University of Twente

EEMCS / Electrical Engineering
Control Engineering

A Safe-Guarded Multi-Agent Control System for Tripod

Gert Bonestroo

MSc Report

Committee:

Prof. dr. ir. J. van Amerongen
Dr. ir. T.J.A. de Vries
T.S. Tadele, MSc
Dr.ir. J. van Dijk

February 2012

Report nr. 006CE2012
Control Engineering
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

A Safe-Guarded Multi-Agent Control System for Tripod

G. Bonestroo, T.J.A. de Vries, *Member, IEEE*, T.S. Tadele and J. van Amerongen, *Member, IEEE*
Control Engineering, University of Twente, The Netherlands

***Abstract*—This paper presents an evaluation of a practical solution for a complex control problem. This control problem concerns the operation of a parallel manipulator. It is complex in the sense that it entails a set of interdependent sub-problems that require solutions with multi-operation modes and safety features. One of the methods to realize such a control system is by using a Multi-Agent based Control System (MACS) that uses autonomous agents to handle specific control problems and coordinate their output to attain an overall goal [1]. Recently a generalized control solution for mechatronic systems together with a supporting framework was developed by using MACS and pattern based design. This control solution for complex control problems is evaluated. This generalized control solution includes safety patterns and is based on the concept of Multi-Agent systems. It is found that after some adaptation this control solution realizes a good working error handling mechanism.**

I. Introduction

This paper is concerned with the position control of a parallel manipulator, Tripod. Tripod is used at the Control Engineering group of the University of Twente for testing different types of advanced controllers. It is a three degree of freedom setup driven by three linear motors, that can move up and down to position a platform in a cylindrical operating volume (Fig 1).

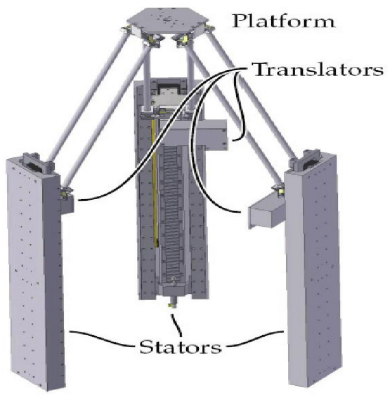


Fig. 1: Tripod

The overall system goal of controlling the Tripod is to move the platform from one point to another in the workspace within a defined accuracy. This can be achieved by controlling each linear motor independently and adding an interaction mechanism between the control systems of each axis. Various mechanisms are needed to generate setpoints, design a controller for each axis and coordinate their activities. So the controller could have unique sub-controllers for each axis that communicate and interact with each other to attain the system goal. In addition, the control system of Tripod should enable multi-operation modes like start-up, homing, normal operation, shutdown, etc. Each operation mode can have different motion trajectories, control system configuration (simple or advanced controller), and control missions (accurate position control or safe-guarded control). Moreover, while performing a certain task, a robot system usually has to cope with various levels of criticality of different errors that can affect the safety of the system. If these errors are not identified and handled correctly, they can bring dangerous situations for human and machine. Handling these errors should also be part of the control system, this will also influence the complexity of the control problem. As a result, controlling Tripod is considered as a complex control problem, because it entails a set of interdependent sub-problems that require solutions with safety features and multi-operation modes.

A commonly used strategy of solving a complex control problem is by decomposing it into partial control problems [2]. This strategy is called the divide-and-conquer approach and consists of three steps:

- Decomposing the overall control problem into a complete set of well-defined partial control problems.
- Solving the partial control problems.
- Integrating the partial solutions into an overall solution

As a result, the solution for a complex control problem is typically a multi-controller system: a set of sub-controllers that is combined into an overall solution. Various interactions between these sub-controllers naturally arise in such systems due to the dependencies between the partial problems to be solved. Thus, a coordinating mechanism is needed to coordinate the activity of each sub-controller such that the system-wide goal is achieved. One general design method that includes coordinating mechanisms is the Multiple Model Approach [2]. Using this approach, the designer has to deal with particular integration aspects, such as deciding when to (in)activate a sub-controller and to combine control actions of several sub-controllers. As an alternative to the above method, a decentralized integration method has been developed, based on the concepts of an agent and multi-agent systems [1]. The result is an open design environment for multi-controller systems, such that individual sub-controllers can be added, modified or removed from the overall multi-controller without redesigning the remaining system [1]. Because each sub-controller of such a system is based on the concept of an agent,

the resulting control system is called a Multi-Agent Control System (MACS).

While different authors have presented the application of MACS in solving various complex problems [3] [4] [5], recent research has extended MACS to deal with safety issues of mechatronic systems [6]. This has resulted in design patterns that can be used to develop a safe-guarded MACS having good performances and a short development time. The focus of this paper is to evaluate MACS including the design pattern, by applying the method to design a safe-guarded controller for the Tripod. Evaluation of this method is done according to the following aspects:

- Timing behavior of the control system: Improper timing will influence the behavior of control system.
- Reusability of the design results: Reusability of the design results will shorten development time.
- Behavior of safety patterns: Safety patterns should work for all error sources that they are designed for.
- Behavior of communication mechanism: Communication with the external environment (which is the real Tripod setup) should be modernized and tested.

II. Background and related works

A. Multi-Agent Control System (MACS)

As already introduced, a multi-controller solution to complex control problems is: a set of sub-controllers that can be combined into an overall solution [6]. When each (sub)controller is capable of autonomous decision making, it is called a controller agent [1]. This means that an agent can decide for itself whether it should undertake some action. When a complex control problem is solved by a pool of controller agents in which each one is responsible for solving a part of the whole problem, it is known as a Multi-Agent Control System (MACS).

Because multiple controller agents are acting on their own particular problems to solve the overall problem, conflicts between individual controller agents may arise, as these partial problems are interdependent. Conflicts between individual controller agents may also arise when outputs of these agents are combined. These conflicts are solved by coordination mechanisms between controller agents. These coordination mechanisms determine when and how actions of the controller agents are applied to the plant. There are at least five main coordination mechanisms used in MACS [6]:

- Fixed priority: Each controller agent in a group is assigned a fixed priority. The sub-agent with the highest priority that wants to become active, becomes active.
- Sequential: Makes controller agents in a group active in succession, for a single round only.
- Cyclic: Makes controller agents in a group active in succession, repeatedly.
- Master-Slave: Master-Slave is a subordination dependency in which the slave agent depends on the master agent. The slave is active only when the master is active.
- Parallel: All controller agents in a group can be concurrently active.

B. OROMACS: implementation framework for Multi-Threaded MACS

OROMACS is an implementation framework for MACS, based on the OROCOS framework. OROCOS is an open source software framework for general robot control that provides a real-time toolkit to develop component-based real-time control applications [7]. Each Orocos component is defined as a TaskContext and executes a certain task on a defined environment or context by using its own thread of control.

In OROCOS, hierarchically structured control programs are not possible because composite components are not supported in OROCOS. OROMACS uses the features of OROCOS to build component-based real-time control applications and makes it possible to create composite components to build a MACS [6]. OROMACS also implemented the coordination objects that are important while designing and implementing MACS.

C. Existing Control System

The original control system of Tripod is a single threaded MACS, running at a personal computer with MS-DOS. To accomplish hardware access, expansion cards with MS-DOS drivers are used for digital and analog input/output signals [8]. A simplified system layout is given in figure 2.

