A FLEXIBLE HARDWARE TEST AND DEMONSTRATION PLATFORM FOR THE FRACTIONATED SYSTEM ARCHITECTURE YETE

Florian Kempf[1], Roland Haber[2], Tristan Tzschichholz[2], Tobias Mikschl[3], Alexander Hilgarth[3], Sergio Montenegro[3], and Klaus Schilling[1]

[1]*Robotics and Telematics Department, University of Würzburg, 97074 Würzburg, Germany,*
{kempf|schi}@informatik.uni-wuerzburg.de
[2]*Zentrum für Telematik e.V., Magdalene-Schoch-Straße 5, 97074 Würzburg, Germany,*
{roland.haber|tristan.tzschichholz}@telematik-zentrum.de
[3]*Aerospace Information Technology Department, University of Würzburg, 97074 Würzburg, Germany,*
{tobias.mikschl|alexander.hilgarth|sergio.montenegro}@uni-wuerzburg.de

ABSTRACT

This paper introduces a hardware-in-the loop test and demonstration platform for the YETE system architecture for fractionated spacecraft. It is designed for rapid prototyping and testing of distributed control approaches for the YETE architecture subject to varying network topologies and transmission channel properties between the individual YETE hardware nodes.

Key words: Hardware-in-the-loop test platform; Fractionated spacecraft architecture; Distributed networked control.

1. INTRODUCTION

One of the trends in the space sector in the last decade is the transition from monolithic spacecraft architectures towards more flexible approaches while still maintaining a high degree of reliability and robustness. This development paves the way for novel collaborative distributed spacecraft concepts with a high degree of modularity as described in [10]. A spacecraft design, in which the functional capabilities of a usually single monolithic spacecraft are split and distributed among several wireless interconnected cooperating homogeneous or heterogeneous spacecraft(-subcomponents) is called a fractionated spacecraft architecture, as further detailed in [1] and [3]. The advantages of such an architecture are increased mission and in-orbit robustness, flexibility to extend system features later on and a lowered mission recovery cost [2], [1]. The bigger the substituted monolithic system and the longer the desired mission duration, the higher is the gain in fractionation value as detailed in [9] for the exemplary F6 system architecture.

Despite the many advantages of a distributed system architecture, such an approach also faces several challenges. For one as all system components/modules communicate over wireless links the varying short- and long-range transmission channel properties, like delay or packet loss, have to be considered in the hardware and software design. Another challenge is related to the often heterogeneous system composition that requires the system to balance the work load distribution according to the individual component capabilities and also to take the different module constraints into account, e.g. actuator limits or power restrictions.

To test the performance of such a distributed system, in this particular case the fractionated spacecraft architecture YETE, under the afore mentioned influences we developed a demonstration and test platform that allows us rapid prototyping of core YETE components and an easy extension of the overall system test complexity. This flexible test platform will be introduced in this paper.

After a brief introduction of the YETE architecture in section 2 the individual parts of the test platform are introduced. The paper is then concluded with a brief outlook on further and currently ongoing work in section 8.

2. SPACECRAFT ARCHITECTURE YETE

In the domain of modern networked spacecraft and unmanned systems employed in search-and-rescue scenarios, robustness to failures of the communication network (high delays, broken links) and individual system components (sensors, actuators, and processors) is a desired trait of the utilized system architecture. The developed fractionated systems architecture 'YETE' realizes this robustness by enforcing a strong modularity on hardware and software level and employing distributed approaches whenever applicable (see [4] and [5]). Failures in system component/subsystem processors are leveraged by transferring the necessary subsystem computations to a gen-

eral purpose computing cluster, which consists of each subsystem processor and several computation modules. Furthermore, failures of whole components (sensors / actuators) is mitigated by the task-migration capability and Publisher/Subscriber communication architecture of the middleware 'RODOS', which is running on all components (also called nodes) in YETE (for a further description of RODOS see [4],[8] and [7]). The resulting loose coupling between data sinks (e.g. sensors), sources (e.g. actuators) and processing tasks allows the switching from a failed component to a working one without interrupting the rest of the system. Another advantage of the high modularity in the YETE architecture is the re-usability of software- as well as hardware components. This is especially advantageous in the space industry, where changing mission requirements often mean major redesigns of an existing spacecraft architecture.
One very important part of YETE is the control of the hardware nodes in a single spacecraft as well as the control of several collections of nodes, e.g. a formation of YETE spacecraft. In YETE the focus lies on robust distributed control approaches (e.g. see [5]). To efficiently validate the performance of implemented approaches we developed a hardware-in-the-loop test platform, which will be introduced in the following sections.

