INTEGRATING THE EUROPEAN PROXIMITY OPERATIONS SIMULATOR WITH THE FORMATION FLYING TESTBED

F. Rems,
GSOC, German Aerospace Center, Oberpfaffenhofen, Germany,
Institute of Astronautics, Technische Universität München, Garching, Germany

Abstract

The European Proximity Operations Simulator EPOS (part of GSOC, Oberpfaffenhofen near Munich) allows simulating Rendezvous and Docking scenarios involving two spacecraft and integrating rendezvous sensors (Hardware-in-the-Loop). Mockups of the spacecraft are mounted on two industrial robots. Both spacecraft can be moved in six degrees of freedom each, thus simulating relative orientation. A real-time control system operates the robots, involving the Real-Time Operating System (RTOS) VxWorks combined with Matlab/Simulink Real-Time Workshop. This environment demands that any customer adapt his simulation code, satellite simulator etc. to EPOS' real-time control. In detail, this involves considerable effort and time which may obliterate the benefits of a Hardware-in-the-Loop simulation with EPOS. This paper presents the author’s diploma thesis in a compressed form. Its aim is to reduce this effort distinctly by designing, implementing and testing a software package which connects any external satellite simulator, “as it is”, via Ethernet to the EPOS real-time control system. Although this strategy solves many problems, new ones are created, like the fact that a non-deterministic network, like Ethernet, is used in a real-time environment. To cope with these problems, an application layer communication protocol is developed, specifically tailored to meet EPOS' needs. It comprises two sub-protocols, the Simulation Connection Protocol (SCP) and the Remote Simulation Protocol (RSP). Among many other tasks, these protocols realize a data connection between two Simulink models, monitor packet delay, manage the interaction between an external simulator and EPOS (realization of starting conditions, timing...) and interpolate the robot trajectory in-between external simulator commands. These communication protocols are implemented in the form of Simulink S-Functions, not only compatible to Windows but also to the Real-Time Operating System VxWorks. SCP/RSP is tested using a demo scenario running on a Formation-Flying-Testbed. This simulation environment is a multi-satellite simulator developed by the Formation-Flying group at GSOC. It is illustrated that connection quality in the local EPOS network allows coupling an external simulation with EPOS via Ethernet, as long as the external simulator's sample frequency is not too large. Moreover, drift, e.g. the time differential between the external simulator's clock and the EPOS real-time clock, shows to be in the limits of timer hardware precision. As a result, simulations can be run for many hours before drift becomes a problem. SCP/RSP adds to the flexibility of EPOS. Before, an initial speed and angular velocity different from zero was inconvenient to realize. SCP/RSP carries out this task automatically, by determining an initial trajectory when needed. Software running on the EPOS real-time control system has to run at a frequency of $250Hz$ obligatorily. SCP/RSP allows running an external simulation at a much lower frequency by translational and rotational interpolation. And SCP/RSP simplifies the simulation process. External simulations can be started and stopped without the need to restart the EPOS real-time simulation.

1. INTRODUCTION

1.1. Overview

This paper is a compressed version of the author’s diploma thesis, which was written in the second half of 2012 at the German Space Operations Center (GSOC, part of German Aerospace Center) in Oberpfaffenhofen near Munich. The author was mentored by the Institute of Astronautics (LRT) at the Technische Universität München (TUM). The works’ central objective is the realization of an Ethernet interface (e.g. communication protocol software) for the European Proximity Operations Simulator (EPOS), which is a real-time controlled Hardware-in-the-Loop simulator for Rendezvous and Docking (RvD) scenarios (see 1.2). This interface allows for an easy and flexible connection of an external simulation system to EPOS. The Formation-Flying (FF) Testbed – a multi-satellite simulator developed at GSOC – represents an example of such an external simulator.

In the following, a description of EPOS is given and the motivation behind the development of the interface is outlined. After a brief presentation of the FF-Testbed as well as of the complete simulation configuration (EPOS + FF-Testbed), the developed communication protocol is described. Provided services and functions are explained. Thereafter, some experimental results are given concerning interface performance, followed by a brief exposition of possible extensions and future development.

1.2. European Proximity Operations Simulator

The European Proximity Operations Simulator (EPOS) is a Hardware-in-the-Loop (HiL) simulator for the last phase (25m to 0m) of Rendezvous and Docking (RvD) scenarios involving two spacecraft. While physical simulation of satellite dynamics on ground is virtually impossible, its accurate numerical simulation is much easier and leads to very good results. And whereas a realistic sensor output (camera, sensor for contact forces...) is hard to simulate numerically, real sensor hardware can easily be used on ground. EPOS profits from combining numerical models of satellite dynamics and real hardware for RvD simulations.

The central elements are two standard 6 Degree-of-Freedom (DOF) industrial robots. One of them, a KUKA KR100HA, is mounted on a linear rail, which is 25m long. The robot can be moved on that rail in order to simulate the approach of two satellites in space. Thus, the linear rail constitutes one additional degree of freedom. The other robot is a KUKA KR240 and is mounted at the end of the rail, its base fixed in the laboratory. Each robot is equipped with a breadboard attached to the tool flange which can be used to mount satellite mockups or sensor devices. Furthermore power supplies with different voltages as well as various data interfaces are available.

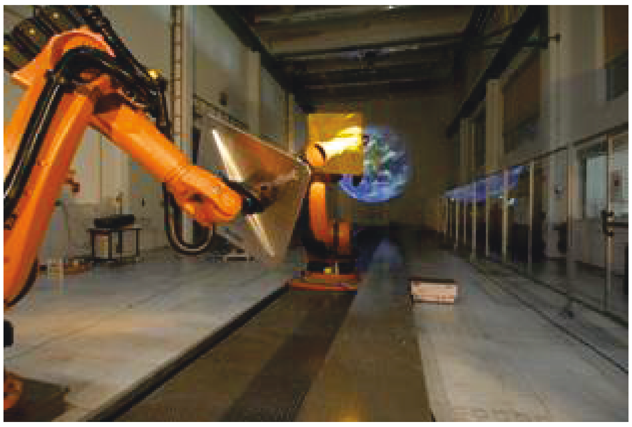
FIG 1. EPOS Facility

In a typical scenario, the mockup of the target satellite is mounted on the front panel of the KR100HA (on linear rail). The servicer satellite is then represented by the KR240 (fixed), which may carry sensor equipment and docking systems on its front panel. Angular and relative transversal movement of both satellites can then be simulated by moving the 6 DOF robots and including the linear rail as a 13th DOF.

