PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL

FACULTY OF INFORMATICS

GRADUATE PROGRAM IN COMPUTER SCIENCE

Fault Supervision for Multi Robotics Systems

Felipe de Fraga Roman

Research plan presented as partial requirement for
obtaining the degree of Master in Computer Science
at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Dr. Alexandre de Morais Amory

Porto Alegre

2013

INDEX

Image Index

Table Index

LIST OF ABBREVIATIONS AND ACRONYMS

API – Application Protocol Interface

DCIM - Data Center Infrastructure Management

HTTP - Hypertext Transfer Protocol

MRS - Multiple Robots Systems

P2P – Peer to Peer

PUCRS – Pontifical Catholic University of Rio Grande do Sul

ROS - Robot Operating System

SRS - Single-Robot Systems

TCP/IP - Transmission Control Protocol and Internet Protocol

XML - Extensible Markup Language

1 Introduction

Robotics started at the beginning of XX century with the needs to improve the productivity and quality of manufacture products. Nowadays robotics becomes more common and people start to use more robotics to help and accomplish a variety of tasks such as robot arm, automated guided vehicles, unmanned aerial vehicles, humanoid robots and others. Robotics is typically used to execute dangerous tasks, unhealthy tasks, places where it is not possible to have human control, remote areas where is too hard or too expansive to send a human, or even for tasks that demands a too big workload that a human is not able to accomplish.

Basically there are two different kinds of robots: stationary robots and mobile robots. Stationary robots are simpler than mobile because they are fixed at some controlled environment. It's common to use the stationary robots on industry to automate repetitive tasks. Also, this kind of robot is built for a very specific application. Typical applications of stationary industrial robots include casting, painting, welding, assembly, materials handling, product inspection, and testing. All these tasks can be performed with more accuracy and speed compared to humans.

Mobile robotics, on the other hand, is the research area that handles the control of autonomous vehicle or semi-autonomous vehicle *[GDU2010]* and *[RSI2004]*. Currently, there are few commercial applications of mobile service robots: goods transportation, surveillance, inspection, cleaning or household robots, lawn mowing, pool cleaning are just some examples. Robotics has been evolving fast in terms of new functionalities and becoming affordable, increasing its use in several aspects of society *[PAR2010]*. This fact increased the development rate of new and more complex robotic applications *[PAR2010]*, which require more complex software stack *[ABA2008]*. Despite these improvements, autonomous mobile robots have not yet made much impact upon industrial and domestic applications, mainly due to the lack of dependability, robustness, reliability and flexibility in real environments. This requires more research to enable the design of more efficient and robust robotic applications.

One cost-effective way to provide effectiveness and robustness to robotic system is to use multi-robots instead of a single robot. Multi-robot systems (MRS) have some advantages over single-robots systems, these advantages include increased of speed for task completion through parallelism and also can increased of robustness and reliability. MRS can implement fault-tolerant systems. For instance, when one member of the team fails, another can take over his work and continue that the task. An MRS of cheaper and simpler robots can typically provide more reliability than a more expensive and complex single robot *[LEP2008]*. On the other hand, MRS also present more complex challenges compared to single robot system. For instance, MRS are more complex to manage and coordinate the collective system, require increased communication capabilities in order to coordinate all robots, and they are more complex to determine its global state and to debug the system due to its distributed nature.

MRS can be classified as homogeneous or heterogeneous *[SVE2005]*. Homogeneous MRS means that all members of the team have the same specification (hardware and software configuration). Heterogeneous MRS can have different kind of robots in the same team. Homogeneous MRS is easier to replace a faulty robot. On other hand, the advantage of heterogeneous MRS is to support different kind of specialized and simpler robots, compared to a robot that does several different tasks.

Robotic systems can also be classified according to its autonomy level, i.e. its ability to decide how to accomplish a task based on its perception of the environment *[RHB2007]*. There are robots with no autonomy at all, called tele-operated, and semi-autonomous robot (fully autonomous robots are currently not feasible for reasonably complex applications). A robot with some level of autonomy can be called an agent. Multi-agents systems are commonly used to implement MRS with autonomy.

1.1 Motivations and Goals

Mobile robotics will become commonplace in the society if it can be cost-effective and dependable. Currently the cost-effectiveness of robotics is evolving since computers and electronics are more accessible. On the other hand, current single mobile robots lack effectiveness and dependability. MRS are naturally more robust than single robots due its intrinsic redundancy, but it increases the software complexity due to its distributed nature. The *goal of this work* is to provide means to easily monitor faults at a team of heterogeneous robotic agents. The detection and isolation the defective agent is a first step toward an adaptive MRS which can execute the desired task even in the presence of faults.

With more dependable robotic systems, more applications can be created to serve the society.

1.2 Organization

The Section 2 presents the theoretical background necessary to understand this work, such as, the autonomous agents concepts, dependability concepts, and multiple robotic systems. Section 3 describes the state of the art in terms of individual robots fault detection and MRS fault detection. Section 4 specifies the research proposal, its activities, and schedule.

2 Theoretical Background

This section presents a theoretical background of the main concepts used in this research plan.

2.1 Autonomous Agents

Functional programs or traditional software work basically receiving an input, process data and produces some output based on the received input *[RHB2007]*. However there are other kinds of programs that do not work on this traditional approach. This different kind of software maintain an ongoing interaction with their environment, they do not compute some function based on the input and return an output. Some example of these programs includes computer operational systems, process control systems and others. Even more complex software that these two previously approaches are the systems called agents system, an agent is a reactive system that contains autonomy in order to take actions determined by himself to accomplish their goals. These different systems are called agents because these systems are active, they are able to figure out one plan to actively pursue their goals. *[RHB2007]*.

2.1.1 Characteristics of Agents

Agents are systems situated in some environment. Some typical examples are the system stock exchange agents, these systems are developed to observe the stock market and, based on this information, take actions. The agent has the capability to percept its environment through its sensors and it is able to cause some effects on the environment via its actuators. See the *Image 1 - Agent interaction with the environment [RHB2007]*.