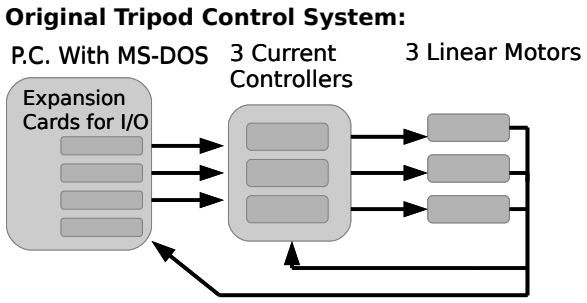


Fig. 2: Simplified system layout of the original Tripod Control System

The existing controllers of Tripod only consist of PID based feedback controllers whose performance can be enhanced with addition of learning feedforward controllers. A learning controller is a controller that uses a learning process to adapt it's behavior during control. This learning process is used in such a way that a desired behavior of the controlled system is obtained [9]. The learning process of such a controller can cause computational load and is not time-critical. Thats why it should preferably run in another (non-real-time) thread than the rest of the control system. This is not possible in the software environment of the original control system, because it does not support multi-threaded applications. The just explained OROMACS framework does support this. The OROMACS framework only runs on a Linux OS. Because there are no Linux drivers for the expansion cards in the P.C., new drivers should be developed or other hardware components should be used to accomplish input/output access.

III. New Control System

A. Introduction

The Control Engineering group at University of Twente is interested in high performance mechatronic systems that implement Distributed Control Systems (DCS) used with different fieldbus technologies to have a modular system design. One of the technologies used in line of this goal is EtherCAT, which enables fast flexible systems to connect different I/O devices. CANopen is used as a high level communication protocol to have a CANopen over EtherCAT (CoE) system.

The research group recently developed a CoE Master driver for OROCOS that can be used to connect a special kind of I/O devices to the computer [10]. These I/O devices are terminals from the Beckhoff Company [11]. Because this CoE Master driver can be used with OROCOS and OROMACS, this driver is used for the Tripod control system. The P.C. of the Tripod Control System can be connected with the EtherCAT I/O devices by means of an Ethernet cable. A simplified system layout of the new Tripod Control System is given in figure 3.

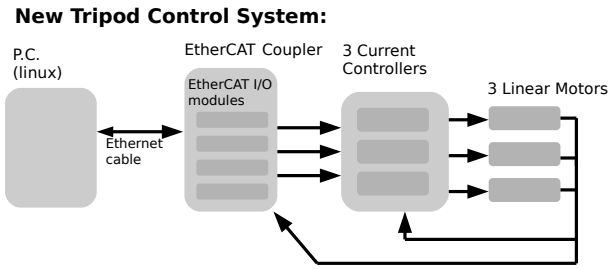


Fig. 3: Simplified system layout of the new Tripod Control System

B. Software requirements

As explained, the Tripod control system should be based on the OROMACS framework, that is based on OROCOS. The Tripod control system also has real time requirements. These requirements can be achieved by using Linux as operating system. There are several definitions of real-time systems. In this thesis hard real-time systems are addressed: those with timing deadlines that must not be missed otherwise the system fails [12].

To achieve real-time behavior of the control system, a Linux kernel is installed that can support the desired deadlines of the real-time tasks (even under worst-case loads). Different real-time architectures are possible. An easy to use architecture is to make the standard Linux kernel preempt-able [13]. This is done by using the PREEMPT RT patch [14]. The PREEMPT RT patch provides several modifications to yield hard real-time support.

Other architectures are for example Real-time Application Interface (RTAI), and Xenomai. These architectures make use of a second kernel that runs separate from the (non real-time) Linux kernel. Real-time tasks are running in this second kernel. The second kernel ensures that the non real-time Linux kernel cannot preempt the operation of it. In contrast with the PREEMPT RT patch, a drawback of this architecture is that non real-time tasks do not have full Linux platform support [13]. Because the PREEMPT RT patch provides full Linux platform support, it is fast and easy to use. For example, there is no extra effort needed to install video drivers and network drivers. That's why this architecture is used in this thesis.

C. System design and Simulation

Goal of this research is to implement and evaluate a MACS for Tripod that is based on a generalized control solution for complex control problems [6] that includes different error handling mechanisms. In order to evaluate different features of the control system described in the introduction, various system configurations were developed. These configurations are described in the next chapters.

A simulation model is used to simulate the Tripod setup. This simulation model is written in the software package 20sim [15], that runs under Windows OS. 20Sim is designed to simulate the behavior of dynamic systems by using sub-models in the form of iconic diagrams, bond graphs, block diagrams and equation models. The Tripod Control System has been developed based on TaskContext components of the OROCOS framework. The connection between the Tripod Control System (running under linux OS) and the simulated Tripod setup (running under Windows OS) is realized by means of an OROCOS-20sim co-simulation tool [16]. This co-simulation tool forms a co-simulation environment where a safe-guarded MACS, running on a Linux OS, can be tested with a 20-sim simulated plant running on a Windows OS.

IV. Configuration 1.a: A simple OROMACS based MACS

A. Modelling

In an OROMACS based MACS, each agent is modularized into two parts: a type and a specification. The type defines its interface to the outside world. The specification defines its implementation. When different agents in a MACS are from the same type and have different specifications, the inside of each agent can be totally different. However, the interface to the outside world is the same.

An OROMACS based MACS consists at least of a main agent. This main agent can only have composite specification and contains all other agents. To test the Tripod Control System before connecting it to the real setup, the controller is first simulated with a co-simulation tool. That's why the main agent has two different specifications: deployment and co-simulation. Depending on its specification, the main agent has different sub-agents with different specifications. In this configuration, the main agent can have the following sub-agents:

- Controller agent: Implements the PD controller for one motor. This controller agent includes a motion generator object to generate the motion profile for one motor.
- Commutator agent: This agent implements a commutation algorithm for a linear motor. The commutation algorithm is implemented by means of a so-called Inverse

Clarke-Park transformation [17]. This transformation is used to transform the control signal from controller to a three-phase control signal that represents the three-phase-current of the motor. The output signal of this agent is dependent on the motor position and the control signal from the controller.

- Co-simulation interface agent: The Co-simulation interface implements an co-simulation interface by means of the OROCOS-20sim co-simulation tool.
- Actuator agent: The actuator agent implements the CoE driver for OROCOS to connect the control system to the real setup. An overview of configuration 1.a with two different specifications is given in figure 4.

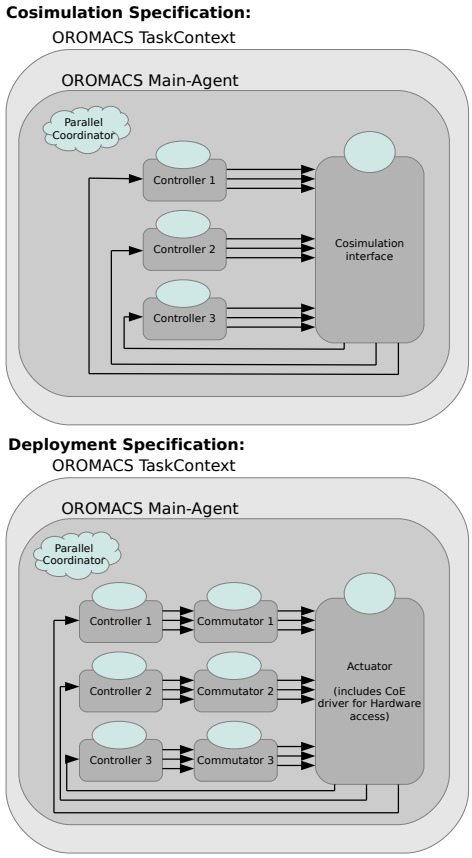


Fig. 4: Configuration 1.a of Tripod Control System, with two different specifications

B. Simulation

Simulation results are given in figure 5. Figure 5 shows a motion profile which is usual for a parallel manipulator like Tripod. It shows that one linear motor of Tripod is tracking it's motion profile within a given accuracy.

C. Testing and Conclusion

Although simulation of the given control system shows a working controller, testing the control system with the real setup shows different behavior. Sometimes the linear motors hit the end stops. Because this behavior could probably

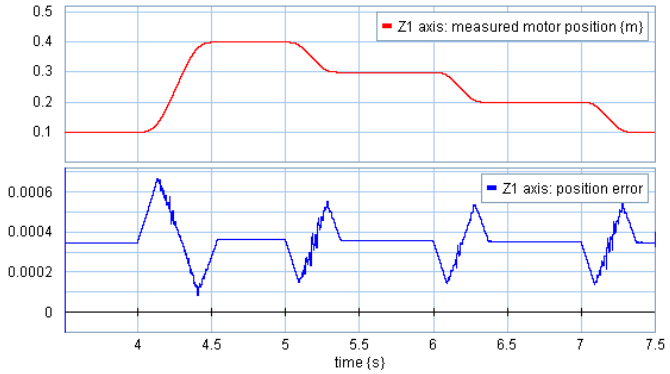


Fig. 5: Simulation results (for one motor) of configuration 1.a

damage the Tripod setup, this configuration was not executed again to show it in a figure.