Control of the EPOS facility and simulation of RvD scenarios is achieved using a set of computers, each fulfilling a specific task in the system and being connected via different types of networks. Facility control can be structured in three different levels (see FIG 2). The first level corresponds to the Application Control System (ACS). The second Level is associated with the Facility Monitoring and Control System (FMC). And finally the third level encompasses the Local Robot Control Units (LRC). Both the ACS and the FMC are comprised of two PCs, a Man Machine Interface (MMI) running Windows XP and a Real Time computer (RT) running Wind River VxWorks, respectively. The LRC consists of two identical KUKA Robot Controllers (KRC) which are directly connected to the particular robots. All these components are interconnected by an Ethernet Windows network via switch. To fulfill real-time requirements, ACS/RT and FMC/RT communicate through EtherCAT, where the FMC/RT serves as master and the ACS/RT as slave. For connecting both KRCs with the FMC/RT another network is used, the so called Robot Sensor Interface (RSI) from KUKA.

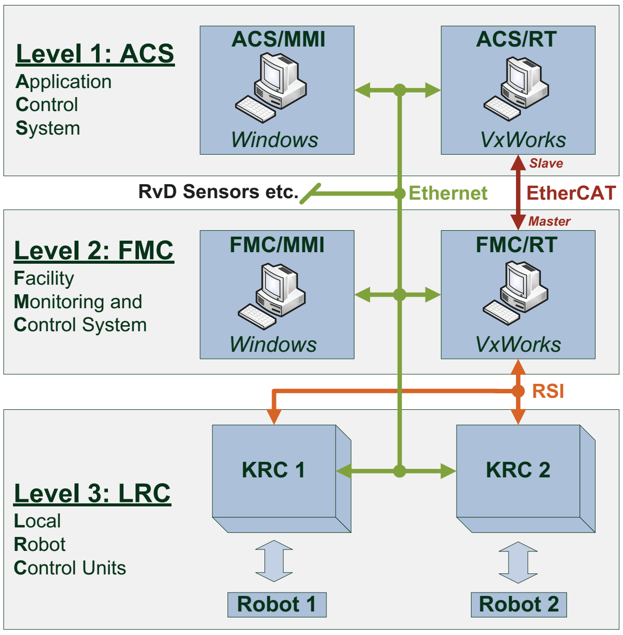

FIG 2. EPOS Control System

The ACS/MMI is used to design a Simulink model of e.g. satellite dynamics, sensor interfaces and controller algorithms – all the elements necessary to command the robots' movements in a specific RvD simulation. This model is translated to real-time code by Simulink Real-Time Workshop (RTW) and run on the ACS/RT. Movement commands are passed to FMC level, then to LRC level, before finally fed to the robots, thereby undergoing diverse processing (coordinate transformation, checks to avoid a crash etc.). The FMC/MMI is used mainly for monitoring purposes.

An Example for a HiL scenario: A camera is mounted onto the servicer robot. Real-time images of the target mockup are fed to the ACS/RT (on which the Simulink model is running) where they are processed to estimate the target's position and attitude. This data is used to have the servicer adapt its attitude and position depending on the target, thus realizing a formation-flight.

The facts presented in this section are based on [1,2,3,4,5], where more detailed information about EPOS can be found.

1.3. Expanding EPOS with an external Simulator

The following scenario is considered: Some research group has available some kind of simulator system specifically designed to support the development of a space camera, a docking mechanism or other RvD hardware. This simulator may be a real-time capable system, using an "exotic" RTOS or it may run with soft real-time conditions using WinXP. There may by hardware components (mockups) used in the loop. The simulation may have been designed with a model-based approach, for example a Simulink model, or with a low-level standard programming language like C/C++ and FORTRAN. Integrating such an external simulator directly with the EPOS real-time control system, which means integrating software and hardware interfaces into the real-time Simulink model, necessitates various problematic tasks:

- Since the ACS/RT uses the RTOS VxWorks combined with Matlab/Simulink Real-Time Workshop, the code has to be ported to this operating system (in the form of S-Functions). The compatibility with VxWorks may require major changes to the code. Furthermore, Simulink RTW may not support S-Functions written in the programming language of the external simulator. Even if Simulink RTW could be modified to do so, most likely the effort necessary would be considerable. Another possibility: The simulator uses Simulink blocks which are incompatible with RTW or it may use a totally different simulation environment (e.g. Modelica). In that case the simulation would have to be designed anew from scratch.
- As mentioned before, the simulation system could include hardware connected to it, e.g. an on-board computer. Then, interfaces which work together with VxWorks are necessary. Since I/O instructions aren't platform independent, most definitely these interfaces have to be redesigned to fulfill compatibility. Moreover, the ACS/RT has only a limited number of different types of interface connections available (Ethernet, serial port, USB without drivers at the time the diploma thesis was written). However, the simulation system may require other kinds of interfaces.
- The simulation on the ACS/RT has to run at a sample frequency of 250Hz. However, satellite flight software (and associated simulators) may run at considerably lower frequencies. From this discrepancy follow several problems. It may not be possible to make the external simulator run at that high rate due to the connected hardware components. Also, the required computation power to make the simulation work at 250Hz could exceed the ACS/RT's capabilities. It is conceivable to have the simulation run at 250Hz, whereas the code of the external simulator is executed at a lower sample frequency. But this may result in "jumps" in commanded robot positions, overstraining the robots' acceleration capabilities.
- While integrating their software, the customers would necessarily come in contact with EPOS internal software and Simulink blocks. This requires additional time and effort of the scientists and renders impossible a "black-box"-like approach which is desirable from perspective of the customer as an external EPOS user.
- The necessary procedure for realizing initial robot conditions with speed and angular rate different from zero is time-consuming and impractical. Before the actual simulation is started, the customer has to determine the initial conditions for each individual simulation run, calculate a proper initial trajectory and provide it to the EPOS FMC/MMI in the form of a file. This procedure would prolong RvD tests needlessly.

It is the motivation of the author's diploma thesis to avoid all these problems by designing a well-defined Ethernet-based interface to connect any external simulation system to EPOS. This means development and definition of a communication protocol as well as implementation of this protocol on the ACS/RT and, exemplarily, on an external simulator. Thus, the external simulation system can stay "`as it is"' and is simply connected to the EPOS Ethernet network. No development work is required, except the implementation of the communication protocol. Ethernet is an established standard so that almost any conceivable simulation system is equipped appropriately. Moreover, the external simulator doesn't necessarily have to be located in the EPOS control room. It doesn't have to be in the same building and may even be in another city, as long as the Ethernet link's latency is of reasonable magnitude. However, there is one disadvantage which comes with Ethernet. It is, by design, non-deterministic. The time some piece of information takes from source to target is of statistical nature. It will be shown to which extent this fact influences the proper functionality of the whole configuration.

As an example for an external simulator, the Formation-Flying (FF) Testbed is used, a multi-satellite simulator based on Matlab/Simulink that has been developed by the FF-group at GSOC for realizing high-precision simulations of FF algorithms. It consists of a number of different components: A Windows PC serves as Flight Control Computer (FCC) which simulates orbit and attitude dynamics, including various effects like drag and solar radiation pressure. The flight software runs on the Onboard Computer (OBC), which is a SPARC architecture embedded system. Additionally, a high-precision GNSS Signal Simulator is used to supply two Phoenix GPS receivers with real Radio Frequency (RF) signals.[5,6,7,8] FIG 3 shows how the FF-Testbed is connected to the EPOS control system.