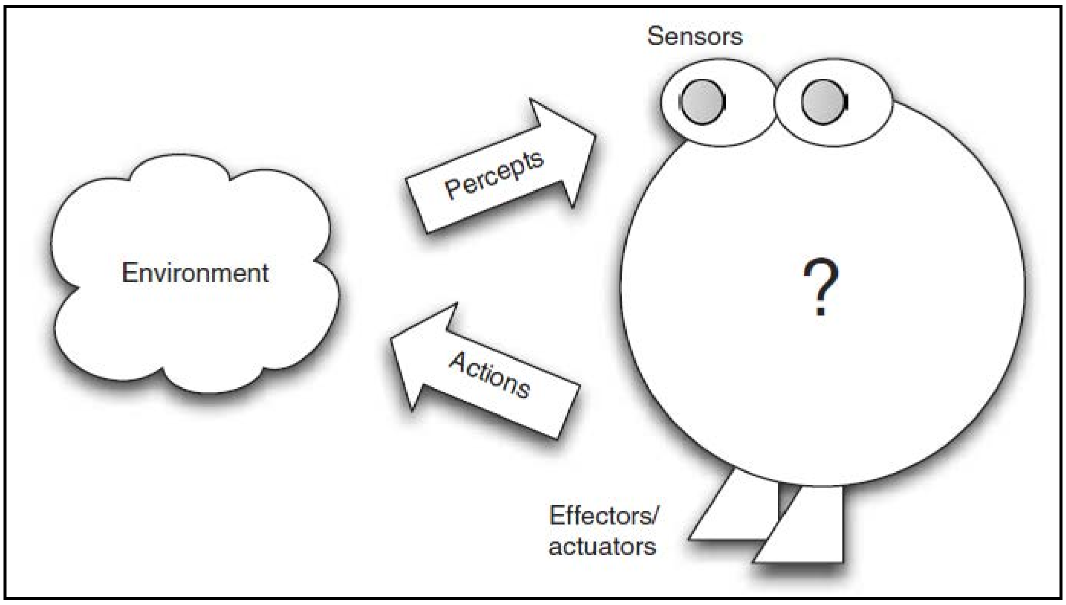

Image 1 - Agent interaction with the environment *[RHB2007]*

According to *[RHB2007]* the environment occupied by an agent could be either physical or virtual (in case of software/simulation environment). For software agent works for virtual environment and robotics works for physical environment. The agents can take actions that will affect the environment, but they cannot, in general, completely control the environment. For example, a robot built to lawn-mower could stuck on a hole and not be able to finish its work. The real environment is dynamic and cannot be controlled so even the highly tested robots will face some unforeseen situations and fail.

There are some important features expected from agents:

- Autonomy: For agents autonomy means that agents have capacity to operate independently. They are able to figure out and execute a determined plan to achieve their goal.
- Pro-activeness: When an agent has been delegated to do a particular goal, the agent needs be able to act according to his goal-directed behavior.
- Re-activeness: Be responsive to the environment changes.
- Social Ability: Instead of simple exchange of bytes and messages, for agents, social ability means to be able to cooperate and to coordinate efforts in order to achieve their goals.

2.1.2 Multi-Agent Systems

Agents inhabit an environment that others agents occupy and each one of these agents have an impact in this environment. It is possible that one agent has control of only part of its environment, but often there are overlapping between the impacts of different agents into the environment, generating more complex scenarios. The *Image 2 - Typical structure of a multi-agent system [RHB2007]* shows a multi-agent system interacting in the same environment.

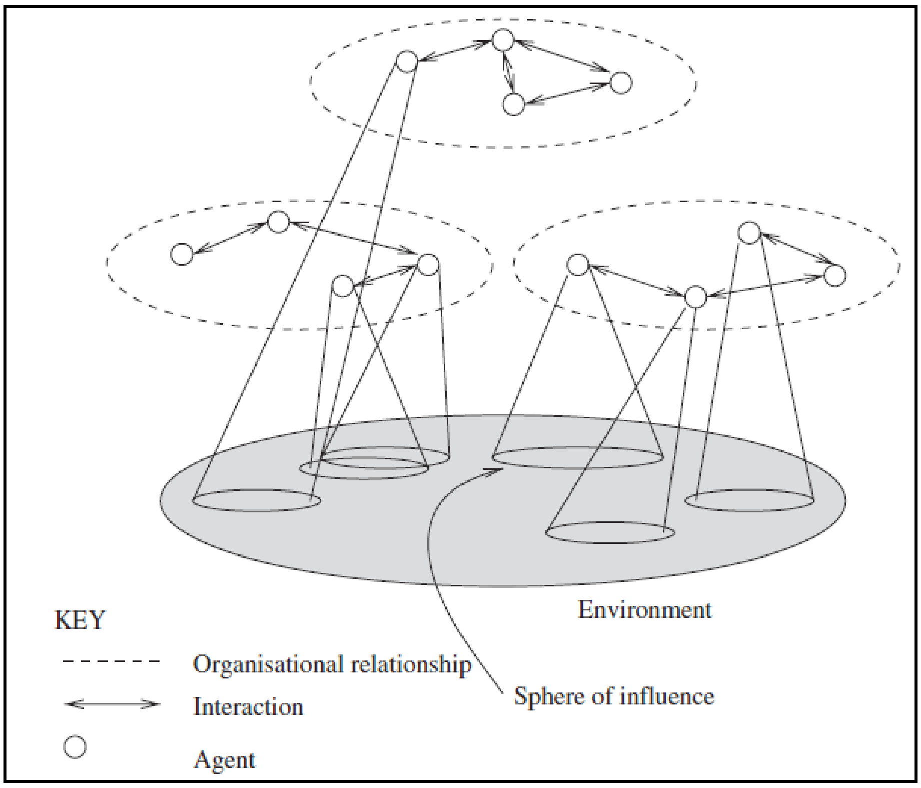

Image 2 - Typical structure of a multi-agent system *[RHB2007]*

2.2 Dependability

The dependability of a computer system is the ability to deliver service that can be trusted *[BLU2004]*. There are three concepts that describe the notion of dependability. The Image 3 demonstrates these concepts.