Because configuration 1.a does not show good performance when controlling the real Tripod setup, a revision of this configuration is needed. First the cause of this behavior should be known and after that a new version of this configuration should be tested. The given behavior can be caused by the timing behavior of the control system. That's why timing behavior of different software components should be tested.

V. Real-time test programs

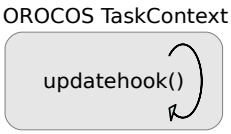
A. Modeling

To find out the cause of the given behavior, test programs are developed to test the real-time behavior of different software components. For each of the following, a test program is developed to test real-time behavior:

- OROCOS: This test program consists of an OROCOS TaskContext, running real-time at a given frequency. Each sample time, this TaskContext derives its own latency. After running a given amount of time, the real-time test stops, and outputs the maximum latency.
- OROCOS + SOEM: SOEM (Simple Open EtherCAT Master) is an EtherCAT master library, written in c [18]. This library is used by the CoE driver for OROCOS, that is used in this thesis. The just explained OROCOS test program consist of an OROCOS TaskContext. This TaskContext contains a function that is executed each sample time. In this function, communication via EtherCAT can be achieved by using the SOEM library. This real-time test program toggles one digital output of an EtherCAT module by using the SOEM library.
- OROMACS: This test program consists of an OROMACS based Main-Agent. This Main-Agent consist of an agent with real-time priority that derives its latency each sample time.

Figure 6 shows the object structure of each real-time test program. The latency of the SOEM driver influences the latency of the TaskContext in which it is running. When a MACS uses the SOEM driver in an agent, this will cause the same latency for the agent in which the SOEM driver is placed. This will result in a latency of the whole control system. In practice, it would be preferable that the latency of

OROCOS and OROCOS+SOEM:



OROMACS:

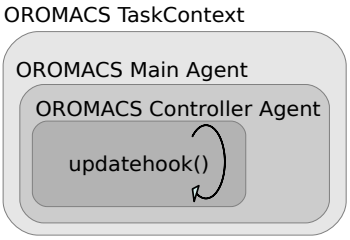


Fig. 6: Object structure of real-time test programs

the SOEM driver will not influence any latency in the control system. A suggestion to achieve that is to place the SOEM driver in another agent than the main agent of the control system. To test this behavior, a test program is developed in which a TaskContext is running separate from a TaskContext that contains the SOEM activity. The two are connected with each other via OROCOS ports. The SOEM TaskContext does not have a periodic activity, but the ports are implemented as event port. This implies that every time when new data is available on that port, it will call its updatehook function. An overview is given in figure 7.

OROCOS+SOEM(event-based):

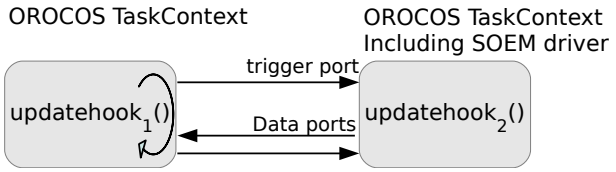


Fig. 7: Object structure of real-time test program for testing latency in OROCOS TaskContext and SOEM driver

B. Test conditions

The main goal of a real-time system is to guarantee timing behavior under all circumstances. The real-time system must also guarantee timing behavior in worst-case scenario. That's why a stress test program is developed to execute the real time test programs under heavy load. All given real time tests in this research are performed simultaneously with this stress test. The stress test includes the following functionality:

- 100 % CPU activity, using continuously called ping command.
- 100 % I/O activity using a continuously called tar command.
- 100 % cache activity using a Cache-Memory and TLB Calibration tool.

Figure 8 shows a screen shot of CPU usage when performing the stress test.

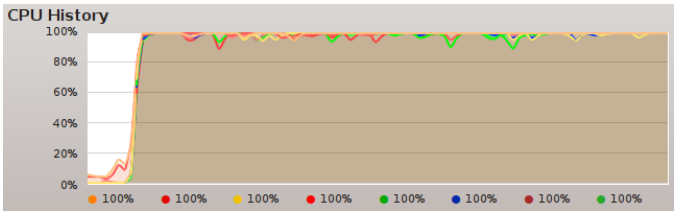


Fig. 8: CPU history when running the stress test

C. Test results

Test results of the real-time tests are shown in table I to IV. The given values are the latencies after running the test for 10 minutes. Table II and IV shows that the SOEM driver causes a relatively high maximum latency. However, this is not the case with the mean latency and standard deviation of it. Table

TABLE I: Mean (and standard deviation) latency (μSec) at 1 kHz

	Core-i7 Mean $\pm$ SD	Pentium 4 Mean $\pm$ SD
OROCOS	1 $\pm$ 2.8	6 $\pm$ 8.6
OROCOS + SOEM	1 $\pm$ 6.1	9 $\pm$ 11.4
OROCOS + SOEM (different threads) - OROCOS - SOEM	 1 $\pm$ 1.3 0 $\pm$ 1.4	 11 $\pm$ 14.1 5 $\pm$ 13.5
OROMACS	1 $\pm$ 1.7	10 $\pm$ 14.0

TABLE II: Maximum latency (μSec) at 1 kHz

	Core-i7	Pentium 4
OROCOS	81	75
OROCOS + SOEM	521	79
OROCOS + SOEM (different threads) - OROCOS - SOEM	 40 877	 83 803
OROMACS	71	93

TABLE III: Mean (and standard deviation) latency (μSec) at 10 kHz

	Core-i7 Mean $\pm$ SD	Pentium 4 Mean $\pm$ SD
OROCOS	0 $\pm$ 1.3	5 $\pm$ 6.9
OROCOS + SOEM	162 $\pm$ 4.8	n/a
OROCOS + SOEM (different threads) - OROCOS - SOEM	 0 $\pm$ 1.4 164 $\pm$ 4.9	 n/a n/a
OROMACS	0 $\pm$ 1.1	1 $\pm$ 1.8

TABLE IV: Maximum latency (μSec) at 10 kHz

	Core-i7	Pentium 4
OROCOS	209	50
OROCOS + SOEM	1127	57179
OROCOS + SOEM (different threads) - OROCOS - SOEM	 200 1138	 n/a n/a
OROMACS	33	41

IV shows a very high latency for the Pentium 4 PC, running at a sample frequency of 10 kHz. This is because the CPU usage has become too high. When the program needs more CPU capacity than possible, real-time behavior is not possible anymore. Figure 9 shows CPU usage when running the test program at a sample frequency of 10 kHz, without using the stress test. This shows that the program needs full CPU usage, which results in non-real-time behavior.

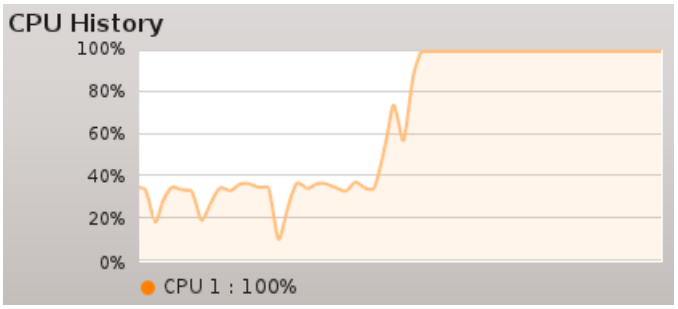


Fig. 9: CPU history when running the stress test

D. Conclusion

The given test results shows that the SOEM driver causes relatively high latencies. When this driver is used in an OROCOS TaskContext the latency of the TaskContext will be influenced by the SOEM driver.

