROS network.

III. Design and Architecture

In this section, the adaptive layered robotic system is described. The basis used is the concept of meta-layer components [7], which offer a better adaptation and execution for the components, In Fig 1, shows the simple approach of fault-tolerant consists of three significant factors. The recovery, diagnosis, and detection. The approach will consist of four basic blocks presented in Fig 2. The ROS middleware and the operating system provides the necessary fundamentals such as hardware abstraction and execution layer for applying the fault tolerance and self-healing capabilities [5]. The robot system manages all of the robots operations. The last block, which is the internal structure and the most crucial block, RoSHA (Robot Self-Healing Architecture) [3]. Each block is structuralized individually, meaning that the block will not affect each other. Inside the RoSHA, blocks consists of the fault tolerant/self-healing fundamentals or the meta-layer components: monitoring, diagnostic, recovery manager and repair executer. The monitoring component collects information from the components and the operating system such as CPU, memory usage, etc. Diagnostic components collect information by identifying failures throughout the system. The recovery manager selects the appropriate recovery policy depending on the type of failure. Lastly, the repair executor conducts the necessary repair actions depending on the failure and the selected policy [3].

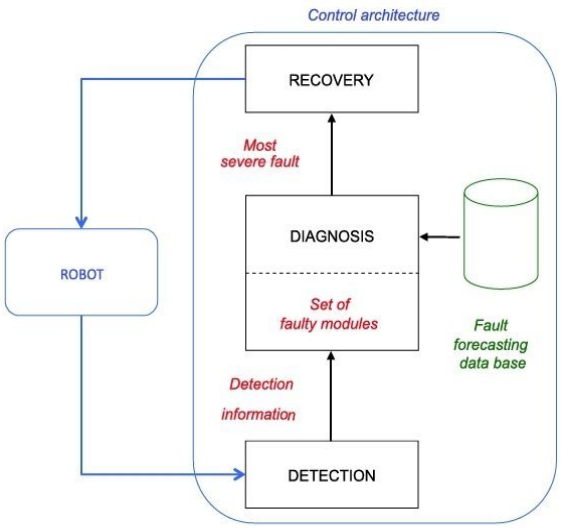

Fig. 1. Fault-Tolerant Diagram [4]

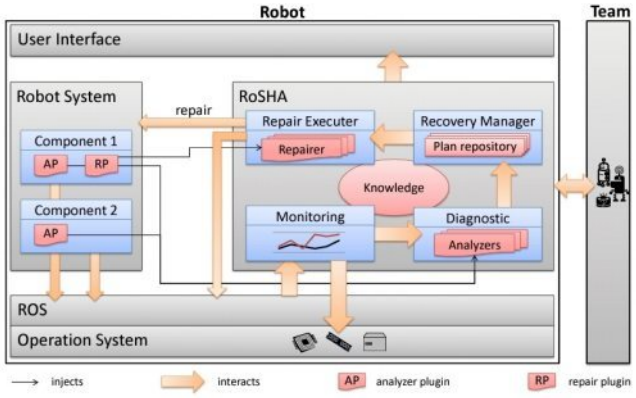

Fig. 2. Robot system Architecture Diagram [3]

In Fig 3 a summary of the different failure conditions to consider for fault tolerance. Also, the architectural principal of the meta-layer components and how each layer acts upon the current state of the robot according to the application-specific policies and goals is mentioned [4].

I.1 Failure Severity

The aim in this section is to determine the potential severity of any of the detected failure throughout the system. It is essential on determining if the robot could accomplish any given mission the human-operator has given it. This affects the efficiency and the accuracy the robot can achieve its's goals. Time and precision is a significant factor such as if the system cannot determine the severity of the situation on-time or the robot fails of determining the severity of the failure, the robot will not necessarily finish its objective. Thus, four different categories are defined for the implementation of the failure severity [4] in Fig 3.

- **Weak** the robot can continue its objective by making minor adaptions to its system.
- **Medium** the robot can continue its objective but with degraded components.
- **Severe** the robot cannot continue its objective with its autonomous structure, needs help from a human.
- **Fatal** the mission can no longer be done.