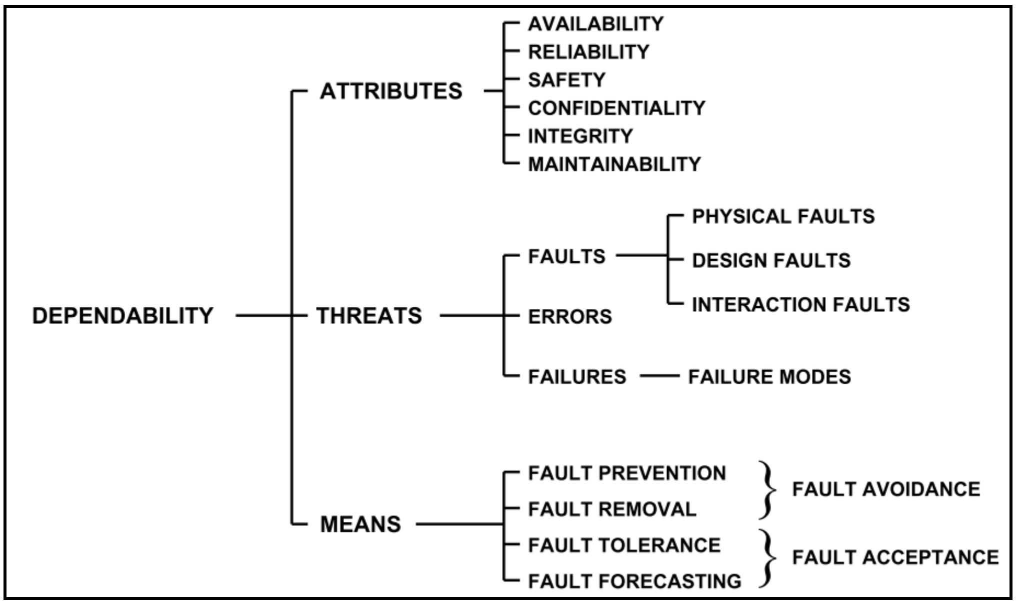

Image 3 - The dependability concepts *[BLU2004]*

2.2.1 Attributes

The dependability attributes could be classified as:

- **Availability:** Be available during a period of time and deliver a correct service during this time.
- **Reliability:** Continuous deliverance of correct service during a period of time.
- **Safety:** Do not cause catastrophic consequences on the users and the environment.
- **Confidentiality:** Does not disclosure unauthorized information.
- **Integrity:** Absence of improper state alterations.
- **Maintainability:** Ability to perform repairs and modifications of the system.

2.2.2 Threats

In this section, we present the taxonomy of threats that may affect a system during its entire life. The life cycle of a system consists of two phases: development and use *[AAV2001]*. The development phase contain all activities from the initial concept presentation, passing by the development itself until the final test phases that shows that the system is ready to deliver the service to the user. During this phase of development, defects or bugs could be introduced by the lack of knowledge of the development team, complexity of the system or even for malicious objectives.

The threats in a system consist in failures, errors and faults. System failures are an event that deviates the delivery of correct service. An error is the part of the system state that may cause a failure. A failure occurs when an error reaches the service interface. A fault is the cause of error. A fault is active when it produces an error, otherwise, it is a dormant fault. A system can fail in different ways. There are three different taxonomies for faults *[BLU2004]* as we show in the Image 4.

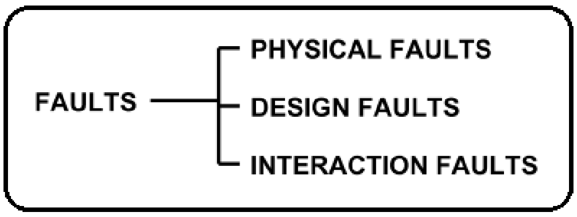

Image 4 - A fault taxonomy *[BLU2004]*

2.2.2.1 Physical Faults

Physical faults are faults due to adverse physical phenomena. For example, a hardware sensor that does not work as expected, returning a non-valid value. A common way to detect this kind of problems is comparing the output of two independent identical units, like a sensor.

2.2.2.2 Design Faults

Design faults are faults unintentionally caused by man during the development of the system. This kind of faults could be either hardware or software faults. Redundant elements are a common way to detect and avoid this kind of faults.

2.2.2.3 Interaction Faults

Interaction faults are faults resulting from the interaction with other systems or users. There is a distinction between accidental faults and malicious interaction faults. An operator mistake is an example of an accidental fault and an intentional attack is a example of malicious fault.

2.2.3 Means

For these three categories of faults mentioned before there are different ways to prevent these faults. These approaches to prevent the faults are called *means* in this diagram below:

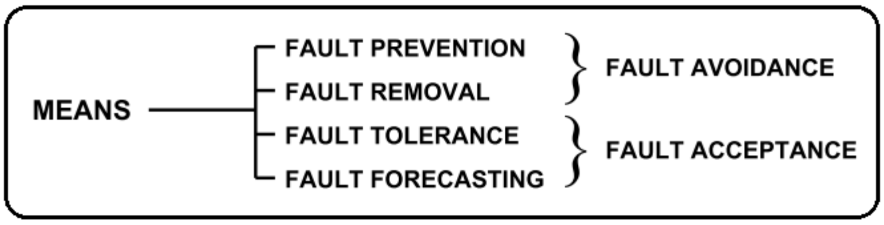

Image 5 - Means - Fault remove techniques *[BLU2004]*

2.2.3.1 Fault Prevention

It is a way to prevent the occurrence or introduction of a fault. Fault prevention can be considered as a fault avoidance system.

2.2.3.2 Fault Removal

It is a way to reduce the number or to reduce the severity of a fault. Fault removal can be considered as a fault avoidance system. Both Fault Prevention and Removal are the attempt to develop a system without faults.

2.2.3.3 Fault Tolerance

It is a way to continue delivering the correct service even when a fault occurs. Fault Tolerance implements the concept of fault acceptance, which attempts to reduce the consequence of a fault. The main difference between fault tolerance and maintenance is that maintenance requires the participation of an external agent and fault tolerance not. This work focuses on fault tolerance mechanisms.

2.2.3.4 Fault Forecasting

Is a way to estimate the future incidence or the consequences of faults. Fault forecasting also implements the same concept of fault acceptance, i.e., an attempt to reduce or estimate the consequence of a fault.

The development of a dependable computing system usually combines different techniques. This work is focused on the Fault Tolerance technique, knowing that fault is almost inevitably. Fault tolerance concepts through the redundancy of multiple robotics or redundant sensors is a good approach to keep the system working as expected, even after faults occur.

2.2.4 Fault Tolerance

Fault tolerance mechanisms typically consist of an error detection and error recovering mechanisms *[LUS2004]*, as illustrated in *Image 6 – Fault Tolerance Techniques [LUS2004]*.