For the same reason, it can be expected that the latency of an OROMACS agent will be influenced when using the SOEM driver within that agent. Probably this will influence the behavior of the whole OROMACS based control system. That's why it is not preferable to use the SOEM driver in this way within an OROMACS based control system.

The SOEM driver uses the standard linux network protocol stack. A suggestion for further research is to check the timing behavior of this network protocol stack and to consider an adaptation to achieve hard real-time behavior. An example of a hard real-time network protocol stack is RTnet [19]. However, this network protocol stack is currently not usable in combination with the RT-PREEMPT patch.

VI. Configuration 1.b: a working OROMACS based MACS

A. Modeling

The first presented MACS (configuration 1.a) did not show good performance when testing it with the real Tripod setup. Configuration 1.a consists of a Main-Agent that contains all the other agents. One of them is the actuator agent that includes the CoE driver. This CoE driver uses the SOEM library to communicate with the outside world. The real-time test results show that SOEM causes unwanted non-real-time behavior in some cases. The latency of the SOEM driver influences the latency of the agent in which it is running. This will result in a different behavior of the whole control system. In practice, it would be preferable that the latency of the SOEM driver will not influence any latency in the control system. A suggestion to achieve that is to place the SOEM driver outside the OROMACS based control system. Just like the real-time test program (fig. 7), this is implemented in the next control system: Configuration 1.b. Figure 10 shows a schematic view of configuration 1.a (which is another view of figure 4) and configuration 1.b. Only the deployment specification is shown. Just like the real-time test programs,

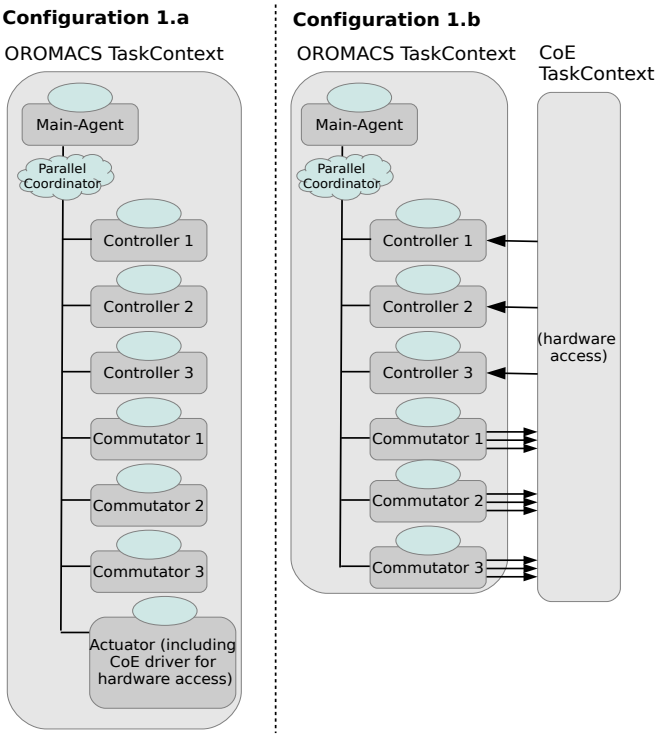


Fig. 10: Deployment specification of configuration 1.a and configuration 1.b. Ports and connections between sub-agents are invisible to keep it clear

configuration 1.b contains an agent that is not influenced by the latency of SOEM. The updated CoE TaskContext is given in figure 11. Each periodic activity is removed, and an event port is added to trigger each updatehook() function.

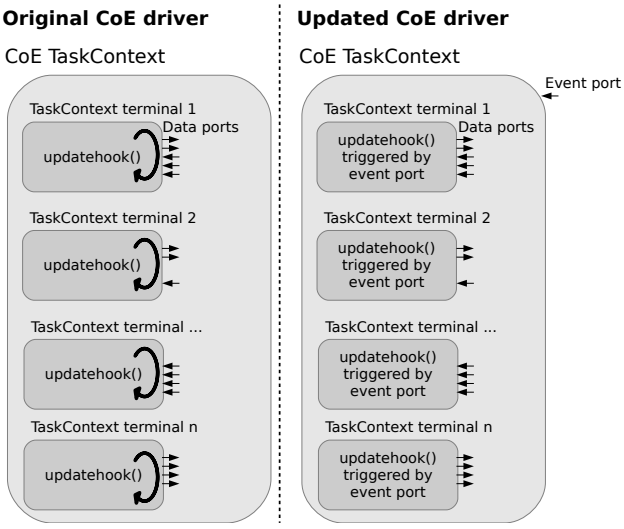


Fig. 11: Original and updated CoE TaskContext

B. Test

Timing results are shown in table VI and V. It clearly shows the difference between the latency of CoE TaskContext and a controller agent of configuration 1.b.

TABLE V: Mean (and standard deviation) latency (μSec) at 1 kHz

	Core-i7 Mean $\pm$ SD	Pentium 4 Mean $\pm$ SD
Configuration 1.b - OROMACS agent - CoE TaskContext	 1 $\pm$ 4.5 4 $\pm$ 10.5	 2 $\pm$ 4.3 6 $\pm$ 8.8

TABLE VI: Maximum latency (μSec) at 1 kHz

	Core-i7	Pentium 4
Configuration 1.b - OROMACS agent - CoE TaskContext	 149 735	 93 633

The controller agents of configuration 1.b contain a log function that logs all input and output data. Before closing the program, the data is written to disk. This data can be imported in 20sim to generate a graph. Figure 12 shows this graph. It shows that configuration 1.b is working with the real tripod setup.

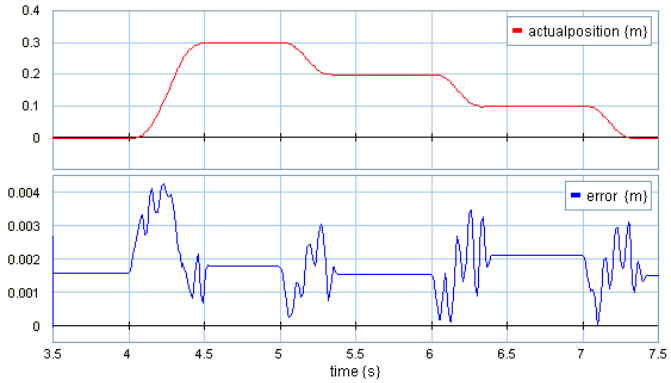


Fig. 12: Actual position and error when controlling the real Tripod setup

C. Conclusion

Configuration 1.a resulted in a non-working control system in combination with the real Tripod setup. Test results of configuration 1.b shows a working MACS when placing the CoE driver outside of it. This suggests that the latency of the SOEM library (that is used by the CoE driver) causes strange behavior in an OROMACS based MACS which will result in uncontrollable behavior.

VII. Configuration 2: A local safe-guarded MACS

A. Modeling

As already mentioned in the background information, a generalized control solution for simple mechatronic control problems has been developed at the University of Twente. This control problem is simple in the sense that it is given by a motion system with one degree-of-freedom.

The control problem in this report is given by the Tripod robot. Tripod consists of three linear motors that can move up and down to move a platform. When the platform of Tripod is removed, it consists of three linear motors that can move independently of each other. In this situation, controlling Tripod can be seen as three simple control problems. The control system of one motor is based on the generalized control solution for simple control problems and is given in figure 14.