3. HW-IN-THE-LOOP APPROACH

The YETE hardware-in-the-loop test platform was designed to satisfy several criteria. It should allow rapid testing and prototyping of distributed control approaches running in real-time on several YETE hardware nodes equipped with sensors, actuators and processors. It should furthermore allow to test the impact of a variety of different node network structures with varying channel properties, ranging from intra-satellite wireless bus to inter-satellite formation network configurations. As a result the hardware-in-the-loop test platform consists of three parts, a synthesized *Simulink* RODOS task running the control algorithm on one or more nodes, a software network simulation which substitutes part of the hardware communication interfaces of the nodes and therefore creates the interconnections between them and the YETE hardware demonstrator which consists of the actuators, sensors and embedded computers representing several YETE nodes.

Each of these parts will be described in more detail in the following sections. In section 4 the network simulation will be described and how it is integrated with the YETE nodes. The following section 5 describes how the Simulink task is generated so that it can run under RODOS on an embedded hardware node. Section 6 briefly covers the distributed control approach that is utilized in this system and finally in section 7 the structure of the hardware demonstrator is described in more detail.

4. COMMUNICATION SIMULATION

To allow the testing of different control algorithms under controlled network conditions we decided to use the discrete event simulator Omnet++ as simulation environment.
The first requirement we imposed on the network simulation is that it should substitute the simulated real system components as accurately as possible. One prerequisite for that is that the simulation is transparent for all non-simulated components in the system. In a later project stage the simulation can then be substituted by real hardware without the need for changes to the non simulated system components. To achieve this we integrated the Omnet++ simulation seamlessly into the RODOS communication layer, this can be seen in figure 1.

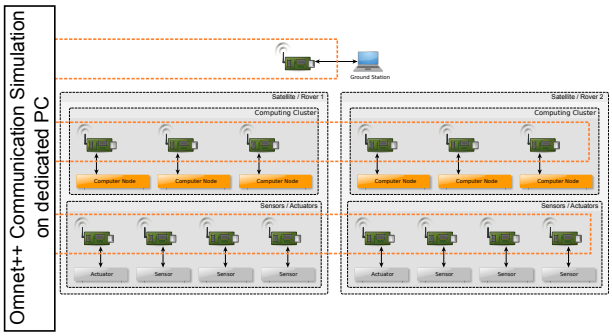

Figure 1. Integration of Omnet++ into the YETE system.

Furthermore the simulation itself should introduce minimal simulation overhead into the system. For this purpose we designed the simulation to be run on a powerful external dedicated PC, to limit the influence of simulation calculations on the simulated system process. The introduced delay by the simulation can be seen in Figure 3. As our distributed control loop is running with an update rate of $10Hz$ and most of the transmitted packets have a delay of less than $150\mu s$ and strictly under $1ms$ the simulation overhead in our case is negligible.

We also extended the Omnet++ Real Time Event Scheduler to handle asynchronous real time events coming from a RODOS-to-Omnet++ Gateway, which is further described below.
Integration of external real-time applications into Omnet++ has already been done before [6], however none of the proposed concepts apply in this case. The reason is, that the external application, RODOS in this case, will be running on several distributed nodes, some of which are embedded devices. Therefore the link between the RODOS Operating System and the Omnet++ communication simulation consists of two parts, one is platform dependent and is running on the RODOS side and one is platform independent and is running on the Omnet++ side. The communication between the two parts can be done via a wide variety of interfaces (TCP/IP-Ethernet, RS232, CAN-Bus, etc.) to support integra-

tion of all the heterogeneous YETE nodes. The part on the RODOS side is implemented as a RODOS *hardware link-interface*. This link-interface is part of the RODOS hardware-abstraction-layer and can later be exchanged by link-interfaces to real communication hardware, e.g. a Bluetooth Low Energy (BLE) module. This way the simulation appears transparent to tasks running in the RODOS system.