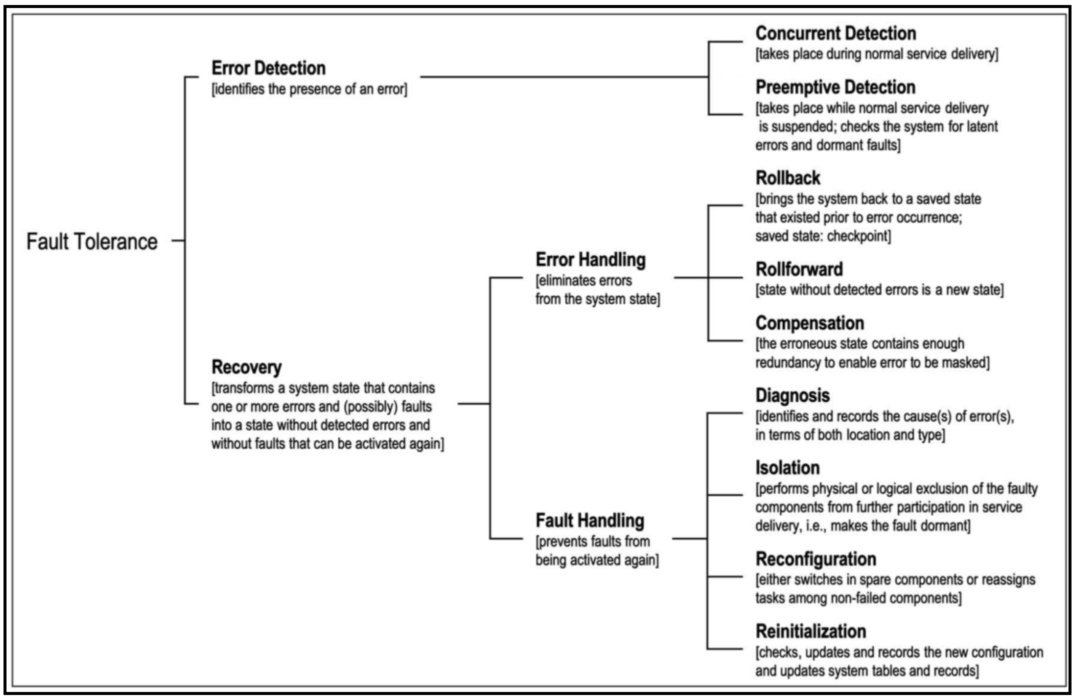
Fault Tolerance
Error Detection
[identifies the presence of an error]
Concurrent Detection
[takes place during normal service delivery]
Preemptive Detection
[takes place while normal service delivery is suspended; checks the system for latent errors and dormant faults]
Recovery
[transforms a system state that contains one or more errors and (possibly) faults into a state without detected errors and without faults that can be activated again]
Error Handling
[eliminates errors from the system state]
Rollback
[brings the system back to a saved state that existed prior to error occurrence; saved state: checkpoint]
Rollforward
[state without detected errors is a new state]
Compensation
[the erroneous state contains enough redundancy to enable error to be masked]
Fault Handling
[prevents faults from being activated again]
Diagnosis
[identifies and records the cause(s) of error(s), in terms of both location and type]
Isolation
[performs physical or logical exclusion of the faulty components from further participation in service delivery, i.e., makes the fault dormant]
Reconfiguration
[either switches in spare components or reassigns tasks among non-failed components]
Reinitialization
[checks, updates and records the new configuration and updates system tables and records]

Image 6 – Fault Tolerance Techniques *[AAV2004]*

2.2.4.1 Error Detection

Error detection originates from an error signal from the system. There are two classes of error detection:

1. Concurrent Error Detection: the error detection works during the same time of the service delivery
2. Preemptive Error Detection: check for error while the service delivery is suspended. Also check for dormant faults.

In this work the focus is on the *concurrent error detection* system that enables the service delivery and fault tolerance at the same time.

2.2.4.2 Error Recovery

Recovery *[BLU2004]* is the process that transforms a system from a state that contains faults and errors to a state that can be activate again without presence of any error or fault. Error recovery eliminates errors in three forms:

- Rollback: Return the system to a previous state where the system can be activated again. The previous saved state is called a checkpoint or safe point. Rollback is the most popular approach to recovery a system, however it is time and resource consuming.
- Rollforward: Put the system in a state where there are no errors or faults. This is a new state not previously recorded. Restart the system is a possible solution for this approach. Note that rollback and rollforward are not mutually exclusive. Usually rollback is the first attempted and then rollforward is a second option.
- Error Compensation: The erroneous state contains enough redundancy to handle the fault situation and enable error elimination. A common approach for error compensation is the fault masking. This approach requires three or more identical or similar components to be used implementing a vote system where the majority is chosen.

These three techniques eliminate errors from the system state. Rollback and rollforward are invoked on demand. Compensation can be applied either on demand or systematically, at pre-scheduled events, independently of the presence of errors.

2.2.4.3 Fault handling

Summon *[ROG2006]*, Fault handling is a technique that prevent fault from being activated again. There are four techniques of fault handling as explained below:

- Diagnosis: Identifies the root cause of error in terms of location and type.
- Isolation: Perform exclusion of the faulty components from further participation in service delivery. The exclusion could be both logical and physical. For physical exclusion the fault component must have a spare component for take over the tasks.
- Reconfiguration: Set up a new configuration avoiding failed components (when it is possible).
- Reinitialization: Checks, updates and records the new configuration and updates system tables and records.

2.3 Multiple Robots Systems (MRS)

A Multiple Robots Systems (or, equivalently Collective Robotic Systems) applies the concept of multi-agent system for robotics. Some of the advantages of the use of MRS over Single-Robot Systems (SRS) are the increased speed of task completion through parallelism, improved solutions for tasks that are inherently distributed in space, time, or functionality, cheaper solutions for complex applications that can be addressed with multiple specialized robots, instead of use of one unique all-capable robot, the increased of robustness and reliability through redundancy *[LEP2008]*. But these advantages do not come for free. For instance, determining how to manage the whole system usually is much more complex than a SRS. The lack of centralized control is one of the reason why the increase of complexity of MRS *[VGO2004]*. Also, MRS requires increased communication to coordinate all the robots in the system. Increasing the number of robots can lead to higher levels of interference between themselves (depends on the used communication device and protocol). Additionally, each individual (robot) in the MRS should be able to work even when the whole system state is unknown *[MJM1995]*.