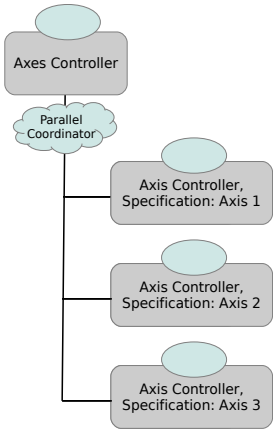


Fig. 13: Overview of Axes Controller

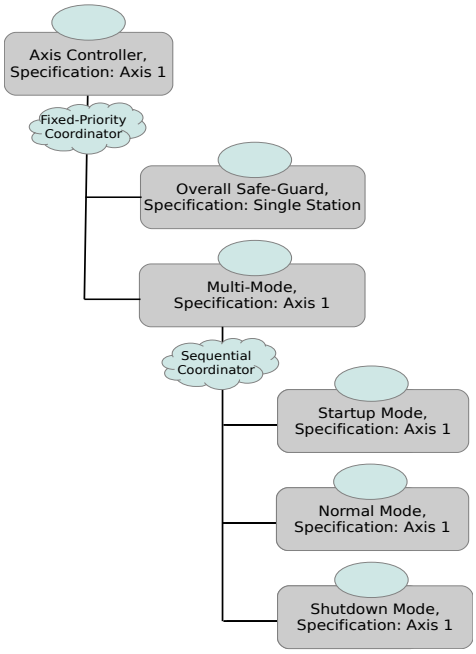


Fig. 14: Overview of Axis Controller

The generalized solution for simple mechatronic control problems is based on different design patterns. These design patterns are described by means of agents and sub-agents that can have different specifications. For example, the Overall Safe-Guard in figure 14 can have the specifications Single Station or Multi Station. In the case of simple mechatronic control problems, the scope of Overall Safe-Guard is limited to one axis (one station). That's why this agent has Single

Station specification in the generalized solution for simple mechatronic control problems. Each specification has its own implementation. So, the inside of each agent can be totally different. However, the interface to the outside world of each specification is the same.

In figure 14, one specification of Axis Controller is given. It consists of two sub-agents: an Overall Safe-Guard, and a Multi-Mode agent. Each sub-agent can decide for itself whether it wants to become active. When both sub-agents want to be active, the one with the highest priority will become active. This is because the coordination mechanism in the axis controller is Fixed-Priority. Because the Overall Safe-Guard has the highest priority, it will become active when it wants to become active.

The Multi-Mode agent in figure 14 consists of different sub-agents that become active in a sequential manner. This is because the coordination mechanism in Multi-Mode is Sequential. Each mode can decide for itself whether it wants to become inactive. For example, when the Startup-Mode agent wants to become inactive after five seconds, the Normal-Mode will become active after five seconds. Each mode consists of a motion generator and a feedback controller to control the motor of one axis.

The Overall Safe-Guard in figure 14 is responsible for detecting safety problems of different levels. Three different problem levels are possible: Dangerous, Serious and Warning. Each problem level has its own priority in which the Dangerous problem level has the highest priority. That's why the Overall Safe-guard consists of three sub-agents, coordinated by a Fixed-Priority coordination mechanism. An overview of the Overall Safe-guard is given in figure 15.

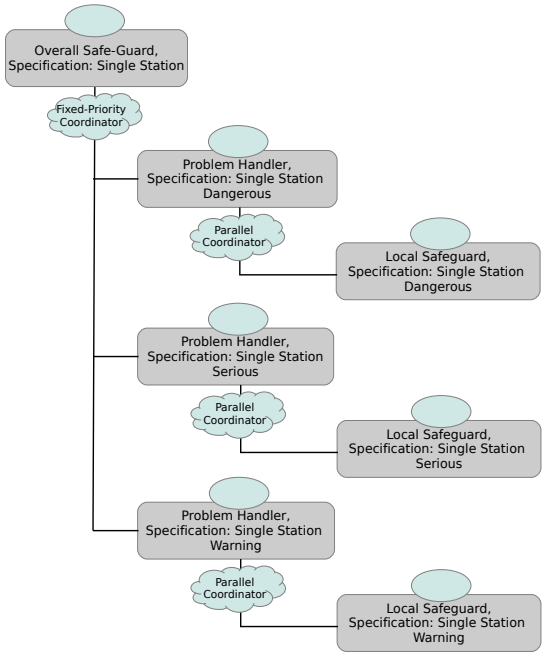


Fig. 15: Overview of Overall Safe-guard

The Local Safe-Guards in figure 15 are responsible for detecting and handling problems from a specific problem level. An overview such an agent is given in figure 16. The Local Safe-Guard is located inside one Axis Controller and only detects problems that are limited to one axis. That's why the Local Safeguard has Single Station specification and contains only one sub-agent. This sub-agent contains an Error Detection agent and a Single Safe-Guarded Activity agent. The Error Detection agent becomes active when it detects a specific problem. Because of the Master-Slave coordinating mechanism, the Single Safe-Guarded Activity becomes active only when the Error Detection agent becomes active. When the Single Safe-Guarded Activity becomes active, it handles the desired error mechanism.

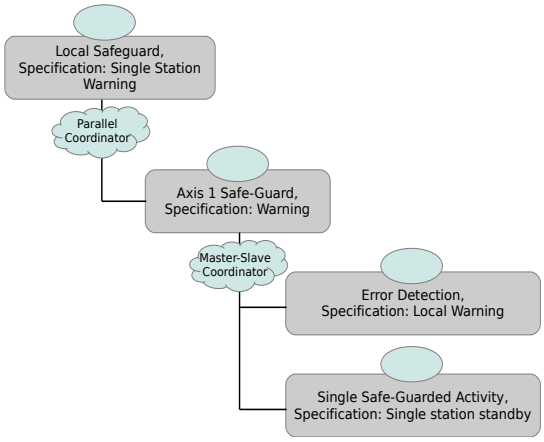


Fig. 16: Overview of an Local Safe-guard

The main-agent of Configuration 2 consists of three sub-agents. Each sub-agent is responsible for controlling one axis. The main-agent of configuration 2 is called Axes Controller. An overview of this Axes Controller is given in figure 13.

B. Simulation

When one Axis Controller detects an error, its Overall Safe-Guard agent becomes active and handles the error. Because the errors that are handled in this configuration are limited to one axis, the other Axis Controllers should not react to this (local) error. Figure 17 shows a simulation of configuration 2. The simulation shows that each motor is controlled according to a given reference motion. First part of the motion is controlled by the Startup Mode agent (see fig. 14). When the Startup Mode becomes inactive, the Normal Mode becomes active. At t=9 seconds, the reference motion of motor 1 contains a step. This step causes a relatively large positioning error of motor 1. That's why its Error Detection agent with Warning specification (see figure 16) becomes active. Because of the Master-Slave coordination mechanism, the single Safe-Guarded Activity becomes active and controls motor 1 to standby mode. This clearly shows an Axis Controller that handles a local error and operates independently of the others.

C. Test and Conclusion

When controlling the real Tripod setup with configuration 2, the same behavior as described in the last paragraph was visible. This shows a working controller that is based on the

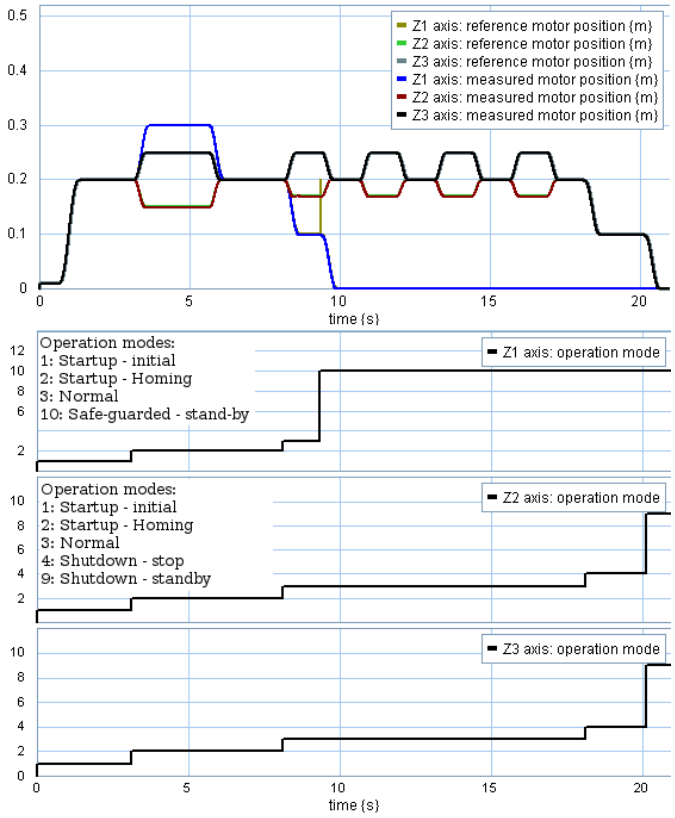


Fig. 17: Measured position and operation mode of each Axis

generalized control solution for simple mechatronic control problems. Test results of configuration 2 also show that the generalized control solution is reusable in a control problem that consists of a set of independent simple mechatronic control problems.