On the Omnet++ side a RODOS gateway module receives RODOS package data from the connected nodes, feeds it into the simulation network and transmits received data from the simulation side back to the destination nodes. A RODOS gateway module is shown in Figure 2, it consists of a *linkcoordinator* module, which manages the asynchronously received/transmitted data from/to the RODOS nodes over various interface modules, which perform the low-level communication hardware access.

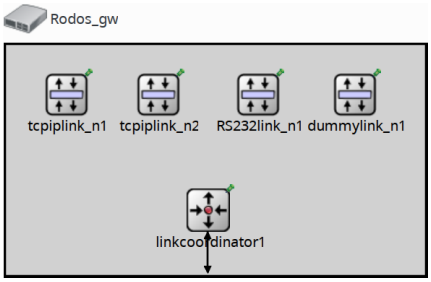

Figure 2. RODOS-Gateway-Module in Omnet++ which acts as the link between the simulation and the hardware nodes.

Further details regarding the implementation of intra- and inter-spacecraft link simulation in Omnet++ can be seen in [4].

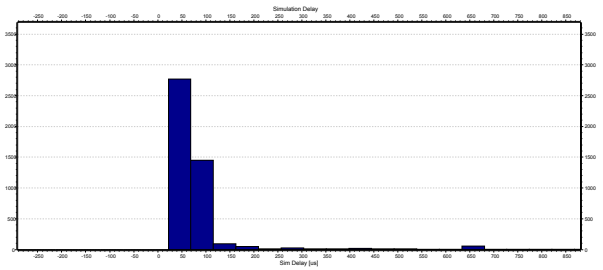

Figure 3. Communication delay introduced by the simulation in [μs]. The introduced delay for most of the 4687 transmitted packets is less than 150 [μs].

5. MATLAB-SIMULINK INTEGRATION

In order to simplify the creation of control algorithms for the proposed distributed computing tasks, a MATLAB Simulink connection to RODOS has been developed. This allows for rapid modeling of control processes that can also be translated into C-language as well as machine code for running on a target microcontroller or any other system. Once a model has been created, the Simulink infrastructure is used to compile code into a standalone binary, which contains a RODOS operating system running the model as an integrated task (Figure 4). These Simulink models also include RODOS data transfer interfaces in the form of Publisher and Subscriber blocks. Therefore, it is possible to transmit signals between different tasks as well as between dedicated hardware systems, which include STM32F4, UDOO, and X86 platforms (Figure 5). This method also creates an interface between sensors and actuators of different hardware architectures. Along with Simulink running in external mode, we are able to perform in-system debugging, which allows for real time access to the active tasks. However, external mode is currently limited to TCP connections on x86 systems.

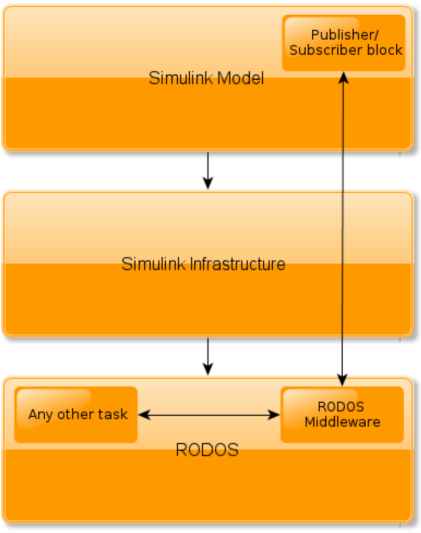

Figure 4. Simulink-RODOS interface including Publisher and Subscriber blocks for data transfer.

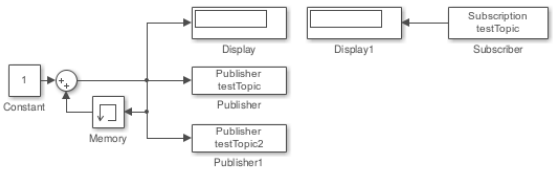

Figure 5. Exemplary Simulink model being run as a RODOS Task. Integers are being displayed and transfered using the Publisher and Subscriber blocks.