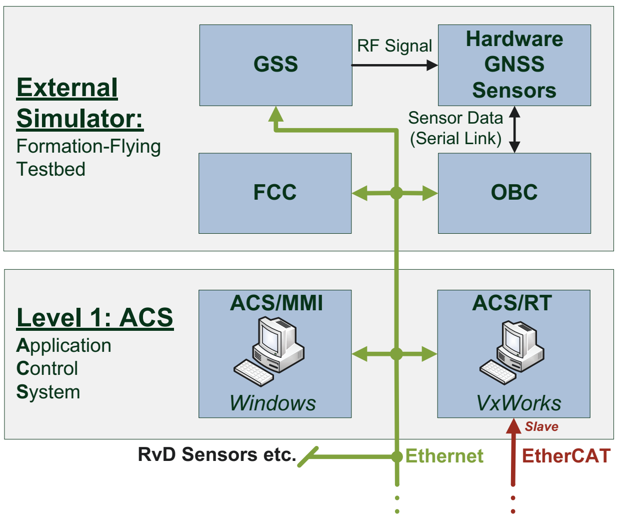

FIG 3. External Simulator connected to EPOS Control

The communication protocol is implemented in C++. The code is integrated into Simulink as c-mex S-function blocks. Thus, one S-function block (denoted “server”) is part of the Simulink model created on the ACS/MMI and executed on the ACS/RT. Another S-function block is part of the Simulink model running on the FCC of the FF-Testbed, or generally on the external simulator. The blocks establish the Ethernet connection between both Simulink models, e.g. between EPOS and the external simulator, and thus realize the communication protocol presented below. In general, the external simulator calculates spacecraft states (position, speed, attitude, angular velocity) as part of the individual simulation. These states are fed to the FCC S-function block which sends them to the EPOS s-function block. There, the states are used to have the robots move accordingly. Also, actual robot states, as measured by EPOS, are sent back to the external simulator.

2. SOFTWARE ARCHITECTURE

2.1. Two-Layer Architecture in the Context of Computer Networks

The nature of the various tasks which have to be dealt with in order to realize a connection between some external simulation and EPOS suggest a splitting into two parts or layers. Here, the author complies with the useful and widespread concept of layers in software design.[9] FIG 4 illustrates the two layers, their connection and their roles in software architecture.

The Simulation Connection Protocol (SCP) constitutes the bottom layer. The implementation of this protocol is also denoted SimCon. The Remote Simulation Protocol (RSP) constitutes the top layer. The implementation of this protocol is also denoted RemoteSim.

SCP provides a bidirectional connection of "raw" data in the form of a vector of numerical values. The format in which this data is exchanged is specified as a SimCon packet. Moreover, SCP monitors connection quality (packet latency, deadtime etc.) and signal status (link intact or broken) and provides this information to RSP. It also supplies the top layer with information concerning the correlation between EPOS' simulation time and the remote computer's simulation time. Thus, SCP can be said to manage the actual data connection and associated low-level functions.

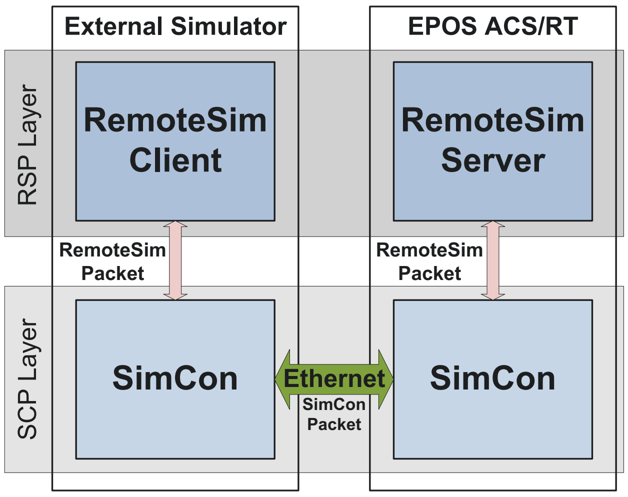

FIG 4. Two-Layer Architecture

RSP deals with the different states of the simulation process, i.e. facility initialization, realization of starting conditions, seamless transition to actual simulation etc.. It handles many other tasks, interpolation and timing being only two of them. RSP utilizes the services provided by SCP. RemoteSim communicates using RemoteSim packets which are comprised of a number of values, each having a well-defined meaning. Since a RemoteSim packet is nothing but a data vector of numerical values, SCP can be used to exchange these packets between external simulation and EPOS ACS/RT. Thus, RSP can be said to manage the interaction between remote simulation and EPOS.

As depicted in FIG 4, the external simulation comprises one instance of SimCon and one of RSP client. On the EPOS ACS/RT one instance of SCP and one of RSP server is required. Note that both are separate components not only on this abstract level but also at code level. However, both may be combined in one Simulink block in the external simulation and one Simulink block on the ACS/RT.

This partitioning of tasks and responsibilities has several advantages. SCP can be used separately as a universal data connection. It may transmit sensor data from simulation to simulation like visual information or something similar. Furthermore, RemoteSim could be combined with a different data exchange protocol/software as long as it provides the services RSP requires.

In the terms of computer networks[9], SCP and RSP comprise the application layer protocol, as illustrated by FIG 5. A RemoteSim packet is encapsulated in a SimCon packet which then constitutes a network message.

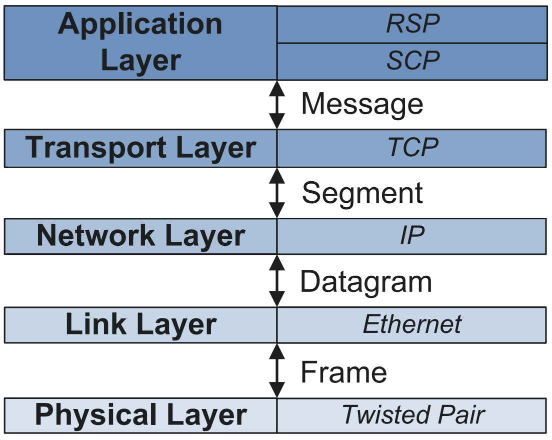

FIG 5. RSP and SCP in Protocol Stack

2.1.1. Simulation Connection Protocol (SCP)

SCP uses TCP/IP as transport- and network layer protocol. A SimCon link uses two ports for two associated connections: a data connection and a control connection. As the terms suggest, data is sent via data connection and control information is sent via control connection. Thus SCP can be said to send control information "out-of-band", similarly to the File Transfer Protocol (FTP) and the Real-Time Streaming Protocol (RTSP).[9] The reason for this is simple. Suppose a large data vector is sent. While it is transmitted, no other information can be exchanged. Using another connection specifically for control information obliterates this drawback. In a way, the multiplexing/de-multiplexing algorithm of the underlying transport layer protocol is exploited to send control information and data vectors at the same time.[9] TCP is chosen as transport layer protocol. It is preferred to UDP because of its reliable data transfer (guarantee of complete transmission in right order).[9] SimCon always establishes a point-to-point link between two SimCon instances. Information is transmitted in the form of SimCon packets with a well-defined structure and of different types. Packets of type data are sent using the data connection, all other types of packets are sent via control connection. A packet received at the wrong port is discarded.