VIII. Configuration 3.a: A global safe-guarded MACS

A. Modeling

During a normal operation of Tripod, the platform has a safe working area that is shaped as a cylinder. Although the platform of Tripod can exceed the dimensions of the cylinder, it is not recommended because this will cause overloading of the joints. Exceeding its safe working area is an example of a global problem, because it depends on the positions of all linear motors.

The generalized control solution for complex mechatronic control problems deals with safety patterns and separates local problems and global problems. Local problems for Tripod are problems within the scope of one axis. Global problems are problems within the scope of more than one axis. Configuration 3.a implements a safe-guarded MACS for Tripod which is based on the generalized control solution presented in [6].

An overview of the Overall Controller of configuration 3.a is given in figure 18. The Overall Controller consists of two sub-agents: Overall Safe-Guard and Axes Controller. The Overall Safe-Guard is responsible for handling local and global problems. As in configuration 2, the Axes Controller consists of three sub-agents that are responsible for controlling each axis of Tripod. In configuration 2, each axis controller contains an agent that is responsible for handling local problems (figure 14). Because the Overall Controller in configuration 3 already contains an agent that is responsible for local problems, the Overall Safe-Guard is not part of each Axis Controller.

As already mentioned, the Overall Safe-Guard is responsible for handling local and global problems. Three problem levels are possible: Dangerous, Serious and Warning, in which each problem level has its own priority. That is why the Overall Safe-Guard consists of three sub-agents with a fixed priority coordinator. Each sub-agent is responsible for local and global problems from a specific problem level. An overview of the Overall Safe-Guard is given in figure 19.

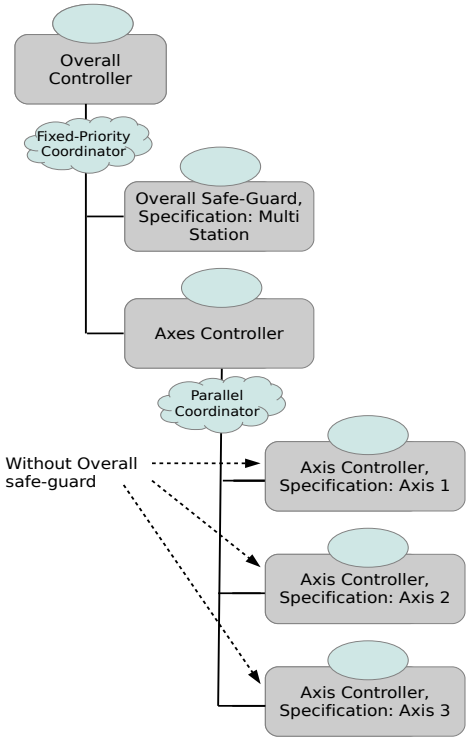


Fig. 18: Overview of Overall Controller

In figure 19, each Local Safeguard is responsible for handling local problems of a specific problem level. An overview of a Local Safeguard is given in figure 20. In case of Tripod, problems are local in the sense that they are in the scope of one axis. Because Tripod has three axes, the Local Safeguard consists of three sub-agents. These sub-agents are responsible for local problem handling of a specific problem level for a specific axis. As in configuration 2, each agent consists of an Error Detection and a Single Safe-Guarded activity. The latter is responsible for error handling that the first one has detected.

In figure 19, each Global Safeguard is responsible for handling global problems of a specific problem level. An overview of a Global Safeguard is given in figure 21. Each Global Safeguard consists of an Error Detection Agent and a Multi Safe-Guarded Activity. Because of the Master-Slave coordination mechanism, the Multi Safe-Guarded Activity becomes active only when the Error Detection agent wants to

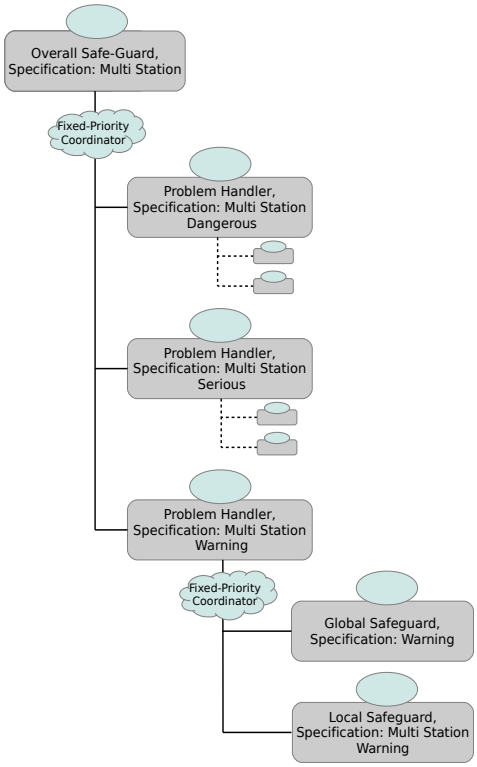


Fig. 19: Overview of Overall Safe-Guard

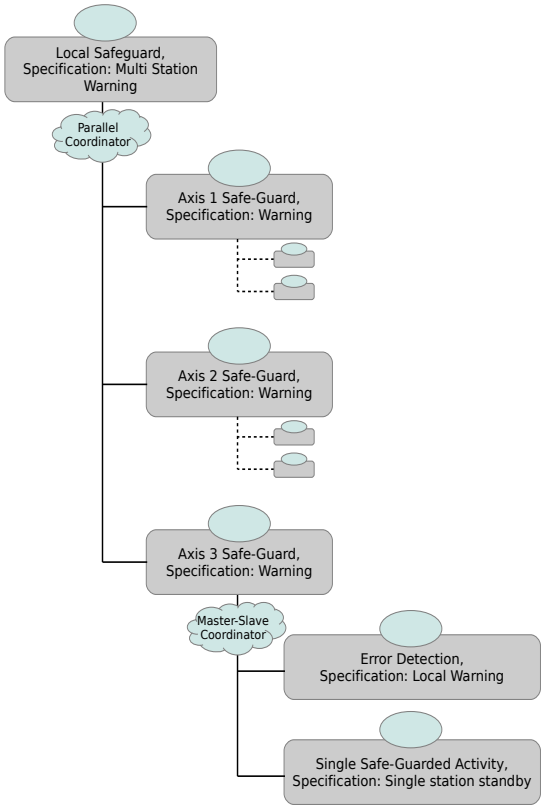


Fig. 20: Overview of Local Safe-Guard

become active. This is the case when the Error Detection agent detects a global error of a specific problem level. The Multi Safe-Guarded activity consist of three sub-agents to activate the error handling mechanism for each axis of Tripod.

B. Simulation

To check the error handling mechanism of local problems, a local error is created during simulation. Just like in the simulation of configuration 2 (see figure 17), a local error is created by simulating a relatively large step in reference position of motor 1. This should activate Axis 1 Safe-Guard, which should handle this local error. Figure 22 shows a simulation plot. After 10.5 seconds, a peak is shown in the reference position of motor 1. Figure 22 shows that the safe-guarded mode of motor 1 becomes active and controls this motor to standby mode. However, in contrast with configuration 2, motor 2 and 3 are not tracking the reference motion after the local error of motor 1 has occurred. This shows that local errors of one axis will influence behavior of the other axes, which should not be the case.

C. Test and Conclusion

When controlling the real Tripod setup with configuration 3.a, the same behavior as described in the last paragraph was visible. This shows a controller that handles local problems incorrect in the sense that a local problem will influence control behavior of other axes, which should not be the case. The next chapter will explain this behavior and will adapt this configuration to overcome this problem. This chapter shows that the general solution for complex control problems should be adapted to enable a good working local error handling mechanism.