In figure 6 the system trajectory for two Simulink processes, one running in Matlab, the other running as a task inside a RODOS hardware node, can be seen. In both cases the same PD-Controller tries to drive a simple double integrator system $m\ddot{\vec{x}} = u; \vec{y} = \vec{x}$ from initial state

$\dot{\vec{x}}_0 = [0.1, 0.3]^T, \vec{x}_0 = [0, 0]^T$ to the exemplary target point $\vec{x}^* = [-3, 0]^T$. On the hardware the system state is sampled with a frequency $f_s = 1\,Hz$ and as can be seen in figure 6 both trajectories of the Simulink controlled systems align, which demonstrates a successful integration of Simulink into RODOS.

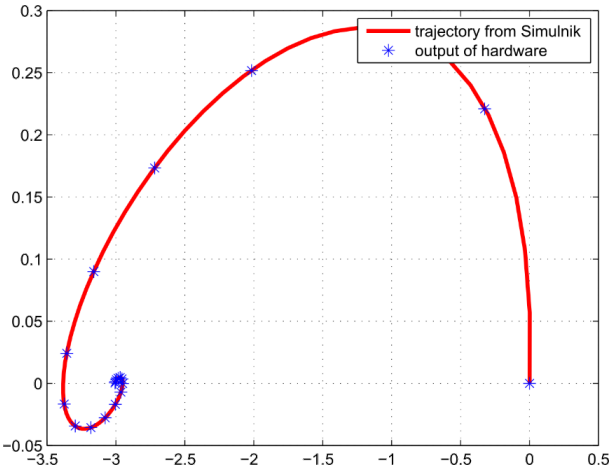

Figure 6. Comparison of two Simulink control process performances. One (red curve) is running in Matlab, the other one (blue dots) is running on-board a RODOS hardware node and is sampled with $f_s = 1\,Hz$.

6. DISTRIBUTED CONTROL APPROACH

The structure of the current distributed control approach in YETE is a master-slave architecture, which is visualized in figure 7. On the master hardware node a RODOS Simulink task is executing a H_∞ controller, which controls the state of both, the master and the slave hardware node according to a reference input y_{ref}. The controller is optimized to exhibit a low disturbance sensitivity on d_S, d_M and d_y as well as a good reference tracking of the master and slave. It furthermore needs to handle a lossy slave state feedback $\hat{y}_S$ subject to a certain packet drop probability, which simulates a disturbed wireless connection between the slave system and the master.

For more details on the controller design and system model we kindly refer you to [5].

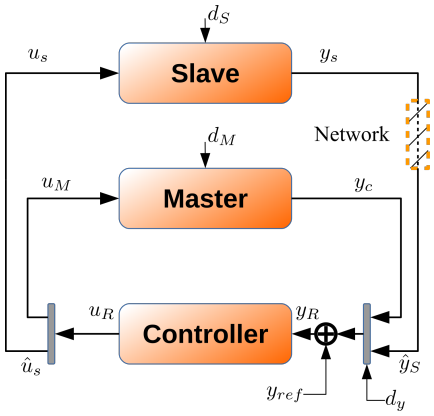

Figure 7. Distributed control system structure in YETE as master-slave architecture. The H_∞ controller running on the master controls both systems according to a external reference trajectory. The slave state feedback is subject to packet drops.

7. HARDWARE DEMONSTRATOR SETUP

As a means of demonstrating the proposed distributed control algorithms, a test platform has been developed (Figure 8). This demonstrator consists of two identical pendulums, each equipped with sensors (gyroscope) and actuators (propeller), which are controlled by separate 32-bit ARM processors. These pendulums in themselves function independently from one another. Each processor therefore stabilizes its pendulum by controlling the propeller velocity accordingly. With the introduction of a Bluetooth link between the two processors, we were able to establish a coupling that forms the basis for distributed computing and control of the propellers. Additionally, a computer for data management as well as active control of the indirectly connected propellers is also coupled over Bluetooth.

