Taking Control: Modular and Adaptive Robotics Process Control Systems

Peter Ulbrich*, Florian Franzmann*, Christian Harkort†, Martin Hoffmann*,
Tobias Klaus*, Anja Rebhan†, Wolfgang Schröder-Preikschat*
*Chair of Distributed Systems and Operating Systems
{ulbrich, franzmann, hoffmann, klaus, wosch}@cs.fau.de
†Chair of Automatic Control
christian.harkort@rt.eei.uni-erlangen.de
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

***Abstract*—Robotics systems usually comprise sophisticated sensor and actuator systems with no less complex control applications. These systems are subject to frequent modifications and extensions and have to adapt to their environment. While automation systems are tailored to particular production processes, autonomous vehicles must adaptively switch their sensors and controllers depending on environmental conditions. However, when designing and implementing the process control system, traditional control theory focuses on the control problem at hand without having this variability in mind. Thus, the resulting models and implementation artefacts are monolithic, additionally complicating the real-time system design.**

In this paper, we present a modularisation approach for the design of robotics process control systems, which not only aims for variability at design-time but also for adaptivity at run-time. Our approach is based on a layered control architecture, which includes an explicit interface between the two domains involved: control engineering and computer science. Our architecture provides *separation of concerns* in terms of independent building blocks and data flows. For example, the replacement of a sensor no longer involves the tedious modification of downstream filters and controllers. Likewise, the error-prone mapping of high-level application behaviour to the process control system can be omitted. We validated our approach by the example of an autonomous vehicle use case. Our experimental results demonstrate ease of use and the capability to maintain quality of control on par with the original monolithic design.

***Keywords*-Software Architecture, Software Design, Operating Systems, Robotics and Automation, Sensor Systems, Control Engineering, Measurement System Data Handling**

I. Introduction

Nowadays digital controllers are omnipresent in embedded cyber-physical systems and control theory is a well-known and well-understood engineering discipline. When designing a controller, control engineers usually aim for the highest quality of control achievable. Therefore, they typically try to model the physical system as accurately as possible, integrating the properties of the sensors and actuators as well as their interdependencies in a global system model. In doing so, traditional control theory, which still is the predominant paradigm for many controllers, assumes that all inputs are sampled instantaneously and equidistantly, providing a temporally consistent snapshot of the physical environment [1]. The resulting *process control system* (PCS), which consists of *digital signal processing* (DSP) and control, is therefore monolithic and structured in a relatively simple manner – sample, compute, and act.

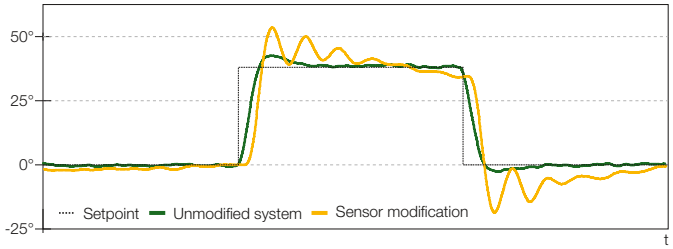

Fig. 1: Decrease in quality of control replacing a sensor by one of slightly different parameters.

On the other hand, robotics systems cover a wide range of application fields, such as industrial automation, medical devices or autonomous vehicles. They are composed of sophisticated mechanical, electronic and software components and are therefore inherently complex systems. Moreover, they have to adapt to their environment, introducing variability at design as well as at run-time. Automation systems, for example, are usually tailored to a specific manufacturing process and will be maintained, modified and extended in conjunction with this process. Likewise, autonomous vehicles may have to adapt to their environment dynamically during operation.

Implementing such complex PCSs and integrating them into a real-time system is anything but easy. Due to its cross-cutting nature this task is interdisciplinary, involving control engineers and computer scientists. Thus, even simple modifications almost inevitably impact both PCS and real-time system. Replacing a sensor, for example, by one of different bandwidth might result in a subtle degradation of control quality.

Fig. 1 shows an example where the assumed sampling rate is violated, which requires the PCS and real-time schedule to be modified appropriately. Although being well-established and suitable for cyber-physical systems with self-contained control applications, this approach has disadvantages when it comes to complex or even adaptive systems. To tackle this issue, a common approach is to integrate the variability into the PCS design by introducing configuration switches into the model.

This work was partly supported by the Bavarian Ministry of State for Economics, Traffic and Technology under the (EU EFRE funds) grant no. 0704/883 25 and the German Research Foundation (DFG) under grant no. SCHR 603/9-1.

This, however, not only moves a lot of application complexity into the PCS by replicating system states and behaviour, but also prevents the computer scientists from implementing a modular real-time system. At this point, the monolithic design of the PCS jeopardizes modularity, adaptivity and reuse.