For clarity in the subsequent discussion, one of the two communicating SCP instances is denoted “home” and the other “target”. Home is where we think ourselves located; target is the distant communication partner. This definition is arbitrary and could be reversed at will, but it simplifies explanations. SCP relies on time information provided by the Operating System (OS). In general, this time on the home OS differs from that of the target OS. This also is valid for simulation time steps, since home and target simulation are most likely not started at the same time and sample frequencies are different. Therefore, some point in time can be expressed in home time frame or in target time frame, which is hereinafter expressed by superscripts ^{H} and ^{T}. At this point, it should be noted that SCP and RSP don’t rely on any synchronous time signal between EPOS and external simulator. At both end-points the aforementioned system time supplied by the OS is used.

This design including the different types of SCP packets is the basis for the services the protocol provides:

- SCP allows for transmission of a vector of numerical values from one SCP instance to another. The data type can be chosen (floating-point double precision, signed and unsigned 8-bit integer). For this purpose, the SCP packet of type *data* is used.
- A simple ping mechanism makes it possible to detect a broken connection. In regular intervals, a SimCon instance sends a packet of type *ping*. The receiving instance responds with a packet of type *ping return*. If this respond is not received after a certain amount of time (timeout), the connection is considered broken.

SCP provides information about the correlation between the simulation time steps of the two communicating SCP instances. RSP requires this information to ensure proper timing between EPOS and external simulator. A SCP packet of type *reference*, carrying the current simulation time step, is sent at each time step. The receiving SCP instance can thus correlate the time step of the simulation it is embedded in with the one of the other SCP instance.
Assuming that the home instance receives such an SCP *reference* packet sent by the target instance, then the time step of transmission (contained in the packet) equals the reference time step expressed in target time frame $\mathbf{n}_{\mathbf{ref}}^{\mathbf{T}}$. The packet is received at the time step $\mathbf{n}_{\mathbf{recv}}^{\mathbf{H}}$. Then the reference time step expressed in home time frame is $\mathbf{n}_{\mathbf{ref}}^{\mathbf{H}} = \mathbf{n}_{\mathbf{recv}}^{\mathbf{H}} + \mathbf{1}$.
The reason for $+\mathbf{1}$ is illustrated in FIG 6: If the next simulation time step at the home instance begins, the RSP layer needs the reference time steps. However, these have been determined just before the beginning of the current time step. Suppose the home sample time is considerably larger than the target sample time, this would introduce a large error. Adding 1 corrects this.
Based on the reference time steps, any point in time in the form of a simulation time step expressed in home time frame can be transformed into target time frame:

$$\text{(1)} \quad \mathbf{n}^{\mathbf{T}} = \left(\mathbf{n}^{\mathbf{H}} - \mathbf{n}_{\mathbf{ref}}^{\mathbf{H}}\right) \cdot \frac{\Delta \mathbf{t}_{\mathbf{sample,h}}}{\Delta \mathbf{t}_{\mathbf{sample,t}}} + \mathbf{n}_{\mathbf{ref}}^{\mathbf{T}}$$

Note that this mechanism neglects the packet’s transmission times, which are small, compared to the external simulator’s sample time (about 10Hz). Note also, that rounding may be necessary with (1) to obtain a full time step.

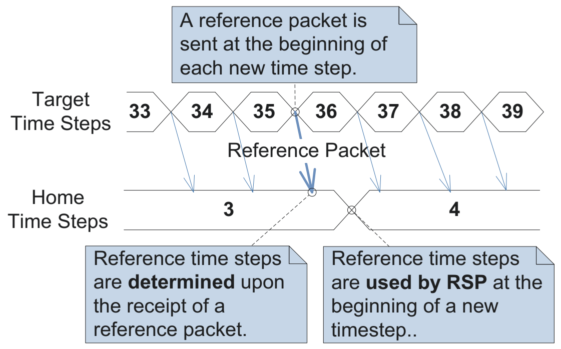
A reference packet is sent at the beginning of each new time step.
Target Time Steps
33
34
35
36
37
38
39
Reference Packet
Home Time Steps
3
4
Reference time steps are **determined** upon the receipt of a reference packet.
Reference time steps are **used by RSP** at the beginning of a new timestep..

FIG 6. Timing Diagram for Reference Packets

- Similarly to the reference steps, SCP provides a reference time $\mathbf{t}^{\mathbf{T}\rightarrow\mathbf{H}}$ which allows to transform any point in time from target to home time frame and vice versa, according to

(2) $\mathbf{t^H = t^{T\to H} + t^T}$

Determining $\mathbf{t^{T\to H}}$ is not trivial. It is not sufficient to have the target instance transmit its current time to the home instance, where the reference time is to be determined, since the transmission time is unknown and cannot be neglected (in contrast to determination of reference time steps). Instead, a SCP packet of type *delay* is sent at regular intervals. The target SCP instance responds with a packet of type *delay return*. The delay packet is sent at $\mathbf{t^H_{trm,delay}}$ and received at $\mathbf{t^T_{recv,delay}}$. The *delay return* packet is sent at $\mathbf{t^T_{trm,delay\ return}}$ and received at $\mathbf{t^H_{recv,delay\ return}}$. The *delay return* packet carries all these values in its header.

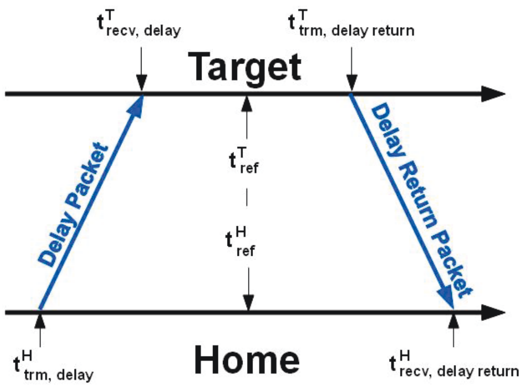

FIG 7. Calculation of Reference Time

The point in time precisely between transmission and receipt at the home instance is

(3) $\mathbf{t^H_{ref} = 0.5 \cdot \left(t^H_{recv,delay\ return} - t^H_{trm,delay}\right) + t^H_{trm,delay}}$

and the according point between transmission and receipt at the target instance

(4) $\mathbf{t^T_{ref} = 0.5 \cdot \left(t^T_{trm,delay\ return} - t^T_{recv,delay}\right) + t^T_{recv,delay}}$

Now it is assumed that it takes the same amount of time for transmission of *delay* and *delay return* packet. Then the times (3) and (4) are identical, but expressed in different time frames. With (2) it follows

(5) $\mathbf{t^{T\to H} = t^H_{ref} - t^T_{ref}}$

Note that the reference time is determined at regular intervals with each received *delay return* packet and averaged using all the calculated values up to this point.