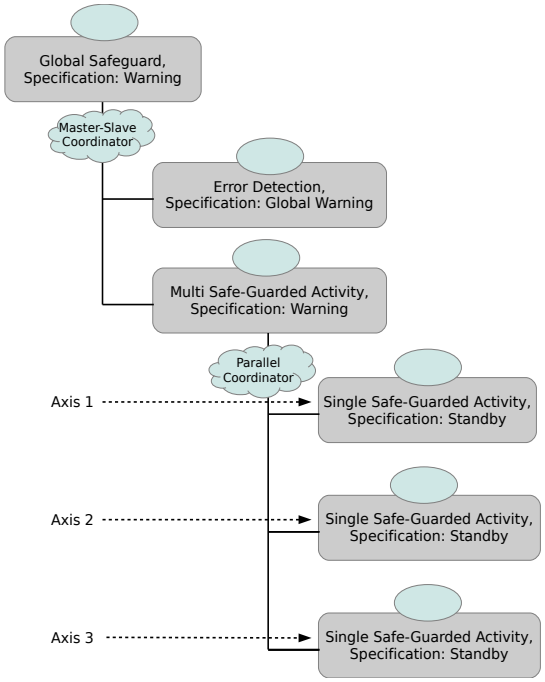


Fig. 21: Overview of Global Safe-Guard

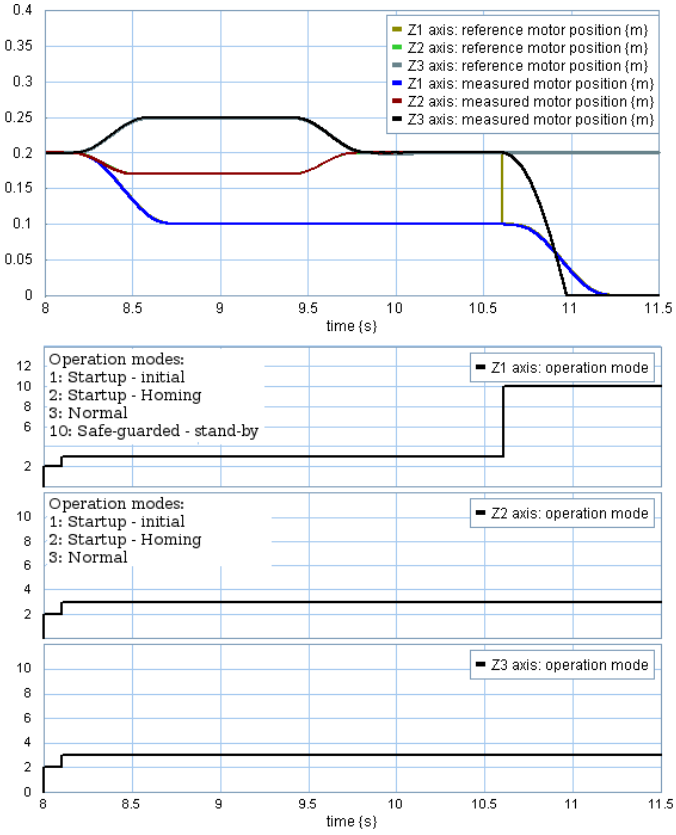


Fig. 22: Measured position and operation mode of each Axis

IX. Configuration 3.b: final safe-guarded MACS

A. Modeling

This chapter deals with an adaptation of configuration 3.a resulting in configuration 3.b. Simulation of configuration 3.a showed that local errors of one axis will result in different control behavior of other axes. This paragraph will explain this behavior.

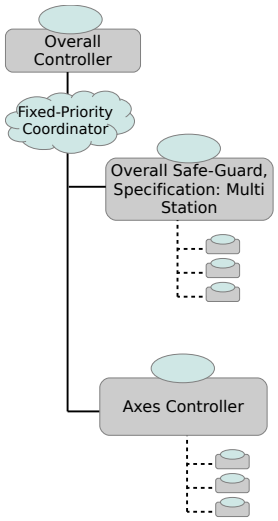


Fig. 23: Simple overview of Overall Controller

Figure 23 shows a simple overview of the Overall Controller of configuration 3.a and 3.b. This Overall Controller consists of an Overall Safe-Guard and an Axes Controller. When the Overall Safe-Guard wants to become active, it will become active because of the Fixed-Priority coordinating mechanism in the Overall Controller. In configuration 3.a, when a local error at one axis is detected, a sub-agent in the hierarchical structure of Overall Safe-Guard will become active. This sub-agent will handle the local error of that axis. When no error is detected at the other axes, no error handling mechanism of the other axes will become active. However, because of the Fixed-Priority coordinating mechanism, the Axes Controller will become inactive. In this situation, the Overall Safe-Guard will control one axis according to the error handling mechanism, and the Axes Controller will control no axes because it is not active anymore. In this situation the other axes are not controlled anymore by any agent. This situation has happened during simulation of configuration 3.a and clearly represents a wrong coordination mechanism.

The previously described situation has occurred because the Overall Safe-Guard (in figure 23) handles local errors, which will result in deactivation of the Axes Controller. To overcome this problem, the Overall Safe-Guard could be changed in such a way that it does not handle local errors anymore, but only global errors. Local errors can than be handled by means of the Axis Controllers as used in configuration 2 (see figure 24). In this situation, each axis controller handles its own local problems. When a local problem at one axis controller is detected, the other axis controllers will stay active, because of the Parallel coordination mechanism in the Axes Controller. In this situation, a local problem at one axis should not influence control behavior of the other axes. Configuration 3.b implements the adapted version of configuration 3.a.

B. Test

As in configuration 3.a, a peak in reference position of motor 1 is created to obtain a local error. Figure 25 shows the results when controlling the real Tripod setup. After 11 seconds a peak is shown in reference motion of motor 1, resulting in a local error. The figure shows that motor 1 goes into local safe-guarded mode which will standby the motor. In contrast with configuration 3.a, the other motors are controlled to their reference position as should happen. Because of the local error handling of motor 1, the platform of Tripod exceeds the safe operation area at 11.5 seconds, which is an example of a global error. As a result, the Global Safe-guarding mode becomes active, which controls all motors to standby mode. This simulation shows a local error handling mechanism that does not influence other local controllers. It also shows a global error handling mechanism that takes over control of all motors.

C. Conclusion

Test results shows a controller that handles local problems that does not influence the behavior of other axes, which should be the case. It also shows a global error mechanism that takes over control of all axes to handle these errors. This paragraph shows that the adapted general solution for complex control problems enables a good local and global error handling mechanism.