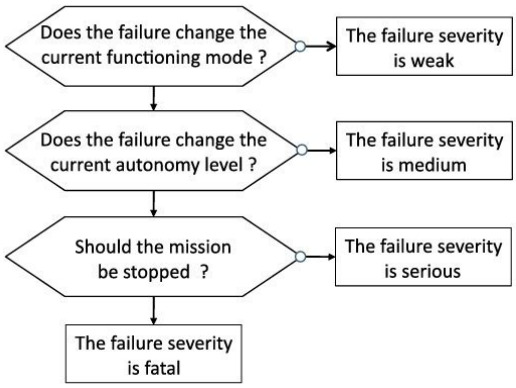

Fig. 3. Determining fault severity [4]

I.2 Meta-Layer Components

An architectural awareness is used which helps each layer within the architecture access to all the sub-elements (components, connects, etc.) of the layer below. The approach uses based on the reactive system [3]. Such system is aware of the environments and acts accordingly to it such as sensing its environment, computes response plans and executes appropriate control actions. The types of meta-layer components are defined below.

I.2.1 Monitoring

The monitoring layer collects the data or states of the components from the robot's system. Two types of monitoring will be used: component monitoring and flow monitoring. Component monitoring collects the state of each component and the flow monitoring, observes the communication between the components. To have a better connection monitoring between the components adaptive monitoring is used. This type of monitoring sends out a heartbeat signal to show the component is still functioning. Thus, it reports any problems to the diagnostic layer [3].

I.2.2 Diagnostic

The diagnostic component or fault tolerant detection and diagnosis scheme [8] estimates the health state of the robot's system and identifies any possible failure causes. In Fig 4, a Bayesian network [3] is used as the diagnostic analysis, which helps determine three mayor key factors: the symptoms layer, root cause layer and the failure layer. The symptoms layer monitors incoming inputs, which determine a possible outcome of a failure that could happen. The root cause layer determines the error caused by symptoms. The Failure layer analyses trigger the failure alert from the layer below. When the diagnostic layer detects the type of failure, it reports the information to the next layer, the Recovery Manager.

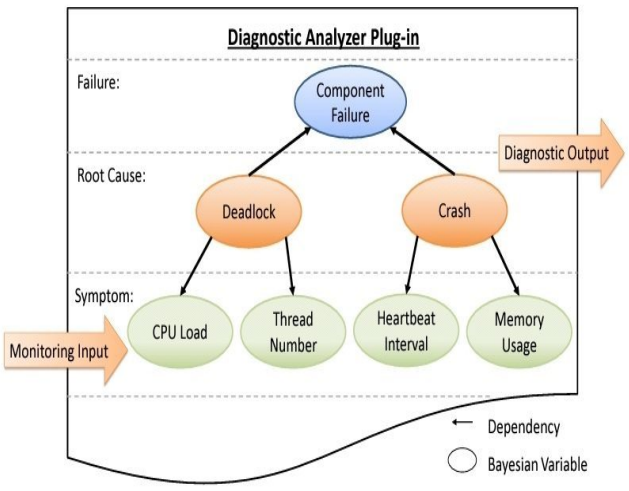

Fig. 4. Diagnostic Analyzer of the Bayesian Network [3]

I.2.3 Recovery Manager

The Recovery Manager collects the necessary data from the diagnostic layer to analyze the failure to determine the appropriate recovery policy to establish and restore the system. The recovery manager uses a reactive principle, which implements recovery in a faster efficient way. Fig 5 shows the reactive principle taken into the effect by using the following policies which the recovery manager will decide what action to take [3][4].

- Reconfiguration (Weak): The system will adjust its parameters accordingly, for which the robot can continue the mission.
- Adaptation (Medium): The system will adapt to its current condition leading to degradation to the robot's system. An adaptation can also occur if an efficient component and substitute for the faulty one.
- Autonomy Adjustment (Serious): The system cannot autonomously adjust its current state, which leads to a human-operator to resolve the issue.
- Definitive Stop (Fatal): The current state of the robot is unable to continue its objective making the robot non-operational.

I.2.4 Repair Executer

From the recovery policies selected by the recovery manager, the repair executer decides the appropriate way to execute the selected policy. Two types of executors can be used for the repair action. Generic repair action is an independent service that repairs different parts of the system and specific repair actions repairs one specific node. A defined property will be included to verify how quickly a system will make the necessary repair by isolating the failure of the system and making the necessary repair [3][4][9].