- SCP supplies information about the transmission times associated with each received data packet. Certain header fields of SCP data packets are used to determine these transmission times. For example, a specific header field contains the time when the packet is transmitted.
  The pure transmission time, i.e. the time between transmission at $\mathbf{t^T_{trm}}$ and receipt of a data packet at $\mathbf{t^H_{recv}}$, is denoted Send2Recv:

(6) $\mathbf{\Delta t_{s2r} = t^H_{recv} - \left(t^{T\to H} + t^T_{trm}\right)}$

The time between acquisition of data by the SCP instance from the simulation at $\mathbf{t^T_{acqu}}$ and output of this very data at the other SCP instance to the simulation at $\mathbf{t^H_{out}}$ is denoted Acqu2Output:

(7) $\mathbf{\Delta t_{a2o} = t^H_{out} - \left(t^{T\to H} + t^T_{acqu}\right)}$

Acqu2Output includes processing time. Note that the time transformation $\mathbf{t^{T\to H}}$ determined by SCP and explained above is needed for calculation of Send2Recv and Acqu2Output. Precision and reliability of these transmission times strongly depend on the validity of the reference time $\mathbf{t^{T\to H}}$ and thus on the quality of the Ethernet network. However, the service of providing packet transmission times is for monitoring purposes only and is not critical for combining an external simulator with EPOS real-time control.
To explain deadtime, the following process shall be considered. Data for a packet is acquired at the home instance at time $\mathbf{t^H_{acqu}}$. This packet is transmitted to the target, where its data is provided to this simulation at some time step. During this very time step (possibly as a respond to this data) a SCP packet is sent back to the home instance, where this packet's data is supplied to the home simulation at $\mathbf{t^H_{out}}$. This time interval is denoted deadtime:

(8) $\mathbf{\Delta t_{dead} = t^H_{out} - t^H_{acqu}}$

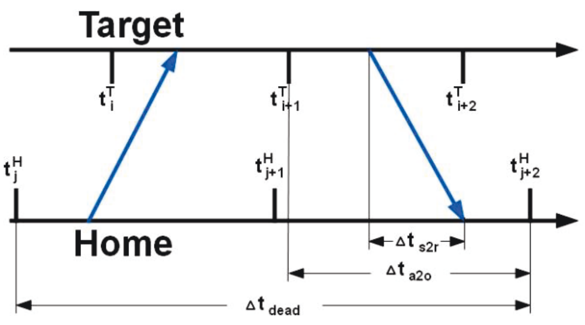

FIG 8. Transmission Times

2.1.2. Remote Simulation Protocol (RSP)

As already indicated in FIG 4, RSP consists of a RSP client and a RSP server. The former has to be part of the external simulator's Simulink model and the latter is to be included in the EPOS Simulink model (ACS/RT). Both rely on the services provided by SCP. They communicate via RSP packets, which are comprised of a set of double-precision floating-point values. These RSP packets are passed to the SCP layer, where they are sent and received as the data section of SCP packets. Together, RSP client and RSP server manage the interaction between EPOS and the external simulator. RSP server performs a variety of tasks:

- It interpolates robot trajectories in-between the commands received from the external simulator (RSP client). This is necessary since the EPOS Simulink model is executed with a sample frequency of 250Hz, while the external simulator works with a much lower frequency. Translational robot position is interpolated using a cubic polynom, e.g. C^1 continuous. Rotational robot attitude (quaternion) is interpolated using a method developed by Kim, Kim and Shin[10].

- As part of the simulation process, RSP server calculates an individual initial trajectory whenever the external simulator is started. This trajectory begins with the current facility states and ends with the desired initial conditions as transmitted by the external simulator. The methods are identical to those utilized with the aforementioned interpolation during regular simulation (after the initial trajectory is finished).
- If the external simulator is stopped or if the connection is broken, RSP server slows down the robots safely and slowly without provoking a hard stop of the robots, which would require a partial restart of the EPOS real-time control system. Thus, start, stop and restart of a simulation are reduced to starting, stopping and restarting the external simulator only.
- The server part of RSP extrapolates robot position and attitude, if commanded robot states expected to be received from the external simulator (RSP client) are delayed, e.g. are received after they are needed for the next upcoming interpolation interval. Thus, RSP can cope with occasionally delayed packets due to the non-deterministic nature of Ethernet.

The RSP client is considerably simpler than RSP server. The client performs the following tasks:

- It calculates simulation start time (when initial trajectory is finished) as well as duration of interpolation interval and transmits this timing information (along with the desired initial robot conditions) to RSP server.
- RSP client transmits desired robots states regularly and in time, so that RSP server can determine interpolated trajectories for the consecutive interpolation intervals.

It is the interaction between client and server that constitutes the RSP protocol. The subsequent description of this interaction shall illustrate the interaction principles, thereby omitting some details which would go beyond the scope of this paper. FIG 9 illustrates the main actions taken and information exchanged during a simulation, from initialization to termination of the external simulator.

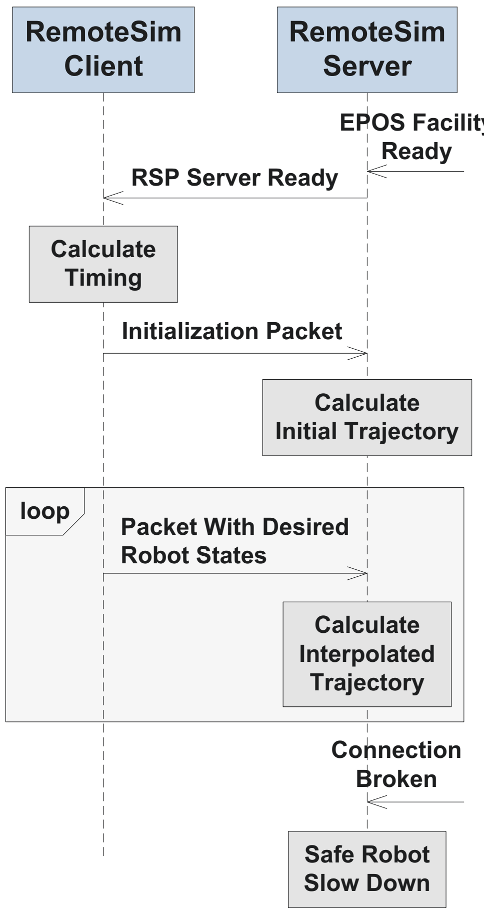

FIG 9. RSP Client/Server Interaction

It is assumed, that a connection has been established successfully. The EPOS facility undergoes some preliminary processes upon startup that are not depicted here. At some point the facility signals that it is ready. Upon receipt of this signal, RSP server sends an according message to the RSP client. Thereupon, the client gets the desired initial conditions (from the Simulink model running on the external simulator) and calculates the simulation's timing. For this timing, two values have to be calculated. RSP client needs to know the initial time step, marking the point when the initial trajectory is finished. Beginning with the step after the initial time step, RSP client will send desired robot states. The client initial time step (subscript $_c$), expressed in client time frame (superscript ^{C}) is

