Simulation-based Development of Embedded Sensor Fusion Applications

Wilfried Elmenreich
Vienna University of Technology
Vienna, Austria
wil@vmars.tuwien.ac.at

Martin Schlager
Vienna University of Technology
Vienna, Austria
smartin@vmars.tuwien.ac.at

***Abstract* – This paper proposes a methodology for the design, testing, and fine-tuning of sensor fusion applications for distributed embedded sensor networks.**
The presented approach involves the following steps: (i) gathering data about the process environment, (ii) modeling of the process and the sensors to be used, (iii) selection of sensor fusion functions to process the sensor measurements, (iv) simulated closed-loop hardware-in-the-loop testing, (v) validation by open-loop testing in the real process, and, finally, (vi) application of the developed embedded sensor fusion application in the system. Particular steps may be repeated in a loop, if the result of a given step is not satisfactory.
The main contribution of the approach is a separation of the modeling of process environment, sensors, and sensor processing. This allows for a strategic refinement of the model and supports the reuse and change of parts, e. g., in case of a sensor upgrade.
The application of the approach is shown by a MATLAB/Simulink model that incorporates block diagrams describing process environment, sensor behavior, and sensor fusion algorithms.

I. INTRODUCTION

There are several reasons for breaking up a control application into distinct parts. The horizontal partitioning enables the distribution of tasks for the sake of fault tolerance, parallelism, maintainability, and spatial distribution. Additionally, a vertical layering supports complexity reduction and reusability of parts of the model in a modified or different application.
Embedded control applications that employ a lot of sensors require both kinds of splitting up. Partitioning is required to manage the sensors and actuators in the system. While systems that employed only a small number of sensors and actuators typically were built around a central processing unit, modern distributed sensor/actuator systems use a standardized communication interface to interconnect distributed smart transducers, i. e., an integration of a sensor/actuator with a signal processing unit and a communication interface [1].
Layering is necessary to support an understandable and extendable modeling of the system. For example, in standard control theory, the controlled system subsumes the process environment and the sensor behavior. A change in the sensor configuration may invalidate a control program tailored to the original configuration.
There exist already several approaches that support the modeling of sensor-control applications. For example, the JDL Fusion Architecture [2] provides an information-centered model that has been used in military and civil applications. However, the model suffers from lacking support of partitioning and reuse and is very abstract when it comes to implementation. The waterfall model [3] provides a detailed model on the sequence of low-level fusion processes, however does not take the control feedback into account. The LAAS architecture [4] was developed as an integrated architecture for the design and implementation of mobile robots with respect to real-time and code reuse. While the architecture provides a good means for the partitioning of large systems into modules, it does not provide an appropriate real-time communication and representation of the fused data at the levels above the functional level. In contrast to the JDL model, the LAAS architecture guides a designer well in implementing reusable modules as part of a real-time application. Furthermore, approaches exist that focus on the cyclic structure of the data processing, such as the Boyd model [5] or the Omnibus model [6]. The latter provides a sophisticated hierarchical separation of the sensor fusion tasks. However, neither the Boyd model nor the Omnibus model explicitly support a horizontal partitioning into tasks that reflect distributed sensing and data processing. Thus, these models do not assist for a decomposition into modules that can be separately implemented, tested, and reused for different applications.
The Time-Triggered Sensor Fusion Model, proposed in [7], was designed to support partitioning as well as layering for modeling feedback control applications. A global synchronized time supports the interpretation and further processing of sensor data in hard real-time systems.
The objective of this paper is to present a methodology for the design, testing, and fine-tuning of sensor fusion applications based on the Time-Triggered Sensor Fusion Model.
The remainder of the paper is organized as follows: Section II. gives an introduction to the overall model, while section III. discusses the stepwise construction of a distributed embedded sensor network. Section IV. presents a case study based on a Matlab/Simulink model of a multi-sensor control application. Section V. contains a discussion and concludes the paper.