2.4 Dependable Multiple Robotic Systems

Summon *[LEP2012]* defines reliability in robotics as the probability of a determined system delivery the correct service without failure during a period of time. Different measures of reliability can be given in robotics. For example, an individual component, or an individual robot, or even a multiple robotic system can be measured. MRS should avoid as much as possible to have a single point of failure. Instead, the system must be distributed and able to work as a single. Because the large number of individual components/robots, the MRS could be fault tolerant to an uncertain environment. Also, the MRS known as swarm robots can properly handle a single robot failure. According to *[MOH2009],* there is a difference between MRS and swarm. Swarm robots are a new approach to the coordination of multi-robot systems which consist of large numbers of relatively simple robots which takes its inspiration from social insects.

2.4.1 Reliability in Robotics

Robotics is a research area with a vast amount of literature, even though only a limited part of this effort addresses reliability in robotics *[MLL1998].* Also the analysis to explore the reasons of how the robots fail is not very common in the literature *[JCA2003]*. Centralized approaches to online diagnosis MRS do not scale well basically for two different reasons: complexity of the solutions and the need of communicate each individual to a central diagnoser [DAI2007].

Modern robots usually use the same electronic components and devices from computers. Computers use unreliable components, for this reason they improve their reliability using techniques like error control codes, duplication with comparison, triplication with voting, diagnostics to locate failed components, etc. Similar reliability techniques can be applicable for robotics. One of the main reasons why mobile robots fail is because the real environment cannot be completely mapped and it is naturally dynamic.

Because of the dynamic environment, fault tolerant systems for mobile robots have to be able to handle and even learn from the new situation several times. Because of this complex scenario, there are several approaches to implement reliability in robotics. This work will introduce some of these techniques and the next section explains dependability in Multiple Robotic Systems.

2.4.2 Reliability in Multiple Robotics Systems

Multiple Robots Systems (MRS) need to be reliable as a whole *[LEP2012]*. For these reasons there are some questions to be addressed:

• How to detect when robots have failed?

• How to diagnose robots failures?

• How to respond to these failures?

Instead of single-robots systems (SRS) that are designed to be robust as a single, multiple robots systems (MRS) are design to be fault tolerant, it means, continue working even after a fault occurs. MRS are designed to take advantage of the collective to accomplish the work as a team, it means, they need to be able to communicate between them and a healthy robot could take over a task from a robot in a faulty state.

The main reason of multiple robots system is to achieve significant level of reliability through the redundancy or multiple robots. The key motivation is that several robots faults can be overcome by the redundancy system. In order to achieve this level of reliability the whole system must be developed with these faults in mind. Internal and external reasons can drive the MRS to a fault state. A software design defect is an internal reason that could lead a robot to a fault state. On the other hand, an unexpected environment changes driving the robot to a fault state is an example of external problem. Usually problems caused by external reasons are more difficult to handle or avoid than the internals.

Follow there are some of the challenges of achieve reliability in MRS:

- Individual robot failure: The total number of individual components parts in a system is directed related with the probability of a fault occurs *[JCA2005]*. In Carlson and Murphy observed many different causes of failures leading to low reliability of robots operated by humans. This study also showed that custom designed components are less reliable than mass-produced components such as power supply and sensors.
- Local perspective: Each one of the robots maintains only a local perspective and is not able to see the system as a whole. In order to keep the entire system fault tolerant, the system should be distributed and not centralized. It allows the system to be more fault tolerant and also brings scalability to the MRS.
- Interference: The existence of MRS sharing the same physical environment can cause interference and contention. These issues must be addressed to enable MRS application.
- Software errors: As all complex software systems, the MRS software can also contain bugs that raise faults. Because of the complexity these software, defects/bugs could be difficult to detect and to fix.
- Communication failures: In multiple robots systems the communication between the individual robots is a requirement to enable the whole system works as expected. According to *[RCA1993],* all individual robots have to be able to work even when the communication with others are not available.

3 State of the Art

According to *[LEP2012]* there are large possibilities of faults in robotics, such as: robot sensors faults, uncertain environment models, limited power and computation limits.

In order to address these complex faulty scenarios there are some tools developed that intend to help engineers and developers to handle these problem. Robot middlewares are one of these tools developed to abstract part of the complexity of these problems.

Several robot middlewares *[BRG2009]*, *[BRG2010]*, *[MAK2007]* try to address the fault detection problem but only single parts of the problem are addressed. Each one of these middleware monitoring tools starts from scratch. And also most of them are driven by the capabilities of the robotics middleware and not by the robotics field needs. Also robot middlewares are usually developed to work as a single and it makes difficult to observe the system as a whole.

3.1 Individual Robots Fault Detection

According to *[MHA2003]* the most popular method of fault detection in robot systems is comparing the sensors values with a pre-determined range of acceptable values (use of thresholds). Other well-known fault detection method is creating a vote system based on different redundant components *[RCA2003]*. If a determined individual component is in faulty state the result will be different of the others. So this individual component could be ignored and the others values are used instead.

Logging is another fault detection technique where data is collected in advance to be analyzed later (off-line fault detection). During the normal runtime, all necessary data is collected and stored in some device. The disadvantages of this technique are that a huge amount of data could be generated. Usually Logging needs another monitor to check if the device is not full and needs clean-up actions *[LOT2011]*. Logging could be used for SRS or for MRS.

3.2 Multiple Robots Fault Detection

Fault detection systems in MRS *[MEN2010]* have the distribution as a coefficient that increases the complexity of the process. The MRS must be able to cooperate and communicate with each other to achieve satisfactory performance and stability. A networked control system is a requirement to connect all agents through communication networks. Because of this complexity these systems are subject to faults, performance deterioration or even interrupt the operation.

According to *[MEN2010],* several different methods and techniques to deal with these problems can be found in the literature. However, usually these methods are centralized designed, without attending the distributed and decentralized nature. A technique that could be used to monitor MRS is the Distributed Artificial Intelligence (DAI). This methodology is based on the creation of a supervision system agent that is able to communicate direct with other agents in order to perform monitor tasks. Summon et al. *[CHR2009]* states that one of the most important advantages of swarm robotic systems is redundancy. In case one robot breaks down, another robot can take steps to repair the failed robot or take over the failed robot's task. The solution proposed in this paper is creating a completely decentralized algorithm to detect non-operational robots in a swarm robotic system. Each robot flashes by lighting up its on-board light-emitting diodes (LEDs), and neighboring robots are driven to flash in synchrony. Robots that contain error do not flash periodically and can be detected by others. This innovative approach does not use conventional networking communication to perform monitoring tasks what is an advantage compared with other approaches because it does not generate network traffic and it does not depend on the network.