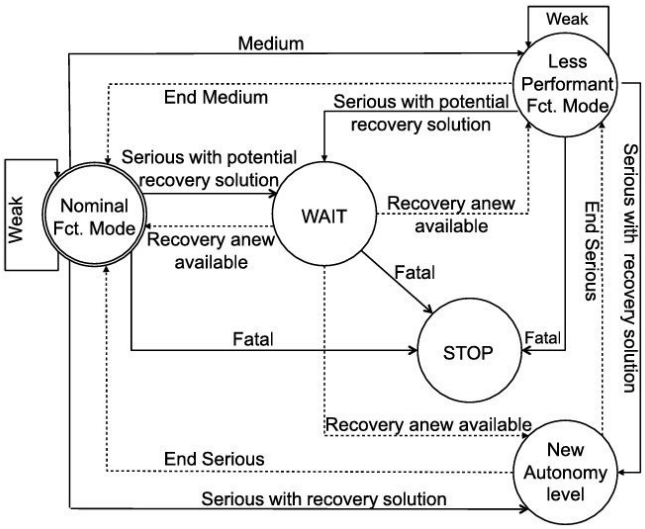

Fig.5. Reactive recovery principle management [4]

IV. PROPOSED ARCHITECTURE

The proposed architecture is a work in progress and will be implemented as future work, consists of the robot control system shown in Fig 6. The robot control system receives sensor inputs from the components which is then that sent to the Sensor Process Controller. The sensor data will be applied to noise filtering and aligned for the sensor fusion process which is the next stage. After the sensor data can be combined to a meaningful decision, it will be input to the motion controller. In Fig 7, the motion control is described in detail. The component will be used for the fault tolerant part of the architecture. The feasibility check will verify the state of the component or the physical constraint the robot may contain. If an error or fault is detected on any of the components, the information is then passed to the diagnosis to verify the type of the fault. If no error is detected, the motion control will by-pass the fault tolerant part and go directly to the outputs. Once the diagnosis has been verified, the recovery system decides the correct policy to implement it. Human-robot interaction is implemented to adjust any parameters the human-operator decides to change or if the user decides to control the robot itself. The rebuild manager then executes the necessary self-healing action is to make to robot operational. The manager if necessary will degrade or substitute components. For example, if the camera is not functioning correctly, an ultra-sound sensor will substitute the camera and make the robot continue its objective. Also, a defined property will be included to verify how quickly a system will make the necessary repair by isolating the failure of the system [3][4][9].