II. GENERIC PROCESS MODEL

The overall process is separated into a *real-time computer system* part and a *process environment* part. While the properties of the real-time computer system are within the scope of the system designer, the behavior of the process environ-

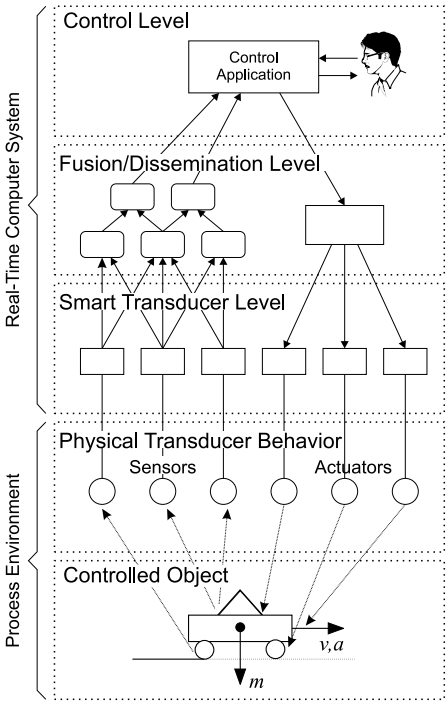

Fig. 1: Generic Process Model

ment is given as part of the problem statement.
Figure 1 depicts a breakdown of the process environment and real-time computer system part. The behavior of the process environment results, on the one hand, from the physical system that has to be controlled, and on the other hand, from the behavior of the physical sensors and actuators.

A. Process Model

When modeling the behavior of the process environment, one has to consider the properties of the *controlled object* as well as the physical behavior of the *transducers*. Both parts can possess different sources of uncertainty. The system uncertainty describes an inexactness in the system model, e. g., quantum effects that are not considered in a physical model. The transducer uncertainty stems from the impossibility to perform exact measurements or to execute a control action without some looseness in time and value domain.
The Kalman Filter [8] is an example for such a bipartite model of the process environment. It is based on a linear system model that precisely distinguishes between imprecision in the model of the controlled system and imprecision in the measurement.

B. Real-Time Computer System

The real-time computer system is designed according to the Time-Triggered Sensor Fusion Model [7]. It identifies three levels of data processing:
The first level is the smart transducer level and consists of nodes with transducers (sensors or actuators) equipped with a smart transducer interface that exports *observations*, i. e., measurements and their context, to the next level.
The second level is the fusion/dissemination level that gathers measurements, performs sensor fusion respectively distributes control information to the actuators. The task of sensor fusion is to use the sensor observations in combination with *a priori* knowledge about the system model in order to get inference about the state of the system model. Plainly spoken, the sensor fusion has to overcome the imprecision in measurement. As a consequence, the data structure of the interface to the control level should be independent of the number and type of employed sensors. The interface between fusion/dissemination level and control level is also called *environmental image* since it maps the relevant properties of the process environment.
At the top level, a control program makes control decisions based on environmental information provided by the fusion/dissemination level. User interaction, if necessary, is also handled at this level.

III. SYNTHESIS STEPS

A. Gathering Information about the Process

The first step can have a very difficult implementation. If the system model is already well-known (as it is often the case in academic examples) this step is accomplished *a priori* and one can proceed to formalize the process model.
In real-world applications, usually it is necessary to perform extensive tests and measurements in order to acquire sufficient information about the nature of the environment to be controlled.
In linear control systems, for example, the behavior of the controlled system can be derived from results of a step function input. Since, in theory, a step function tests the whole frequency spectrum and the system behavior is assumed to be linear, the resulting function can be analyzed in order to derive the parameters of the system.
For an arbitrary system, such an approach is not always possible so that it is necessary to collect and analyze a large set of data. Since the control application anyway requires a network of sensors (or at least one sensor), it is often possible to use the same hardware architecture for system analysis and system control.

B. Modeling of Process and Physical Transducers

The behavior of the employed sensors also has to be described formally in order to construct the process environment model. Nominal data about sensors, such as range, accuracy, or conversion time are available in the sensor's data sheet. However, for critical applications, sensors often have to be tuned and calibrated, i. e., the correction of sensor reading and physical outputs so that they exhibit the required behavior [9].

C. Implementation of Sensor Fusion Functions

The selection of the sensor fusion functions that process the sensor readings depends on the following properties:

Process Environment Model: A sensor fusion algorithm is always based on a particular model of the environment. Some algorithms explicitly specify their model, others do not, but nevertheless make implicit assumptions about the environment. For example, simple averaging assumes, that the measured property comes from a continuous value domain and that the quality of a value between two values is at least as good as the quality of one of the given values.