$$\mathbf{n}^{\mathbf{C}}_{\mathbf{init,c}} = \mathbf{ceil}\left(\frac{\Delta\mathbf{t}_{\mathbf{init}}}{\Delta\mathbf{t}_{\mathbf{sample,c}}}\right) + \mathbf{n}^{\mathbf{C}}_{\mathbf{curr}} \quad (9)$$

where $\mathbf{n}^{\mathbf{C}}_{\mathbf{curr}}$ is the current time step, e.g. the time step when the timing calculation is carried out and the initialization packet is sent to RSP server, $\Delta\mathbf{t}_{\mathbf{init}}$ is the initial time span, e.g. the time interval available to the robots to realize the starting conditions and thus the duration of the initial trajectory and $\Delta\mathbf{t}_{\mathbf{sample,c}}$ is the sample time of the Simulink model the RSP client is embedded in, e.g. the sample time of the external simulator. Note that $\Delta\mathbf{t}_{\mathbf{init}}$ must be of reasonable magnitude (30 to 60 sec) and has to be provided by the user. On the other end, RSP server must be informed about when the initial trajectory has to end and when to expect the first regular simulation command from RSP client containing desired robot states. The server initial time step (subscript $_s$), expressed in server time frame (superscript ^{S}) is

$$(10)\qquad \mathbf{n}^{\mathbf{S}}_{\mathbf{init,s}} = \mathbf{n}^{\mathbf{S}}_{\mathbf{init,c}} + \mathbf{ceil}\left([\mathbf{1}+\boldsymbol{\nu}]\cdot\frac{\Delta\mathbf{t}_{\mathbf{sample,c}}}{\Delta\mathbf{t}_{\mathbf{sample,s}}}\right)$$

$\boldsymbol{\nu}$ is the forerun, a fraction of $\Delta\mathbf{t}_{\mathbf{sample,c}}/\Delta\mathbf{t}_{\mathbf{sample,s}}$ between 0 and 1, that specifies a time span the desired robot states are sent to RSP server *earlier* than needed, so that varying packet transmission times and processing times (calculation of interpolated trajectories) are regarded. The forerun is added and thus the RSP server initial time step occurs later, implying a spare time constituted by $\boldsymbol{\nu}$. $\Delta\mathbf{t}_{\mathbf{sample,c}}$ is RSP client sample time, e.g. sample time of the external simulator. $\Delta\mathbf{t}_{\mathbf{sample,s}}$ is RSP server sample time, e.g. EPOS sample time (250Hz). $\mathbf{n}^{\mathbf{S}}_{\mathbf{init,c}}$ is the client initial time step expressed in RSP server time frame. It can be determined from (9) using the reference time steps provided by SCP (see (1)).

$$(11)\qquad \mathbf{n}^{\mathbf{S}}_{\mathbf{init,c}} = \left(\mathbf{n}^{\mathbf{C}}_{\mathbf{init,c}} - \mathbf{n}^{\mathbf{C}}_{\mathbf{ref}}\right)\cdot\frac{\Delta\mathbf{t}_{\mathbf{sample,c}}}{\Delta\mathbf{t}_{\mathbf{sample,s}}} + \mathbf{n}^{\mathbf{S}}_{\mathbf{ref}}$$

$\mathbf{n}^{\mathbf{S}}_{\mathbf{init,s}}$, along with the initial conditions is sent to RSP server in the form of an initialization RSP packet. As a next step, RSP server calculates the initial trajectory and has the robots move accordingly. Shortly before the initial trajectory is complete (considering the forerun), RSP client starts the actual external simulation and transmits the first desired robot states. Precisely when the initial trajectory is finished, RSP server uses the desired states to determine the interpolated trajectory for the upcoming interpolation interval and has the robots move accordingly. Again, shortly before this interpolation interval is finished, RSP client sends the desired robot states for the next interpolation interval. Precisely when the current interpolation interval is finished, the lately received robot states are used to determine the interpolated trajectory of the next interpolation interval and the robots are moved accordingly. In this manner, this principle is repeated during the simulation. The timing diagram shown in FIG 10 illustrates this with an example, where $\mathbf{n}^{\mathbf{S}}_{\mathbf{init,s}} = \mathbf{47}$, $\mathbf{n}^{\mathbf{C}}_{\mathbf{init,c}} = \mathbf{5}$, $\boldsymbol{\nu} = \mathbf{0.5}$ and $\Delta\mathbf{t}_{\mathbf{sample,c}}/\Delta\mathbf{t}_{\mathbf{sample,s}} = \mathbf{4}$.

At some point, the external simulator may be stopped. This is recognized by RSP server, which safely slows down the robots to a soft halt. From there, a new initial trajectory could begin, should the external simulator be restarted by the user.

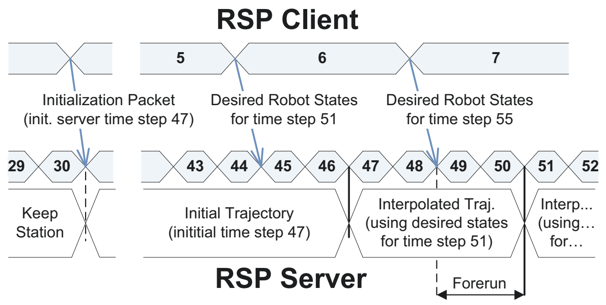
RSP Client
5
6
7
Initialization Packet (init. server time step 47)
Desired Robot States for time step 51
Desired Robot States for time step 55
29
30
43
44
45
46
47
48
49
50
51
52
Keep Station
Initial Trajectory (initial time step 47)
Interpolated Traj. (using desired states for time step 51)
Interp... (using... for...
RSP Server
Forerun

FIG 10. Timing of Client/Server Interaction

3. SIMULATION RESULTS

3.1. Test Scenario

The simulation, the analysis in this section is based on, has not been carried out using the complete configuration depicted in FIG 3. Rather, a minimum configuration was chosen to illustrate proper simulation interconnection. It is not the purpose of this section to evaluate a specific FF simulation with associated algorithms etc., but to show that SCP/RSP works as it is designed and that the principal setup - EPOS connected to an external simulator via Ethernet - is indeed capable of realizing a serious simulation. On the FF side, the Simulink model running on the FCC constitutes the complete external simulator. No other components are used. However, from perspective of the RSP client, this makes no difference.

The simulation running on the FCC (mainly) comprises the RSP client and an enabled subsystem containing the FF model. The sample frequency is $\mathbf{1Hz}$. Attitude is constant for both spacecraft.

The EPOS part of the simulation configuration is comprised of a RSP-Server block and another Simulink block constituting the interface to the robots.