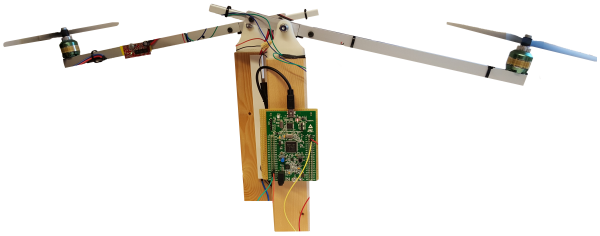

Figure 8. Test platform for distributed control algorithms consisting of two identical pendulums with Bluetooth interconnection for data exchange.

Figure 9 illustrates the data flow from each sensor to the corresponding MCU and ultimately to the respective rotor. The before mentioned Bluetooth data exchange between the two MCUs as well as between the MCUs and the PC is also indicated. During operations, the PC is running a graphical user interface, which provides data display as well as the ability to actively alter propeller

velocities.

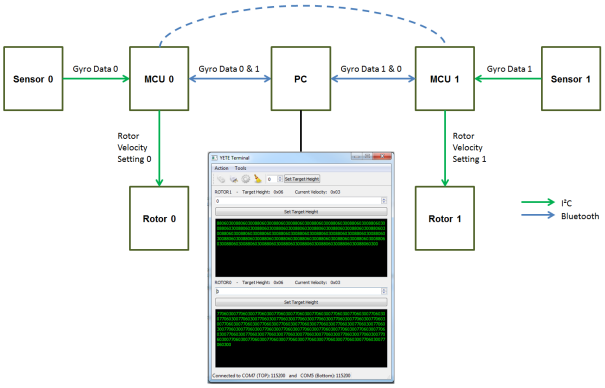

Figure 9. Data flow using I^2C and Bluetooth within and between both pendulums.

In a first demonstration, an exchange of sensor data is implemented in the running cycle of the microcontrollers. This implementation already allows for the possibility of comparing and/or replacing sensor outputs. In the case of a temporary sensor failure in one of the propellers, the data received from the other microcontroller can be used instead. Similarly, one of the microcontrollers may entirely rely on the external sensor data and thus completely omit its own inputs in order to avoid discrepancies between the two readouts. In addition to simply exchanging sensor data, one of the MCUs can take over computation tasks of the other, thereby processing the foreign sensor readouts and returning the calculated results. This procedure demonstrates on a small scale how distributed computing can help unburden processors during a high system load.

The combined control performance for the two pendulums in a master-slave configuration can be seen in figure 10. The states of the master q_r and slave system q_l should follow individual sinusoidal reference trajectories q_r^* and q_l^*. On the bottom of the figure the packet drop events in the state feedback from the slave is plotted. With a one indicating a packet drop. Despite the many packet drops that occur with a probability of $p = 32\,\%$, the controller manages to keep the two systems on the respective reference trajectories. However as seen in the middle of the figure, the control effort for the slave system is increased to compensate for the estimation error of the slave system state.

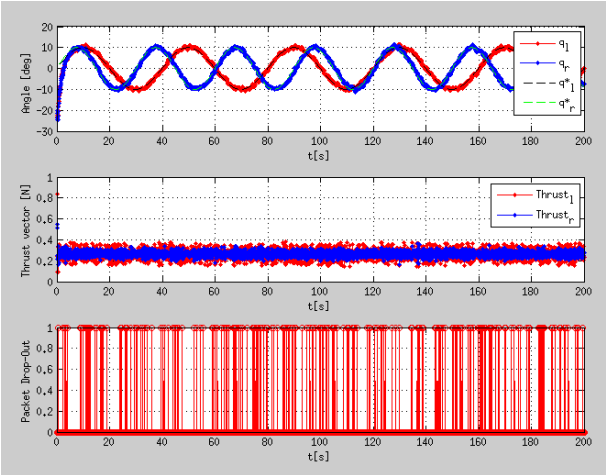

Figure 10. Master-Slave distributed control performance for the two pendulum hardware setup. The master state q_r and slave state q_l are controlled to follow two sinusoidal reference trajectories q_r^* and q_l^*. The packet-drop probability in the slave feedback is $p = 32\,\%$.