Robustness Requirements: Usually there is a tradeoff between the achievable performance and the robustness of a result from sensor fusion [10]. For example, a fault-tolerant algorithm typically depends on the exclusion of extreme values from the calculation, while the performance of sensor fusion increases with the number of data sources. This tradeoff is equivalent to the well-known worst-case versus average-case optimization.

Resource Requirements: Some algorithms such as the Kalman Filter require a considerable number of matrix operations on floating point numbers. In embedded systems without a hardware floating point unit such calculations are very costly in terms of code size and performance while fixed-point arithmetic is probably too inexact. As another example, image registration algorithms require processing power beyond the scope of many embedded microcontrollers. Thus, often a compromise between achievable quality and available resources has to be taken.

Real-Time: Real-time does not mean fast, it means *on time*. Therefore, an algorithm often is required to guarantee its timeliness for the whole space of possible input patterns. Research on Worst-Case Execution Time analysis has shown that many programs have a differing execution time depending on the input data and the hardware platform.

Determinism: Deterministic behavior is required in many applications for the sake of redundant computation or error tracing. If an algorithm is deterministic, it produces a foreseeable result for a given input data. Most sensor fusion algorithms, such as the Kalman Filter calculate their result influenced by internal state variables that contain information from earlier measurements. If this state is not made explicit, these algorithms are not deterministic and therefore could jeopardize a reintegration strategy of a distributed system.

D. Hardware-in-the-Loop (HIL) Testing

When the real-time computer system has been implemented on the target platform, it is required to perform tests before using it with the target system since direct testing in the target system might be harmful if there is an error in the implementation. Testing the real-time computer system by feeding it artificial inputs is sumptuous and probably not meaningful for complex systems.

A sophisticated approach for conducting meaningful tests without risk is hardware-in-the-loop (HIL) testing. In doing so, the input and output interfaces of the embedded real-time computer system are connected to a simulation host that simulates the environment behavior while protocolling the behavior of the real-time computer system. Thus, it is possible to test the system under regular and even extreme situations, which would not be possible to generate in reality without endangering man or machine.

HIL testing interfaces the real-time computer system at the Smart Transducer level, i. e., HIL testing emulates the behavior of a subset of smart Transducers.

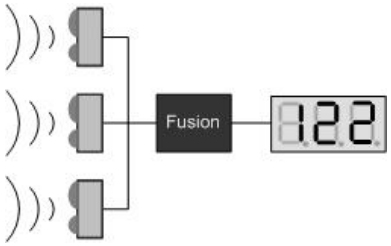

Fig. 2: Distance measurement system

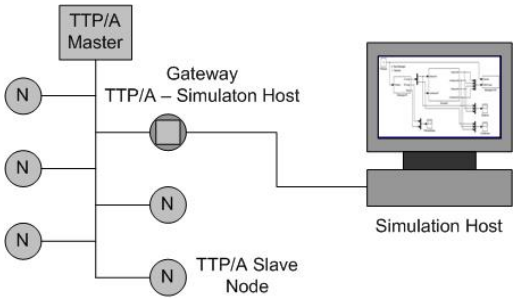

Fig. 3: Set-up of simulation architecture

E. Synthesis of the System

In the last step, the real-time computer system will be connected to smart transducers, thus feeding the controlling system with real data and putting its output to the actuators.

In a perfect world, the control application will perform well as predicted, however, there are several scenarios where the result would deviate from the predicted behavior. Probably in most cases, the system acts well, and only some situations lead to unindented behavior. These rare scenarios need to be investigated, since most likely they contain the information leading to the detection of implementation or modeling errors. In the latter case, it might be necessary to restart the synthesis from step 2 as described in Section B.

IV. CASE STUDY

In recent years, a significant number of cars have been equipped with rear distance measurement systems in order to aid the driver when backing into a parking space. We decided to develop a prototype of such a system, comprising three infra-red (IR) distance sensors, a display, and an appropriate sensor fusion controller (refer to figure 2).

According to the previously introduced synthesis steps we identified the process environment, modeled the sensor fusion algorithm and the sensors, parameterized the sensor fusion algorithm, and carried out several hardware-in-the-loop (HIL) simulation runs. Figure 3 depicts an overview of the applied Simulation Architecture, consisting of a TTP/A fieldbus network, a simulation host, running Linux as operating system, and a gateway node [11].