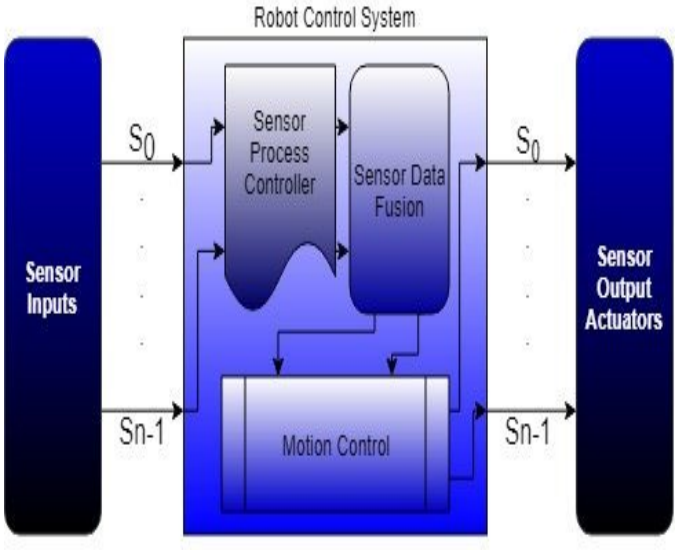

Fig. 6. Robot Control System

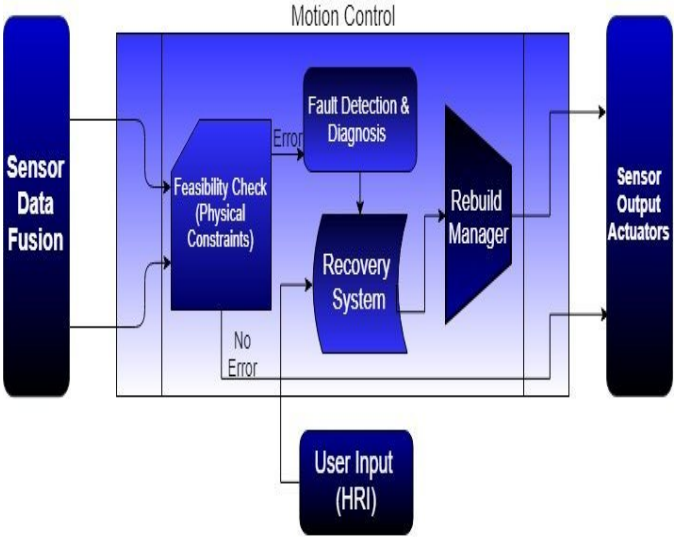

Fig.7. Motion Control (Fault Tolerance)

V. CONCLUSION

The paper outlined a few fault tolerant architectures in literature for robotic systems for self-healing and recovery. The purpose of this paper is to investigate a central and critical issue of faults and recovery within the robotic systems. Many of the faults can run into serious errors and the best way is to fix it with an architecture with human in the loop. An analysis of previous studies has shown that fault tolerance is a crucial importance for the making of a fully autonomous robotic control architecture. This can help with the reliability and costs of the robot [1][10]. The proposed architecture is a work in progress, which will help implement an approach on fault tolerance and self-healing that consists of four main principles: monitoring, diagnosis, recovery and rebuild [3][4][9]. Enhancements will be made to the model to make it more efficient and smarter. Some improvements such as a failure in learning methods will help the robot learn from its errors to make the self-healing much faster and efficient. Many implementations will be added to the proposed architecture. ROS [5] will be the focus of the robot since this will be the operating system. A toolbox from the Matlab/Simulink [6] that utilizes the robotic system will be used as another major focal point for the architecture and Gazebo [11] for the interfacing between the robotic system and the component sensors. This architecture is yet to be fully developed and ready to begin test. The robot will be tested with different scenarios and recorded to see how the robot reacts to each scenario, how it recovers accordingly and to develop estimation techniques to evaluate the reliability of the fault tolerant in any given environment.

REFERENCES

[1] V. P. Nelson, "Fault-tolerant computing: fundamental concepts." IEEE Computer, Vol.23, No.7, pp. 19-25, July 1990.

[2] A. K. Somani, N. H. Vaidya, "Understanding Fault Tolerance and Reliability", IEEE Computer, pp. 45-46, April 1997.

[3] D. Kirchner, S. Niemczyk, and K. Geihs, “RoSHA: A Multi-robot Self-healing Architecture,” *RoboCup 2013: Robot World Cup XVII Lecture Notes in Computer Science*, pp. 304–315, 2014.

[4] 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.

[5] “Powering the world's robots,” *ROS.org*. [Online]. Available: http://www.ros.org/. [Accessed: 10-Nov-2018]

[6] “Robotics System Toolbox,” *Reconstructing an Image from Projection Data - MATLAB & Simulink Example*. [Online]. Available: https://www.mathworks.com/products/robotics.html. [Accessed: 10-Nov-2018]

[7] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus, “Architecture-driven self-adaptation and self-management in robotics systems,” *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2009.

[8] M. A. Kamel, X. Yu, and Y. Zhang, “Fault-Tolerant Cooperative Control Design of Multiple Wheeled Mobile Robots,” *IEEE Transactions on Control Systems Technology*, vol. 26, no. 2, pp. 756–764, Mar. 2018.

[9] K. Becker, S. Voss, and B. Schätz, “Formal analysis of feature degradation in fault-tolerant automotive systems,” *Science of Computer Programming*, vol. 154, pp. 89–133, 2018.

[10] E. Hsu, J. Zhuo, and H. Mirand, *The Intellectual Excitement of Computer Science*, 2003. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/fault-tolerant-computing/index.html. [Accessed: 10-Nov-2018]

[11] Osrf, “Why Gazebo?,” *gazebo*. [Online]. Available: http://gazebosim.org/. [Accessed: 10-Nov-2018]