We believe that an interface is missing between control engineers and computer scientists that allows for a *separation of concerns*. In this paper we therefore focus on the decomposition of the monolithic PCS by introducing a layered control architecture. We show the implications and pitfalls of this kind of modularisation by the example of an *unmanned aerial vehicle* (UAV) robot and show why it is worthwhile for both sides. Experimental results substantiate the feasibility and efficiency of our approach.

The paper is structured as follows: First, we discuss related approaches in Section II. Section III details the proposed control architecture and shows how to integrate interdependencies despite its modular design. We demonstrate the feasibility of our approach on the basis of the use case in Section IV. Finally, we discuss the consequences and benefits of a modular control architecture in Section V and conclude the paper.

II. Related Approaches

The main tool of computer science for dealing with complexity is dividing systems into manageable modules. This approach relies on abstraction, separation of concerns and well-defined interfaces, as described by Parnas [2], and can be used for the modularisation of a PCS in general. Kopetz's *Time-Triggered Architecture* (TTA) [3], for example, is widely used for control applications. It is based on static schedules, which in principle can be reused in new projects and therefore constitute modules. However, the TTA focuses on temporal precision and isolation, rather than addressing PCS specific issues.

Older design models for sensor systems did not focus on modularity: the JDL model [4], the waterfall model [5], the LAAS architecture [6], and the Omnibus model [7] are process models for the creation of data fusion systems. They propose a general workflow when designing such systems but offer no guidelines w. r. t. the structural details of the architecture. Therefore these models are orthogonal to our approach.

Durrant-Whyte [8] shows that in principle modularization of filters and observers is possible, however, this is not the intuitive way of designing such algorithms and the reasons for modularity in control engineering are fundamentally different from those in computer science. According to Chong [9], when modularizing, control engineers aim for cost reduction, load distribution or redundancy. Groves [10] provides a survey of the different ways control engineers may design modular observers and filters. These approaches convincingly show that modularisation is possible from a control engineering point of view. However, real-time aspects of the execution and the interdisciplinary cooperation are not part of the picture.

Interesting architectures similar to our approach are the sensor fusion architectures proposed by Elmenreich [11], [12], Zug et al. [13], or Hightower et al. [14]. Elmenreich's approach was developed in the context of the TTA [3] and is therefore specifically tailored to the time-triggered paradigm. The resulting architectural simplifications impede its general use for example in event-triggered systems.

Zug et al. focus on fault detection in distributed sensor systems. Their architecture provides building blocks for reliable sensor systems communicating through messages.

Hightower's approach is inspired by the ISO/OSI model [15] and divides the DSP system into multiple layers (sensors, measurements, fusion, ...). While this segmentation is helpful, Hightower's approach does not target real-time systems, rendering it inappropriate for our purposes.

III. Control Architecture

We believe a control architecture is missing that facilitates the modification and extension of individual parts of the process control system and bridges control engineering and real-time system design. To gain an optimal system design, a fine-grained mapping of the PCS modules onto real-time system artefacts is desirable. This further allows for effectively isolating components, easing reintegration and reconfiguration. We therefore propose to unravel the traditional monolithic design by introducing a layered architecture.

A. Architectural Concepts

The architecture's building blocks are the outcome of the PCS's basic structure: sample, compute, and act. The mandatory components are data acquisition (sample), processing (compute), and output (act). Each data flow in the PCS is implemented by a sequence of basic components. We decided to use the *measurand's unit* and *value* as well as the *valid range* as a preliminary interface between these components.

When composing the PCS, it is one of the main goals to achieve efficient rejection of various types of disturbances. The impact of disturbances on the overall control loop plays an essential role for choosing the execution rate of the controllers. Compensator concepts involving Kalman-type filtering [16] allow for a design that takes disturbances into account quantitatively. In particular, the *measurement uncertainty* (MU) [17], which results primarily from sensor noise, is an important parameter for the design of the PCS. Consequently, separating the components and data-flows requires the *measurement uncertainties* to be explicitly represented and integrated into the interface. We are convinced that MU is the common ground between control engineer and real-time expert since most temporal properties can be directly derived from it.

In contrast to the monolithic case, additional delays may be introduced by the fine-grained scheduling on component-level. For example, the temporal distance between two – formerly consecutive – sampling instants may increase because of other components executed in-between. These delays are usually neglected, as they are constant and limited to the execution time in the monolithic design. To allow for compensation, the interface therefore includes the *delay* as well. In consequence, a fourth component is necessary to compensate for the MU and the delay, which we refer to as *data integration*. This task is performed by the actual controller.