The work [KBL2006] proposes a metric for evaluation the effectiveness of fault-tolerance system. Through the development of metrics to measure fault-tolerance within the context of system performance. The goal of this work is to measure by identifying the influence of fault-tolerance towards overall system performance. The work also focus on capture the effect of intelligence, reasoning, or learning on the effective fault-tolerance of the system.

Only few methods are designed to attend the distributed and decentralized nature of MRS. An appropriate fault tolerant controller that implements fault detection and diagnosis systems is necessary for monitoring MRS.

4 Research Proposal

This chapter presents the research proposal and its goals. It also presents the activities required and schedule to be executed next year.

4.1 Research Problem

According to *[LEP2012]* even MRS designed to be robust will face unexpected faults from a very large range of possibilities. Detecting the sources of faults is the very first step towards a fault tolerant MRS. The large number of robots, the large number of possible faults in each robot, and a dynamic environment make the fault monitoring a complex and mandatory task for MRS with reliability constraints.

4.2 Goals

The *goal* of this work is to propose a fault monitoring tool for MRS. Our proposal is to integrate a traditional infrastructure networking monitoring tool with a robotics middleware. Our *hypothesis* is that by combining two consolidated tools we are able to reduce development cost/time by developing an extension for both tools. In return, the proposed MRS fault monitoring tool will have the network scalability, software stability, and software extensibility.

4.3 Research Questions

Considering the main goal and the hypothesis presented previously, this research project intend to address the following research questions:

Is it possible to adapt an industry standard in IT infrastructure monitoring tool to monitoring and detecting faults in MRS?

How effective this monitoring system will be?

4.4 Techniques and Tools Analyzed

This section compares the two main types of tools used in this research: IT infrastructure monitoring and robotics middleware.

4.4.1 IT Infrastructure Monitoring

The goal of a DCIM is to provide to the administrator/users an overview of the entire datacenter status. DCIM tools allow the administrators to store and analyze data related to datacenter servers *[COL2012]*. There are several DCIM tools consolidated in the market with both commercial and free-software licenses. Some of the solutions are open-source and also support the development of extensions or plugins. These plugins are used to enhance the capability of the monitoring tool. The rest of this section introduces well-known IT infrastructure monitoring tools.

Ganglia *[GAN2013]* is a "scalable distributed monitoring system" focused on clusters and grids. It gives the user a quick and easy-to-read overview of your entire clustered system. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and data storage and visualization. The algorithms were developed to achieve very low overheads per-node and high concurrency.

Spiceworks *[SPI2013]* is becoming one of the industry standard free network/system monitoring tools. This tool uses SNMP protocol since it has low impact on the network communication with monitoring tasks. Pre-defined alerts can be configured to monitoring the system status. The administrator is also able to select each of these alerts and see more detailed information about the node.

Zabbix *[ZAB2013]* is a network monitoring tool which offers user-defined views, zooming, and mapping on its Web-based console. This tool uses MySQL to store historical information, its backend is developed in C and the administrator front-end is developed in PHP. The protocols SNMP, TCP and ICMP are supported by the agents that run in the host capturing and sending information to the server.

Nagios is the industry standard in IT infrastructure monitoring according to *[NAG2013]*. This monitoring system was developed focused on scalability and flexibility. Nagios provides information about mission-critical IT infrastructure, allowing detecting and repairing problems and mitigating future issues. Nagios supports the development of extensions or plugin to enhance the original tool capability according with the needs.

The Nagios plugin is a small piece of software that must be developed following the Nagios plugin specification in order to support Nagios API. These plugins can monitor virtually any kind of equipment/devices. Based on these flexible aspects, the proposal is to create a custom plugin to monitor both software information and also hardware information. Besides the flexibility, Nagios also supports almost all protocols and features supported by the others.

4.4.2 Robotic Middleware

According to *[ELK2012],* robots middleware is a layer between the operating system and software applications, as illustrated in Image 7. It is designed to manage the heterogeneity of the hardware, improve software application quality, simplify software design, reduce development costs, and improve software reusability.

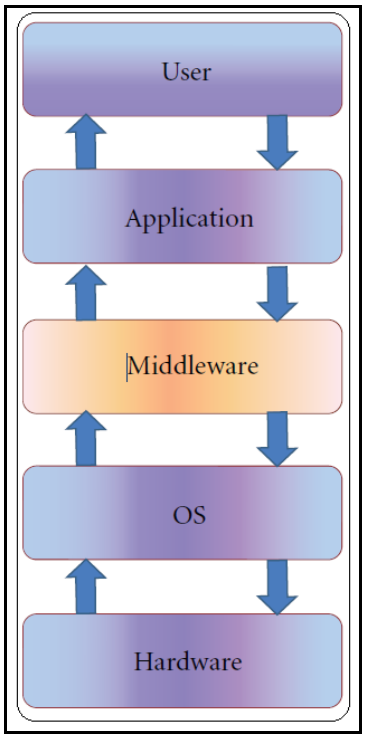

Image 7 - Middleware layer *[ELK2012]*

Modem robots are considered complex distributed systems consisting of a number of integrated hardware (such as the embedded computer and specific robotics sensors) and software modules. The robot's modules cooperate together to achieve their goals *[MOH2008]*. This section describes some of these existent solutions and briefly explains some criteria used to select one.

Miro *[SEN2001]* and *[HUT2002]* is an object-oriented middleware for robots developed by University of Ulm, Germany. Miro is designed and implemented by applying object oriented design and implementation approaches using the common object request broker architecture (CORBA) standard. According to [MIR2013] the core components have been developed under the aid of ACE (Adaptive Communications Environment), an object oriented multi-platform framework for OS-independent interprocess, network and real time communication.

Orca *[AMA2006]* is a middleware framework for developing component-based robotics. It is designed to target applications from single vehicles to distributed sensor networks. The main goal of Orca is to enable software reuse in robotics. According to [ORC2013] it provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks.