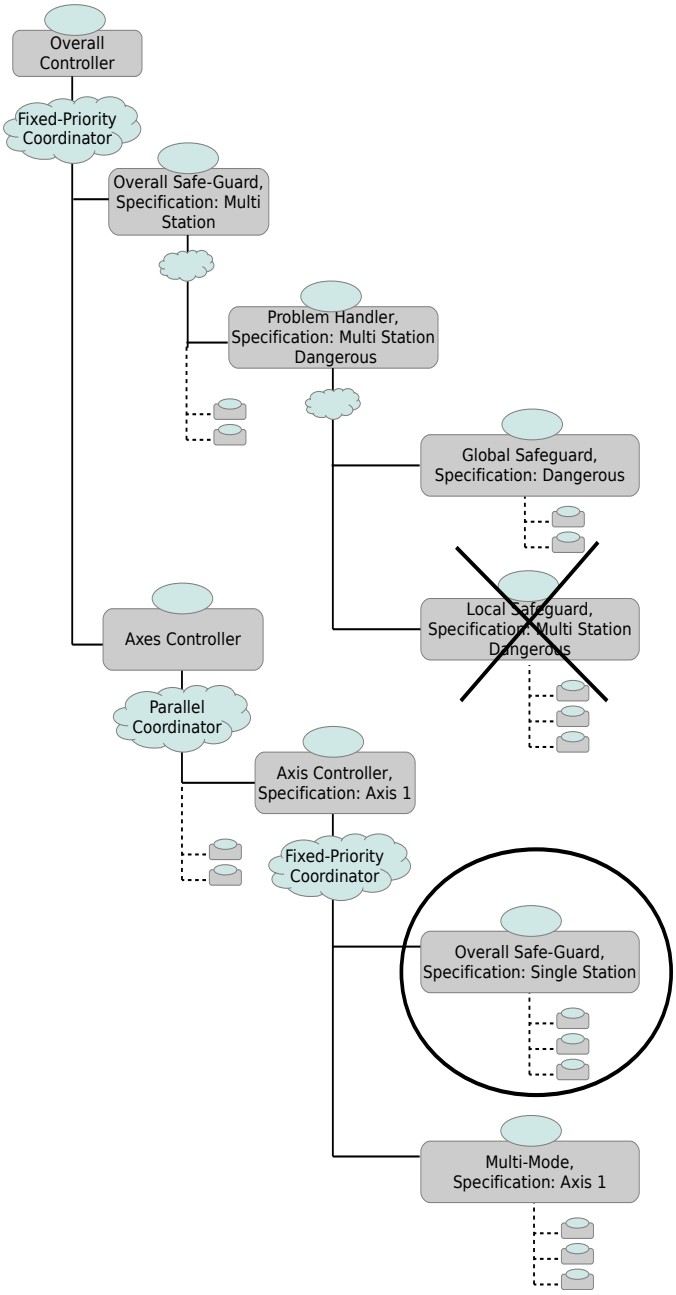


Fig. 24: Adaptation of configuration 3.a to configuration 3.b

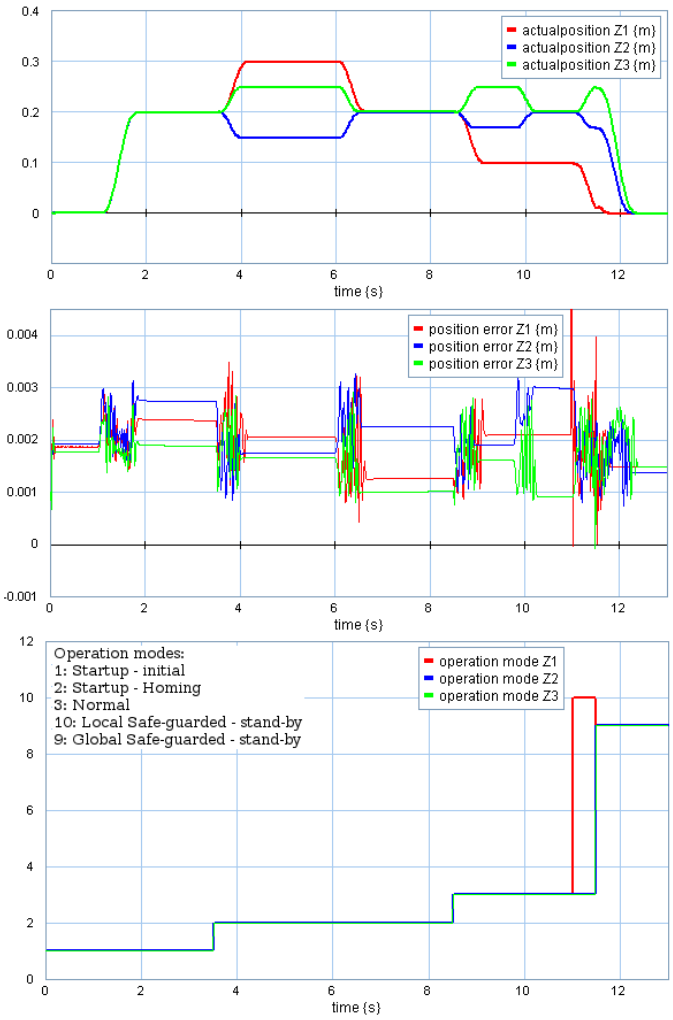


Fig. 25: Measured position and operation mode of each Axis

X. Conclusions and Recommendations

This paper evaluates the timing, communication, reusability and safety handling issues of a proposed generalized control system for mechatronics systems on a three D.O.F. parallel manipulator. The test set-up has been modified to use EtherCAT, OROCOS and OROMACS.

The given real-time test results show that the SOEM driver causes relatively high latencies. When using this SOEM driver in an OROMACS agent, the behavior of the whole OROMACS based control system can be influenced in such a way its behavior is unpredictable. In contrast, test results show that placing the SOEM driver in a separate OROCOS thread, outside the OROMACS based control system, results in a control system with the desired behavior. However, the problem of relatively high latencies is not solved. Because the SOEM driver is used as a communication mechanism between the controller and plant, it can be expected that its timing behavior will still influence the behavior of the whole system. A Recommendation for further research is to fix this behavior of the SOEM driver in combination with OROCOS and OROMACS. Because the SOEM driver uses the standard linux network protocol stack, it is recommended to study the timing behavior of this network protocol stack, and to consider an adaptation to achieve hard real-time behavior.

Test results also show a working controller based on the generalized control solution for simple mechatronic control problems. This control solution is easily reusable for different simple control problems that are independent.

Additional test results show that the generalized control solution for complex mechatronic control problems causes changes in behavior of well-operating axes because of a local problem in another axis. In this research, an adaptation is done to this generalized control solution which results in the desired local and global error handling.

REFERENCES

[1] A. J. N. van Breemen, "Agent-based multi-controller systems," Ph.D. dissertation, University of Twente, 2001.

[2] T. Johansen and R. Murray-Smith, *The Operating Regime Approach to Nonlinear Modelling and Control*. Taylor & Francis, 1997.

[3] H. Proenca and E. Oliveira, "Marcs - multi-agent railway control system," *Advances in artificial intelligence*, vol. 3315, pp. 12–21, 2004.

[4] H. D. Wang, G. Z. Qiu, and S. S. Huang, "Cement industry control system based on multi agent," *Journal of central south university of technology*, vol. 11, pp. 41–44, 2004.

[5] A. J. N. van Breemen and T. J. A. de Vries, "Design and implementation of a room thermostat using an agent-based approach," *Control engineering practice 9*, pp. 233–248, 2001.

[6] D. B. Phong, "Safe-guarded multi-agent control for mechatronic systems," Ph.D. dissertation, University of Twente, 2011.

[7] Orocos, "The open robot control software project," 2011. [Online]. Available: http://www.orocos.org

[8] M. Eglence, "Design and realization of a safe control system for a parallel manipulator," Master's thesis, Control Laboratory, University of Twente, 2003.

[9] W. J. R. Velthuis, "Learning feed-forward control - theory, design and applications," Ph.D. dissertation, University of Twente, 2000.

[10] B. Burgers, "Automated i/o access with coe in orocos," Master's thesis, Control Laboratory, University of Twente, 2010.

[11] Beckhoff, "Beckhoff automation," 2011. [Online]. Available: http://www.beckhoff.com

[12] R. Stephan, "Real-time linux in control applications area," Master's thesis, Control Laboratory, University of Twente, 2002.

[13] M. T. J. Jones, "Anatomy of real-time linux architectures," 2008.

[14] P. McKenney, "A realtime preemption overview," 2005. [Online]. Available: http://lwn.net/Articles/146861/

[15] Controllab, "Controllab products b.v." 2011. [Online]. Available: http://www.20sim.com

[16] Z. Bozlak, "Co-simulation of an orocos-based controller and a 20-sim plant," 2009.

[17] L. Prokop and P. Grasblum, "3-phase pm synchronous motor vector control using a 56f80x, 56f8100, or 56f8300 device," Freescale Semiconductor, Application Note 1931, 2005.

[18] SOEM, "Simple open ethercat master," 2011. [Online]. Available: http://developer.berlios.de/projects/soem

[19] RTnet, "Hard real-time network protocol stack for xenomai and rtai," 2011. [Online]. Available: http://www.rtnet.org