Finally, to facilitate switching components of a data-flow at run-time (e.g., induced by a mode transition), we propose a *state transfer interface* providing access to the internal state of the components involved. Thereby, the activated component can operate on the current state instantaneously, preventing transient oscillation that may impair the MU.

B. Layered Architecture

Based on these concepts, we propose the layered architecture exemplified in Fig. 2. It basically extends the well-known input-process-output model by inserting additional fusion layers that separate the non-functional properties of the sensor and actuator system from the actual controllers. In the following paragraphs we detail the resulting five layers.

1) Sensor Layer: Contrary to the monolithic system, each sensor applicable to the system is implemented by a dedicated input component. Besides the actual device driver, an input component is defined by its maximum sampling rate and the best MU achievable in the given setup. Thus, the engineer can focus on the implementation at hand and need not concern himself with internal details of other control components.

Each sensor can be operated at its optimum sampling rate, simplifying the mapping to real-time tasks and avoiding pointless oversampling due to the formerly unified period. In consequence, the component implementation has to be extended by a rate conversion filter, adapting it to the subsequent layer.

2) Sensor Fusion Layer: It aggregates and processes data flows to provide the control layer with all necessary measurands. This incorporates fusion of functionally equivalent sensors or estimation of unmeasurable states by observer components. Great care is taken to hide data flows of individual sensors from the observer inputs. The MU of the resulting merged data flow is adapted accordingly. Thus, the developer can disregard the details of the input components and focus on the individual fusion and possibly further filter algorithms.

Preprocessing sensor signals in generic rate conversion filters results in non-static transfer behaviour since the outputs then depend on the history of sensed signals. Since controllers are dynamic systems as well, the interaction of rate conversion, sensor fusion and control may have undesirable effects. It is possible that a combination of rate conversion filters and controllers may affect the performance of the overall control systems in an undesired way. Clearly, the benefits of arranging sensor fusion and controllers in two separated layers can be exploited only if this interaction is avoided. It is crucial to design rate conversion filters such that the dynamics corresponding to their errors are *homogeneous*. Errors may be excited by disturbances only but not by the system response to the control commands. This can be ensured by using observer-type filters, with the side-effect that system states needed in the control layer for the feedback law are generated accidentally. In summary, the sensor fusion layer consists of blocks without internal state and therefore static behaviour, and of observers for filtering and state reconstruction.

Due to the separation of sensor fusion and control layer the outputs of fusion blocks are available not only to the controllers (as in a monolithic setup) but also to additional applications like diagnostics, superordinate control, or mission-specific tasks.

3) Control Layer: Finally, the developer can focus entirely on the specific control application since all sensor and processing specifics are hidden by the preceding layers. Instead of dealing with these details, the controller's requirements are specified in terms of its physical inputs and maximum acceptable MU. Individual controllers may run at their optimum rates, which is an important difference from a monolithic PCS, where the most demanding part of the system determines the global execution rate, resulting in a waste of computation time. The separation of concerns, introduced by our architecture, improves reconfiguration of the control layer by means of additional controllers or dynamic adaptation at run-time.

4) Actuator Fusion Layer: Analogous to the sensor fusion layer, on the actuator side this layer facilitates abstraction from details of actual hardware. On the one hand, it aggregates the controllers' outputs. On the other hand, it distributes the setpoints to the actual actuator system. Hence, the controllers remain independent of the hardware- and rate-independent physical representation of the setpoints. Thus, reconfiguration of the actuators does not require adjustments to the controllers and often simplifies their design process and validation. Similar to the Sensor Fusion Layer, actuator fusion must be *dynamically* separated as well. However, we think that confining actuator fusion to purely static input-output-behaviour is not a difficult restriction, since it just means that the actuator fusion is free of internal states, so that interaction of dynamics is not an issue.

5) Actuator Layer: Finally, the actuator layer provides the output components by means of actuator device drivers, which are operated at a rate determined by the respective actuator fusion component.

C. Design Flow

At first glance, the workflow of the modular architecture appears considerably more complex. However, a large portion of the complexity was hidden inside the monolithic controller design. Most of the additional elements, such as rate conversion, can be derived automatically once MUs are known.

At the price of a slightly more complex interface description, control engineers may continue to use their accustomed design tools. They must, however, annotate sensors and controllers with the provided or required MU. The additional work required to create these annotations will, however, be negligible, since the necessary information is already available as part of the controller and observer design process. On the pro side, the new interface eliminates the need for tedious derivation of temporal properties for a monolithic PCS.