According to *[SAH2006]* and *[SKJ2006],* UPnP middleware was developed to utilize the Universal Plug and Play (UPnP) architecture for dynamic robot internal and external software integrations and for ubiquitous robot control. UPnP was developed to offer peer-to-peer network connectivity between PCs wireless devices *[UPN2013]. UPnP uses existent standards protocols, such as TCP/IP, HTTP and XML to connect networked devices and manage them.*

Robot Operating System (ROS) is a middleware that provides a communication layer above the host operating system of a heterogeneous computing node. ROS was designed to meet a specific set of challenges encountered when developing large-scale robots systems. According to *[MQU2009]* the ROS main features are:

• Peer to peer (P2P): the purpose of use p2p communication is to avoid unnecessary traffic in the network

• Tools-based: micro kernel designed instead of monolithic kernel;

• Multi-lingual: developed to be language neutral at the message layer;

•Thin: drivers and algorithm development using standalone libraries that have no dependencies on ROS;

• Free and Open-source: The full source code of ROS is publicly available;

• Collaborative development: in order to build large systems the ROS software system is organized into packages.

ROS has a modular design that allows advanced communication functionalities. These advanced communication features could be extended to communicate with any kind of other tools. Despite ROS middleware provides some tools for monitoring and fault tolerance*[*LOT2011*]*. These tools are useful for development purposes but limited for monitoring a MRS. These tools address only a single part of the overall problem of runtime monitoring because they allow to check the status of one component/module at time.

An IT monitoring tool is an example of one tool that could be extended to communicate with ROS modules and generate useful monitoring information from the MRS as a whole.

4.5 Research Activities List and Schedule

This section discusses the main research activities. Table 1 presents the proposed research schedule:

1. **Study theoretical background**: Deeper study and research on dependability concepts, MRS and dependable MRS. Perform a study and comparison between individual robots fault detection and MRS fault detection techniques. This activity delivers a complete documentation with all theoretical background necessary to continue studying and developing this research.
2. **Study and tests using an IT infrastructure tools**: Create a simulated environment to deploy and configure some of the existent IT infrastructure tools. Test and compare the results to make sure which one is the most appropriate tool to be adapted and used for robotic purposes. Create a report containing the test results and a justification.
3. **Study and tests robotics middleware**: Investigate some of the most well-known robots middleware with the purpose to integrate the sensors information with an external IT monitoring tool API. Create a report containing the test results and a justification.
4. **Implement a plugin to integrate robot middleware with the IT monitoring tool**: Based on the results of the previous investigation, implement an extension or plugin that must be able to collect all robot events and communicate/ integrate it with the IT monitoring tool. The deliverable of this task is the source code of the plugin and a configured environment to demonstrate the results.
5. **Define the robot’s parameters to be monitored**: Study and investigate all PUCRS robots sensors, data, or events that should be collected and monitored. Also identifies techniques such as value interval, thresholds or pre-defined acceptable values to monitor and generate warning/alerts on demand. This

activity releases a table of values contain the following columns: State or Sensor, recommended interval to check, recommended technique to monitor.

6. **Planning and executing the experiment**: Create a detailed plan on how to mount and configure a controlled environment (containing more than one robot - MRS) and define strategies to simulate fault circumstances. The output of this task is a plan containing the steps to reproduce the tests.
7. **Tests and analyses of results**: Execute the tests as planned before for a continuous period of time, for repeated times, and document all the results. Analyze the results and document the conclusions.
8. **Write the "Seminário de Andamento"**: Create a report following the PUCRS instructions and present the results obtained so far. Plan some corrective action if necessary and present the next steps of the work.
9. **Review the state of the art**: Keep looking for related papers and approaches.
10. **Write papers and Master's dissertation**: Write and submit papers for selected conferences. Also write Master's dissertation according to PUCRS specifications. The deliverables of this task are the submitted papers to conferences and the final dissertation containing the complete work.
11. **Present the work**: Make a presentation of all work developed to the board of professors selected to evaluate this work.

Table 1 – Research Activities and Schedule.

Tabela 1 - Research Activities and Schedule

Activities	2014 / 2015												
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan
Study theoretical background	■	■											
Study and tests using an IT infrastructure tools		■	■	■									
Study and tests robotics middleware			■	■	■								
Implement a plugin to integrate robot middleware with the IT monitoring tool				■	■	■							
Define the robot’s parameters to be monitored					■	■	■						
Planning and executing the experiment							■	■					
Tests and analyses the results								■	■				
Write the “Seminário de Andamento”					■	■							
Review the state of the art			■	■	■	■	■	■	■	■	■	■	■
Write papers and Master's dissertation										■	■	■	■
Present the work													■

5 References

[AAV2001] A. Avizienis, J.-C. Laprie, B. Randell et al., Fundamental concepts of dependability. University of Newcastle upon Tyne, Computing Science, 2001.

[AAV2004] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” Dependable and Secure Computing, IEEE Transactions on, vol. 1, no. 1, pp. 11–33, 2004.

[ABA2008] A. Basu, M. Gallien, C. Lesire, and T. Nguyen, “Incremental component-based construction and verification of a robotic system,” ECAI, 2008. [Online].

[AMA2006] A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for Robotics," In International Conference on Intelligent Robots and Systems (IROS), pp. 163-168, Oct. 2006.

[BLU2004] B. Lussier, R. Chatila, F. Ingrand, M.-O. Killijian, and D. Powell, “On fault tolerance and robustness in autonomous systems”, in Proceedings of the 3rdI ARP-IEEE/RASEURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments, 2004.

[BRG2009] Brugali, Davide, and Patrizia Scandurra. "Component-based robotic engineering (part i)[tutorial]." Robotics & Automation Magazine, IEEE 16.4 (2009): 84-96. 2009.

[BRG2010] Brugali, Davide, and Azamat Shakhimardanov. "Component-based robotic engineering (part ii)." Robotics & Automation Magazine, IEEE 17.1 (2010): 100-112. 2010.

[CHR2009] Christensen, Anders Lyhne, Rehan O'Grady, and Marco Dorigo. "From fireflies to fault-tolerant swarms of robots." Evolutionary Computation, IEEE Transactions on 13.4 (2009): 754-766. 2009.

[COL2012] Cole, Dave. "Data center infrastructure management." Data Center Knowledge (2012).

[DAI2007] Daigle, Matthew J., Xenofon D. Koutsoukos, and Gautam Biswas. "Distributed diagnosis in formations of mobile robots." Robotics, IEEE Transactions on 23.2 (2007): 353-369.

[ELK2012] Elkady, Ayssam, and Tarek Sobh. "Robotics middleware: a comprehensive literature survey and attribute-based bibliography." Journal of Robotics 2012.

[GAN2013] Ganglia Website. [Online] Available from: http://ganglia.sourceforge.net/ 2013.