TTP/A is a low-cost field-bus protocol that is harmonized with the fault-tolerant system bus TTP/C [12] of the time-triggered architecture (TTA) [13]. TTP/A is intended for the connection of smart sensors and actuators in embedded real-time systems in different application domains, e.g., industrial, automotive, etc. It is the objective of TTP/A to provide all services needed by a smart sensor, including timely communication, remote on-line diagnostics and plug-and-

play capability. The main advantages of the TTP/A protocol are its deterministic behavior, resource efficiency, and its support for monitoring and configuration. TTP/A is internationally standardized by the Object Management Group as part of the Smart Transducers Interface Standard [14].
The following subsections will show how the concepts presented in this paper are reflected by the case study.

A. Process Environment

The distance measurement system will be embodied in a car. Thus, the confidence of distance values received from the sensors will depend on their conformance with the physical behavior of a car. For instance, a vehicle is bound to some maximum acceleration value (positive or negative).
In general, simulation of the process environment is a trade-off between precision and complexity of the simulation model. In the case study set-up we decided to consider a basic time continuous system with state variables *(t, v, s, a)* reflecting time, velocity, space, and acceleration.
Disruptive factors of the process environment like vibration, temperature, or ambient light have been neglected in the case study. Furthermore, the model has been discretized in order to allow stepwise hardware-in-the-loop simulation. Equation 1 shows the underlying basic formulas to model the physical behavior of the process environment (movement of the vehicle). In equations 2 and 3 the respective discrete integrations are given.

$$v = \lim_{\Delta t \to 0} \frac{\Delta s}{\Delta t} = \frac{ds}{dt} \quad a = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt} \tag{1}$$

$$v_k = \frac{s_k - s_{k-1}}{t_k - t_{k-1}} \quad a_k = \frac{v_k - v_{k-1}}{t_k - t_{k-1}} \tag{2}$$

$$s_k = s_{k-1} + v_k(t_k - t_{k-1}) \quad v_k = v_{k-1} + a_k(t_k - t_{k-1}) \tag{3}$$

The movement of the vehicle can be modeled in different ways. For instance, the environment simulator could calculate the distance of a car to its rear counterpart along a well defined path. At each point in time (t), values for v, s, and a are calculated by the environment simulator.
We decided for an alternative solution by using the values of one IR sensor. Thus, the measurement values of an IR sensor together with a confidence value are sent to the environment simulator. The confidence value, an integer within the interval of $[1,12]$, is calculated at the sensor node and gives the sensor's level of trustworthiness. The environment simulator fuses several distance values and calculates the position of the vehicle according to the physical model above.
The environment simulation has been modeled with MATLAB/Simulink[1] a graphical environment for block-based simulation development. Figure 4 shows the internal view of the main simulation block which consists of two more blocks: *System Model* and *Measurement Model*. The System Model block (refer to figure 5) consists of two further blocks: *Weighted Average Dist* and *Vehicle Dynamics*.

[1]MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

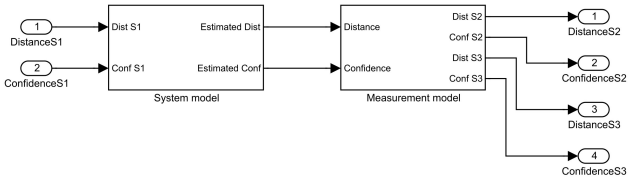

Fig. 4: Simulation block

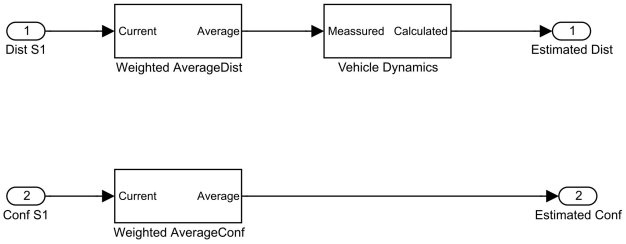

Fig. 5: System Model block

At each simulation step, the first block which is called *Weighted Average Dist* processes the last five sensor measurements of the single IR sensor and calculates a weighted average in order to provide the second block with smoothed values. The second block called *Vehicle Dynamics* is provided with current (smoothed) sensor measurements and calculates an approximation of the current position by combining the knowledge of physical behavior of the car and the sensor measurements. Figure 6 depicts the internal view (implementation) of the Vehicle Dynamics block.