8. CONCLUSION

In this paper we introduced the individual components of a test and demonstration platform for the fractionated system architecture YETE. By directly integrating a software network simulator and the controller design tool Simulink into a hardware-in-the-loop test setup we maintained the flexibility of these modular software tools for rapid prototyping while gaining the accuracy of a verification directly on the test hardware. Furthermore promising test results of the individual components and the combined test system were presented.

We are currently in the process of also integrating the YETE test-environment on the Space Maneuver Simulator (SMS), developed at the Satellite Aerospace Information Technology Department at the University of Würzburg. This will allow us to perform more elaborate tests of the YETE architecture, e.g. robustness of the distributed control regarding thrusting errors and multi SMS-vehicle resource sharing and cooperation.

Two further enhancements regarding the YETE system that are currently being implemented are the substitution of the bluetooth inter-module communication by ultra-wideband transceivers and the transition from a centralized time synchronization approach to a decentralized one based on a maximum value consensus.

In the future we aim to also test another distributed motion planning and control approach based on Rapidly-Exploring-Random-Trees (RRT) and more in depth FDIR tests, especially for hardware node failures.

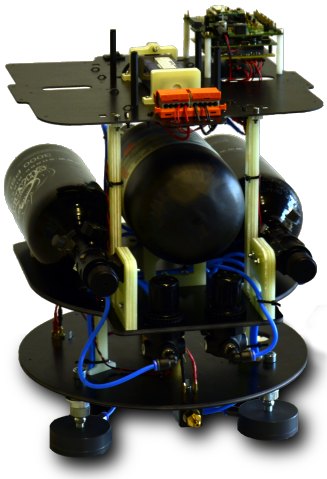

Figure 11. Three DoF Space Maneuver Simulator (SMS) of the Satellite Aerospace Information Technology Department of the University of Würzburg.

ACKNOWLEDGMENT

The authors would like to thank the Space Agency of the German Aerospace Center (DLR) for funding the project with federal funds of the German Federal Ministry of Economics and Technology (BMWi) under 50RA1330.

REFERENCES

[1] Owen Brown and Paul Eremenko. Fractionated space architectures: a vision for responsive space. Technical report, DTIC Document, 2006.

[2] Owen Brown, Paul Eremenko, and B Hamilton. The value proposition for fractionated space architectures. *Sciences*, 99(1):2538–2545, 2002.

[3] J Guo, DC Maessen, and EKA Gill. Fractionated spacecraft: the new sprout in distributed space systems. In *60th International Astronautical Congress: IAC 2009, 12-16 October 2009, Daejeon, Republic of Korea*, 2009.

[4] F. Kempf, A. Hilgarth, A. Kheirkhah, T. Mikschl, T. Tzschichholz, S. Montenegro, and K. Schilling. Reliable networked distributed on-board data handling using a odular approach with heterogeneus components. In *4S Symposium*, May 2014.

[5] A Kheirkhah, F Kempf, T Tzschichholz, and K Schilling. Robust distributed control for a mechanical-electrical demonstrator considering communication constraints. *IFAC-PapersOnLine*, 48(10):246–251, 2015.

[6] Christoph P Mayer and Thomas Gamer. Integrating real world applications into omnet++. *Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2*, 2008.

[7] Sergio Montenegro. Rodos: Real time kernel design for dependability, 2010.

[8] Sergio Montenegro, Vladimir Petrovic, and Gunter Schoof. Network centric systems for space applications. In *Advances in Satellite and Space Communications (SPACOMM), 2010 Second International Conference on*, pages 146–150. IEEE, 2010.

[9] Mohsen Mosleh, Kia Dalili, and Babak Heydari. Optimal modularity for fractionated spacecraft: The case of system f6. *Procedia Computer Science*, 28:164–170, 2014.

[10] James Skinner, Tollefson Mark, and Jeremy Rosenstock. Cooperating intelligent agents for distributed satellite systems. In *Proceedings of the AIAA Civil and Defense Systems Conference, Hunstville AL*, 1998.