[GDU2010] G. Dudek and M. Jenkin, Computational principles of mobile robotics. Cambridge university press, 2010.

[HUT2002] H. Utz, S. Sablatng, S. Enderle, G. Kraetzschmar, "Miro- Middleware for Mobile Robot Applications," IEEE Transactions on Robotics and Automation, 18(4):493-497, Aug. 2002.

[JCA2003] J. Carlson and R. R. Murphy, “Reliability analysis of mobile robots,” in Robotics and Automation, 2003. Proceedings. ICRA’03. IEEE International Conference on, vol.1. IEEE, pp. 274–281, 2003.

[JCA2005] J. Carlson and R. R. Murphy, “How ugvs physically fail in the field,” Robotics, IEEE Transactions on, vol. 21, no. 3, pp. 423–437, 2005.

[KBL2006] Kannan, Balajee, and Lynne E. Parker. "Fault-tolerance based metrics for evaluating system performance in multi-robot teams." Proceedings of Performance Metrics for Intelligent Systems Workshop. 2006.

[LEP2008] L. E. Parker, “Multiple mobile robot systems,” Springer Handbook of Robotics, pp. 921–941, 2008.

[LEP2012] L. E. Parker, “Reliability and fault tolerance in collective robot systems,” Handbook on Collective Robotics: Fundamentals and Challenges, page To appear. Pan Stanford Publishing, 2012.

[LOT2011] Lotz, Alex, Andreas Steck, and Christian Schlegel. "Runtime monitoring of robotics software components: Increasing robustness of service robotic systems." Advanced Robotics (ICAR), 2011 15th International Conference on. IEEE, 2011.

[MAK2007] Makarenko, Alexei, Alex Brooks, and Tobias Kaupp. "On the benefits of making robotic software frameworks thin." International Conference on Intelligent Robots and Systems. Vol. 2. 2007.

[MEN2010] Mendes, Mário JGC, and J. da Costa. "A multi-agent approach to a networked fault detection system." Control and Fault-Tolerant Systems (SysTol), 2010 Conference on. IEEE, 2010.

[MHA2003] M. Hashimoto, H. Kawashima, and F. Oba, “A multi-model based fault detection and diagnosis of internal sensors for mobile robot,” in Intelligent Robots and Systems, 2003. (IROS2003). Proceedings. 2003 IEEE/RSJ International Conference on, vol. 4. IEEE, pp. 3787–3792, 2003.

[MIR2013] Miro – Middleware for Robots Website. [Online] Available from: http://www.ohloh.net/p/miro-middleware. 2013.

[MJM1995] M. J. Mataric, M. Nilsson, and K. Simsarin, “Cooperative multi-robot box-pushing,” in Intelligent Robots and Systems 95.’Human Robot Interaction and Cooperative Robots’, Proceedings. 1995 IEEE/RSJ International Conference on, vol. 3. IEEE, pp. 556–561, 1995.

[MLL1998] M. L. Leuschen, I. D. Walker, and J. R. Cavallaro, “Robot reliability through fuzzy markov models, ”in Reliability and Maintainability Symposium, 1998. Proceedings., Annual. IEEE, pp. 209–214, 1998.

[MLV1994] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker, “Robotic fault detection and fault tolerance: Asurvey”, Reliability Engineering & System Safety, vol.46, no.2, pp.139–158, 1994.

[MOH2008] Mohamed, Nader, Jameela Al-Jaroodi, and Imad Jawhar. “Middleware for robotics: A survey Robotics, Automation and Mechatronics”, 2008 IEEE Conference on IEEE, 2008.

[MOH2009] Mohan, Yogeswaran, and S. G. Ponnambalam. "An extensive review of research in swarm robotics." Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on. IEEE, 2009.

[MQU2009] 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.

[NAG2013] Nagios, “Nagios - the industry standard in it infrastructure monitoring, 2013. Available from: http://nagios.org,” [Online]. Available: http://nagios.org, 2013.

[ORC2013] Orca: Components for Robotics Website. [Online] Available from http://orca-robotics.sourceforge.net/. 2013.

[PAR2010] P. A. R. Fagundes, “Plataforma de controlo e simulacao robotica,” 2010.

[RCA1993] R. C. Arkin, T. Balch, and E. Nitz, “Communication of behavorial state in multi-agent retrieval tasks,” in Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on. IEEE, pp. 588–594, 1993.

[RCA2003] R. Canham, A. H. Jackson, and A. Tyrrell, “Robot error detection using an artificial immune system,” in Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on. IEEE, pp. 199–207, 2003.

[RHB2007] R. H. Bordini, J. F. H¨ubner, and M. Wooldridge, Programming multi-agent systems in AgentSpeakusingJason. Wiley. com, vol. 8, 2007.

[ROG2006] L. D. Rogério. Adaptability and Fault Tolerance. University of Kent, UK. 2006.

[ROS2013] ROS Website. [Online] Available from: http://www.ros.org, 2013.

[RSI2004] R. Siegwart and I. R. Nourbakhsh, Introduction to Autonomous Mobile Robots. The MIT press, 2004.

[SAH2006] S. Ahn, J. Lee, K. Lim, H. Ko, Y. Kwon, and H. Kim, "Requirements to UPnP for Robot Middleware," in Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Oct. 2006.

[SEN2001] S. Enderle, H. Utz, S. Sablatng, S. Simon, G. Kraetzschmar, and G. Palm, "Miro: Middleware for autonomous mobile robots," IFAC Conference on Telematics Applications in Automation and Robotics, 2001.

[SKJ2006] S. Ahn, K. Lim, J. Lee, H. Ko, Y. Kwon and H. Kim, "UPnP Robot Middleware for Ubiquitous Robot Control," The 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2006), Oct. 2006.

[SPI2013] Spirceworks Website. [Online] Available from: http://www.spiceworks.com/ 2013.

[SVE2005] S. Verret, “Current state of the art in multirobot systems,” Defence Research and Development Canada-Suffield, no. December, 2005.

[UPN2013] UPnP Website. [Online] Available from: http://www.upnp.org 2013.

[VGO2004] V. Goldmanand S. Zilberstein, “Decentralized control of cooperative systems: Categorization and complexity analysis,” J. Artif. Intell. Res.(JAIR), vol. 22, pp. 143–174, 2004.

[ZAB2013] Zabbix Website. [Online] Available from: http://www.zabbix.com/ 2013.