B. Process and Sensors

After implementing the process environment (the vehicle and its movement) the process itself and simulation models for the emulated sensors have to be considered. Thus, the design of the distance measurement application (in particular the communication schedule) is taken as a guideline for the implementation of the process. The communication schedule defines at which points in time, sensor measurements and confidence values are provided by the sensor nodes. The same order and timeliness provision of simulation values has to be kept when emulating several nodes.
To model the process, we identified the following parts within the case study application: (1) Distance measurements are taken by three sensors at nearly the same time. (2) Each sensor calculates a value that defines the confidence in the distance measurement. (3) All sensor and confidence values are transmitted to a fusion node. (4) The fusion node processes the values, calculates an improved distance value and sends this value to a display node. (5) The distance value is displayed on a seven-segment display.
The emulation of two IR sensors of the distance measurement system affects parts (1) - (3) and has to be reflected by the process simulation. Parts (4) and (5) are implemented on the TTP/A cluster and do not have to be modeled. In the case study we used the predicted (distance, confidence) pair of the system model block as input to a *Measurement Model* block. The Measurement Model block adds normally distributed random noise to the simulated IR sensor measurements.
Figure 7 depicts the internal view of a sensor measurement

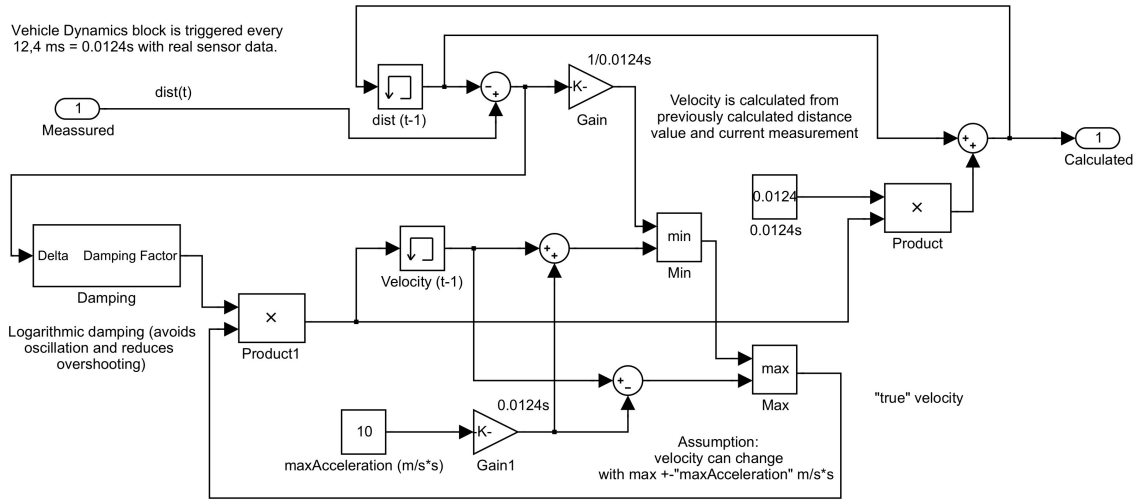

Fig. 6: Vehicle dynamics block

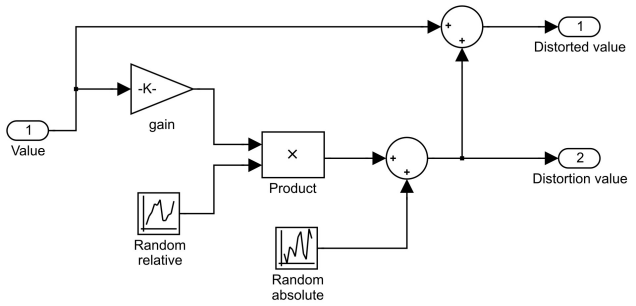

Fig. 7: Distortion block

model. In accordance to measurements with a real IR sensor, we assumed that the precision of an IR sensor is composed in two parts. A random disturbance value of the absolute part does not depend on the current distance while a random disturbance value of the relative part gets bigger when the distance increases. The distortion value, i. e., the disturbance factor that is added to the input value, is taken as an additional output value in order to calculate the confidence value of the simulated distance value. Distortion and confidence are indirectly proportional, i. e., the confidence value decreases with decreasing distortion.