3.2. Initial Trajectory

In FIG 11 target spacecraft position during the initial trajectory is depicted. To keep a good overview, only the x component is shown, representative for the trajectory. The plot comprises three curves: The requested position (requested by external simulator), the commanded position (given to the robots) and the current position (supplied by EPOS).

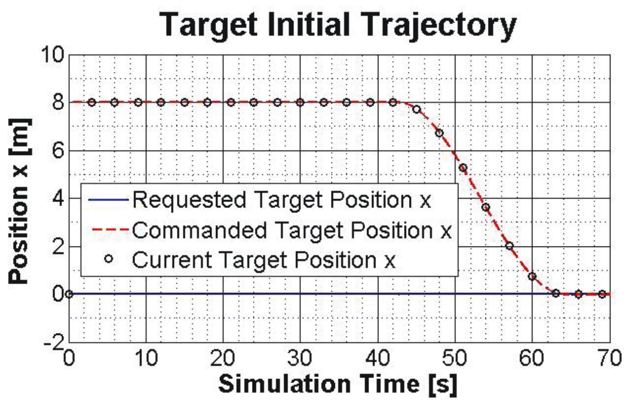

FIG 11. Initial Trajectory

The plot starts at $\mathbf{t = 0}$, e.g. when the RSP server simulation is started. Current position follows the commanded position by an additional small delay of 32ms due to EPOS control chain which cannot be seen in the figure. Requested position is $\mathbf{0}$ at the beginning of the simulation. As soon as RSP server receives the initialization packet, the target starts to follow the initial trajectory which approaches the requested position (and speed) at facility initialization smoothly and asymptotically. It is important to note that there are no edges in the trajectory.

3.3. Continuity of Interpolation

In FIG 12 target position is depicted for a small time interval during regular simulation, after the initial trajectory. Requested speed changes every second, consistent with

RemoteSim-Client sample frequency of $1\mathrm{Hz}$. The interpolated trajectory which is commanded follows the requested one precisely and smoothly. There are no peaks or undesirable curving between simulation points. That the interpolated trajectory is indeed C^1 continuous becomes clear when looking at the associated speed plot, depicted in FIG 13, where requested and current speed is compared. Speed follows the requested values. There are no jumps in velocity and therefore the first derivative of position is continuous as required.

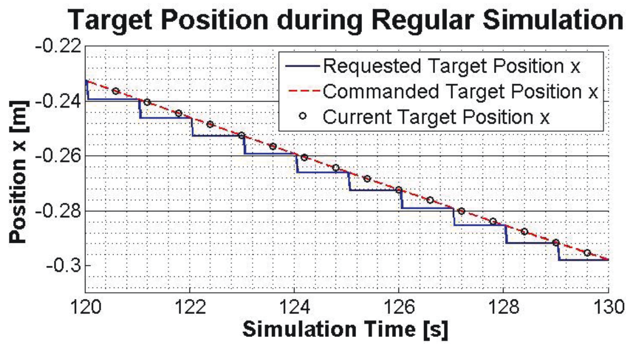

FIG 12. Interpolated Position

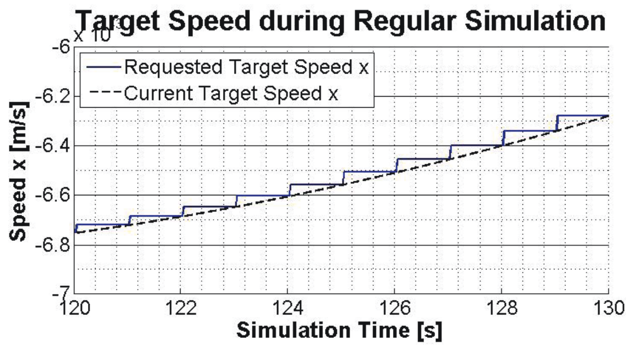

FIG 13. Interpolated Speed

3.4. Connection Quality

In this section, the demo scenario is used to present data which allows ascertain the suitability of the local EPOS network for realizing a distributed simulation via Ethernet.

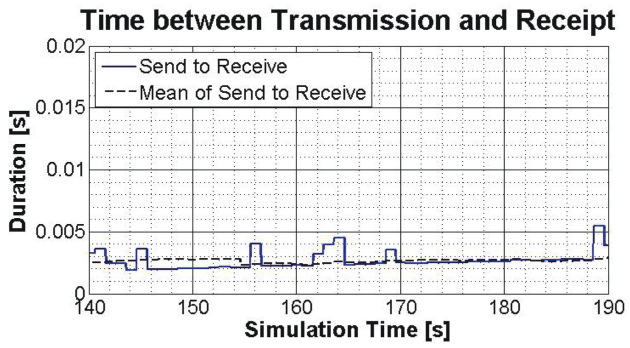

FIG 14. Send to Receive

FIG 14 shows the Send to Receive delay and its mean (calculated at each point with the last 20 values). This delay represents the time between transmission of the packet by one SCP instance (here the external simulation, e.g. RSP client) and receipt by the other SCP instance (here the EPOS simulation, e.g. RSP server). As expected, there are statistical variations. The mean value is about $2.5\mathrm{ms}$. This is even below EPOS sample time of $4ms$. Note also that changes occur at a $1\mathrm{s}$ rhythm, consistent with RSP client sample frequency of $1\mathrm{Hz}$.

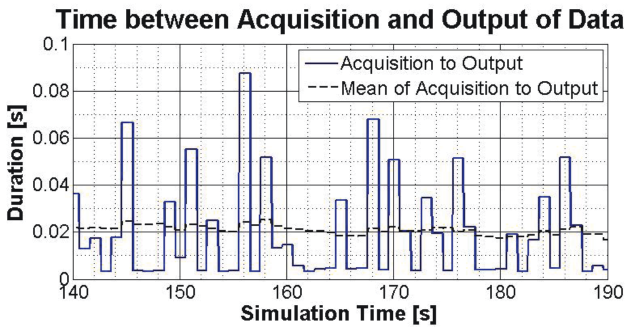

FIG 15. Acquisition to Output

Compared to Send to Receive, the delay Acquisition to Output depicted in FIG 15 is considerably larger. This delay represents the time between acquisition of data from the inputs of RSP client block and the provision of the data to the outputs of RSP server block. Not only is the mean larger, namely $20\mathrm{ms}$, but also the variation is more distinct. This can be ascribed to two main reasons. Acquistion to Output also includes the processing time of the packet by the software. Various threads are at work in parallel. Therefore, processing time may vary to some degree. Moreover, acquisition of data and output of data can only be carried out at discrete time steps. At some point, server and client time steps may be located temporally closer and at some point wider.