C. Sensor Fusion

We decided to compare a confidence-weighted average algorithm and a sensor selection algorithm for fusing the sensor measurements [10]. The formula for the confidence-weighted average algorithm is given by

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i \cdot \frac{1}{\mathbb{V}[S_i]}}{\sum_{i=1}^{n} \frac{1}{\mathbb{V}[S_i]}} \tag{4}$$

where n is the number of sensor measurements (three in our case), x_i represents the measured values, and $\mathbb{V}[S_i]$ is the estimated variance for measurement S_i. The values of the variance are derived from the confidence values of the sensor measurements.
In order to add fault tolerance capability, it is also possible to eliminate the biggest and the smallest sensor measurement and accept only the median value (sensor selection).
We compared several approaches by applying the same sensor input data to the confidence weighted average algorithm and to the sensor selection algorithm. The results (fused distance measurements) have been compared with the real distance between sensor and obstacle.
With our test runs, the confidence weighted average algorithm generally delivered better results, especially in case of fast movement of the obstacle. Besides better performance of the confidence-weighted average algorithm in case of non-faulty sensors, this is due to the environment simulation process that does not allow fast changes of the simulated IR sensor measurements. Thus, for a short period the real IR sensor is marked faulty because both simulated IR sensors deliver values that lie close together but are worse then the measurement of the real sensor.

D. Hardware-in-the-Loop

The set-up of the case study enabled us to perform several hardware-in-the-loop tests. According to figure 2, our system consists of three IR sensor nodes, a fusion node, and a display node. The fusion application has been executed on the TTP/A master node (an Atmel AT90S8515 microchip), the display is controlled by a TTP/A slave node (an Atmel AT90S4433 microchip). One of the three IR sensors physically exists (an Atmel AT90S4433 microchip together with a Sharp GP2D12 IR sensor). The other IR sensors are simulated by the simulation host, an i686 machine running Linux (kernel 2.6.2).
The simulation host is connected to the TTP/A cluster via a RS232 interface. An Atmel ATMega128 microchip serves as a simulation gateway node, i. e., it establishes a connection to the TTP/A cluster via its TTP/A interface and to the simulation host via its UART (refer to figure 3).
The simulation host runs MATLAB/Simulink which allows us to set up the environment simulation together with the simulation of the two emulated IR sensors in the Simulink environment. In subsections A. and B. several parts of the process environment simulations have already been presented. After designing the simulator parts we performed measurements in order to parameterize our implementation.

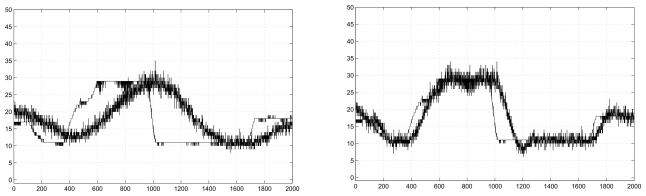

Fig. 8: Simulated distances: max acceleration 2 $\frac{m}{s^2}$ and 10 $\frac{m}{s^2}$

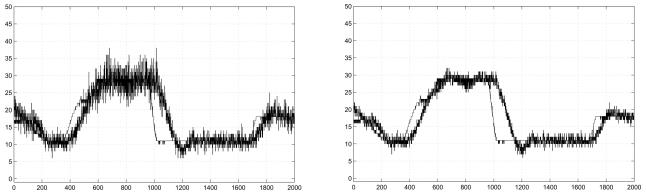

Fig. 9: Simulated disturbances: relative and absolute

Figure 8 depicts a scenario, where the environment simulator processed the same input data with different values for maximum acceleration of the vehicle. The small line which shows fast changes of the distance values represents the actual measurements of the physically existing IR sensor. The other line shows the distance values of one simulated IR sensor which slowly follows the physical sensor (according to the vehicle model). The maximum acceleration for the vehicle model has been 2 $\frac{m}{s^2}$, and 10 $\frac{m}{s^2}$ respectively. We can see that the thick line follows the smaller line faster when faster acceleration is allowed.

Furthermore, we performed several test runs in order to show the differences between relative and absolute disturbances of the measurements. We assumed gaussian normally distributed random disturbances. Figure 9 depicts two scenarios: In the left plot, the disturbance depends on the distance value (larger distance values imply greater disturbances), while in the right plot, the disturbance is absolute (and does not depend on the distance value).

V. CONCLUSION

In this paper we have proposed a generic process model and a methodology for a stepwise synthesis of a sensor fusion system.

The generic process model distinguishes between the controlling real-time computer system and the process environment. The real-time computer system is separated into a smart transducer level, a fusion/dissemination level, and a control level. The process environment consists of the controlled object and the physical aspects of sensors and actuators. This approach breaks down a control application into logically different parts and supports the reuse of application parts, e. g., the description of a sensor's behavior can be reused when the sensor is employed in another application. Furthermore, this model establishes the basic frame for a stepwise construction of a control system.

First, data about the process environment is analyzed and used for modeling the controlled object and the transducer behavior. Next, the functionality of the fusion/dissemination level is implemented based on knowledge about environment and sensors. This task may vary from a parametrization of generic fusion algorithms up to a customized approach, which is more elaborate but also potentially more effective. At this stage, a simulated closed-loop hardware-in-the-loop testing is used to validate the approach taken for the fusion/dissemination implementation. The next step is an open-loop testing in the real process before, finally, the application of the developed embedded sensor fusion is employed in the system. Particular steps may be repeated in a loop, if the result of a given step is not satisfactory.

The application of the approach has been shown by a MATLAB/Simulink model that incorporates block diagrams describing process environment, sensor behavior, and sensor fusion algorithms.

VI. ACKNOWLEDGMENTS

We would like to thank our colleagues Christian Trödhandl, Klaus Steinhammer, Ingomar Wenzel, and the anonymous reviewers for their comments on earlier versions of this paper. This work has been supported in part by DOC [DOKTORANDENPROGRAMM DER ÖSTERREICHISCHEN AKADEMIE DER WISSENSCHAFTEN], by the European IST project DECOS under contract No. IST-511764, and by the Networks of Excellence European IST project ARTIST under contract No. IST-2001-34820.

VII. REFERENCES

[1] W. Elmenreich and S. Pitzek. Smart transducers – principles, communications, and configuration. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems (INES)*, volume 2, pages 510–515, Assuit – Luxor, Egypt, March 2003.

[2] E. Waltz and J. Llinas. *Multisensor Data Fusion*. Artech House, Norwood, Massachusetts, 1990.

[3] M. Markin, C. Harris, M. Bernhardt, J. Austin, M. Bedworth, P. Greenway, R. Johnston, A. Little, and D. Lowe. Technology foresight on data fusion and data processing. Publication, The Royal Aeronautical Society, 1997.

[4] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[5] J. R. Boyd. A discourse on winning and losing. Unpublished set of briefing slides, Air University Library, Maxwell AFB, AL, USA, May 1987.

[6] M. D. Bedworth and J. O'Brien. The omnibus model: A new architecture for data fusion? In *Proceedings of the 2nd International Conference on Information Fusion (FUSION'99)*, Helsinki, Finnland, July 1999.

[7] W. Elmenreich and S. Pitzek. The time-triggered sensor fusion model. In *Proceedings of the 5th IEEE International Conference on Intelligent Engineering Systems*, pages 297–300, Helsinki–Stockholm–Helsinki, Finland, September 2001.

[8] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME, Series*

D, Journal of Basic Engineering, 82:35–45, March 1960.

[9] J. Berge. *Fieldbuses for Process Control: Engineering, Operation, and Maintenance*. ISA-The Instrumentation, Systems, and Automation Society, 2002.

[10] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[11] M. Schlager. A simulation architecture for time-triggered transducer networks. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 39–49, Vienna, Austria, June 2003.

[12] TTA Group. Specification of the Time-Triggered Protocol TTP/C. Technical report, TTA Group, Schönbrunner Straße 7, A-1040 Vienna, Austria, July 2002. Specification edition 1.0.0, Available at http://www.ttpforum.org.

[13] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 – 126, January 2003.

[14] Object Management Group (OMG). *Smart Transducers Interface Final Adopted Specification*, August 2002. Available at http://www.omg.org as document ptc/2002-10-02.