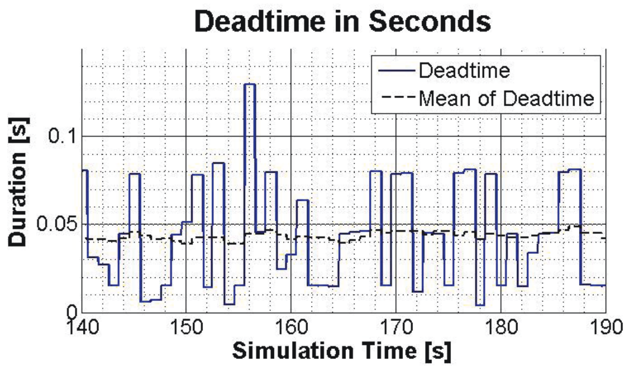

FIG 16. Deadtime

Another important delay value is deadtime depicted in FIG 16 along with its mean. It represents the time between acquisition of data by RSP server and output of the data a specific received packet contains. It is the packet RSP client has sent first after having received the aforementioned packet from RSP server. Thus, deadtime is a measure for the delay between a general command of some sort and receipt of some kind of feedback information related to this command. It is the deadtime the connection introduces in the simulation system. The mean deadtime is about $0.04s$ which equals about twice the mean of Acquisition to Output. This makes sense from a logical point of view, since two times the process of acquisition and output of data is involved.

In order to appraise the impact of the presented data on the simulation process, consider the range of reasonable sample frequencies for the external simulator. The FF-Testbed works with a frequency of at most $\mathbf{10Hz}$. Other external simulations running flight software will work in that area of sample frequency, too. Hence, the delay times in this section, especially a mean Acquisiton to Output delay of $\mathbf{20ms}$, are distinctly below critical values for the simulation system. Individual peaks can be dealt with by extrapolation (which RSP is capable of).

3.5. Drift

Drift is the time differential between the external simulation and the EPOS simulation. It is different from zero if the clocks the simulations are synchronized with don't run precisely with equal speed. The tolerance of electronic components alone dictates that a drift is to be expected. With RSP, a measure for drift is the change in actual forerun. The actual forerun showed to be about $\mathbf{10^{-6}s/s}$. This means that after $\mathbf{1000000s}$ a drift of $\mathbf{1s}$ can be measured. This is well within the order of magnitude of a quartz crystal's precision. Since SCP/RSP depends on the clock, drift cannot be reduced any further.

4. CONCLUSION AND PROSPECT

A software package has been developed that allows connecting an external satellite simulator via Ethernet with the EPOS real-time control system, thereby reducing the adaption effort for external customers and introducing new functionalities. The distribution of tasks and responsibilities led to the design of two application layer protocols. The Simulation Connection Protocol (SCP) provides the data connection, the Remote Simulation Protocol (RSP) manages simulation interaction, where RSP relies on the services of SCP. Both communication protocols are implemented as Simulink c-mex S-function blocks.

A demo scenario was simulated. Interpolation even tested with a sample frequency of $\mathbf{1Hz}$, e.g. interpolation intervals of $\mathbf{1s}$, turned out to be absolutely smooth and continuous. External simulators running at even such low frequencies can work together with EPOS. Analysis of connection quality showed a mean delay of $\mathbf{20ms}$ including processing from data acquisition at the remote simulation to data output at EPOS. Considering a maximum sample frequency of $\mathbf{10Hz}$, this delay is far below critical values. Drift between the FF-Testbed and the EPOS ACS/RT was within the limits of standard mainboard timer precision ($\mathbf{10^{-6}s/s}$). As a result, several hours of continuous simulation are possible before drift becomes a problem. The software cannot reduce drift any further, with the given boundary conditions.

In the future, it might be beneficial to extend SCP/RSP with a synchronization mechanism via Ethernet. A special packet could be used to serve as a synchronization signal. EPOS would provide the reference time. This would obliterate the drift problem altogether and allow even longer simulations. Another interesting idea is to not only use the EPOS local network but to place the external simulation in another building or even farther away. Thorough examinations would be necessary concerning packet delay and suitability of the connection for real-time simulation (with an appropriately low sample frequency).

In summary, additional functionalities have been added to EPOS with SCP/RSP developed in the author’s diploma thesis. Up till now, a simulation running directly on the EPOS real-time computer was restricted to an initial speed and initial angular velocity of zero for the robots/spacecraft. Now, using an external simulation connected via Ethernet, both initial speed and initial angular velocity can be realized in the form of an automatically calculated initial trajectory. Moreover, it is possible to restart the external simulation or even a different external simulation multiple times without stopping the EPOS real-time simulation. The robots are braked automatically, so that another initial trajectory can be calculated as a starting point for another simulation. Also, the tough restriction of a sample frequency of $\mathbf{250Hz}$ is circumvented by interpolation and extrapolation, if required.

In this way, SCP/RSP contributes to the flexibility and operational capabilities of the Hardware-in-the-Loop Rendezvous and Docking simulator EPOS.

5. ACKNOWLEDGEMENTS

My diploma thesis which this paper presents in a compressed form would not have been possible without the kindly assistance of a variety of people. I would like to thank my colleagues at the German Space Operations Center: Dr. Toralf Boge, head of the EPOS group, Heike Benninghoff and Tristan Tzschichholz. At the Institute of Astronautics (Technische Universität München), I have to thank Prof. Dr. Ulrich Walter, head of the institute, and last but not least Markus Pietras.

6. REFERENCES

[1] Boge T., Wimmer T., Ma O., Zebenay M., EPOS - A Robotics-Based Hardware-in-the-Loop Simulator for Simulating Satellite RvD Operations, The 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Sapporo, Japan, 2010

[2] Boge T., Wimmer T., Ma O., Tzschicholz T., EPOS - Using Robotics for RvD Simulation of On-Orbit Servicing Missions, AIAA Modeling and Simulation Technologies Conference, Toronto, Canada, 2010

[3] Tzschicholz T., Boge T., GNC Systems Development in Conjunction with a RVD Hardware-in-the-loop Simulator, 4th International Conference on Astrodynamics Tools and Techniques, Madrid, 2010

[4] Boge T., Rupp Th., Landzettel K., Wimmer T., Mietner Ch., Bosse J., Thaler B., Hardware in the Loop Simulator für Rendezvous und Docking Manöver, DGLR Jahrestagung, Aachen, 2009

[5] Gaias G., D'Amico S., Boge T., Hardware-in-the-loop Multi-satellite Simulator for Proximity Operations, 11th Int. WS on Simulation & EGSE facilities for Space Programmes; SESP 2010, Noordwijk, Netherlands, 2010

[6] D'Amico S., Autonomous formation flying in low earth orbit, PhD thesis, Technical University of Delft, 2010

[7] D'Amico S., Larsson R., Navigation and Control of the PRISMA formation: In-Orbit Experience, 18th IFAC World Congress, Milano, Italy, 2011

[8] Montenbruck O., Nortier B., Mostert S., A Miniature GPS Receiver for Precise Orbit Determination of the SUNSAT2004 Micro-Satellite, ION Natinal Technical Meeting, San Diego, California, 26-28 Januar 2004

[9] James F. Kurose, Keith W. Ross, Computer Networking - A Top-Down Approach, Fourth Edition, Pearson International Edition, 2008

[10] Kim, Kim, Shin, A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives, SIGGRAPH Proceeding 1995, p.369-376