IV. UAV Use Case

To evaluate our architectural approach, we used the *I4Copter* [18], a family of quadrotor robots, as an example of complex autonomous robotics systems. It consists of many variants featuring different microcontroller architectures, sensor settings, as well as diverse propulsion systems. In the remainder of this section we will first introduce the system, describe the

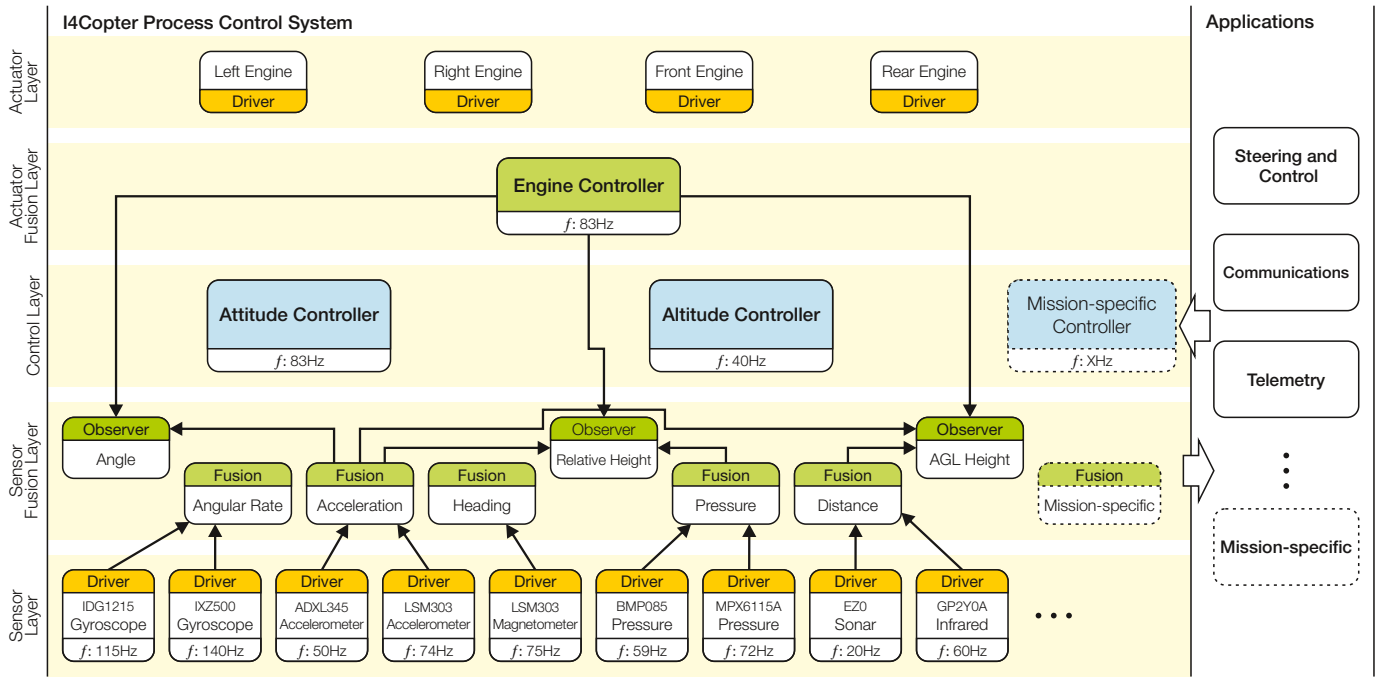

Fig. 2: The controller architecture decouples the crosscutting controllers by introducing additional sensor and actuator fusion layers. They ensure the temporal isolation and independence of the dedicated components and thereby facilitate mission-specific modifications and extensions.

transformation of its formerly monolithic PCS and detail the run-time adaptivity. Finally, we show the effectiveness by the example of two experimental setups.

A. Target System

The *I4Copter* demonstrates typical problems engineers encounter when developing robotics systems. It combines elements of real-time systems and control engineering and has a complex behavioural model. Problems usually encountered in interdisciplinary projects are aggravated by the fact that it is developed by multiple specialist teams.

Since the *I4Copter* is a research project, a high degree of flexibility at design time is necessary. For example, the family model is ranging from the *I4Nano* miniature quadrotor, based on an ARM® Cortex™-M3 microcontoller, featuring four sensors, to a high-reliability variant [19], based on an Infineon TriCore®, offering a redundant setting with up to 12 sensors.

Further, the system includes various behaviour modes that are switched at runtime. For example, being on the ground calls for different behaviour than during take-off or level flight. On the ground and during take-off the distance sensor will be used to determine the vehicle's height, while in flight the distance sensor is unavailable and pressure sensors are employed for altitude measurement. These mode changes originate from different parts of the system – not only control – and are therefore an inherent part of the application logic.

B. Modularisation

When originally implementing the PCS in a monolithic way, a cascaded control scheme was used, the inner control loop being responsible for the stabilization of the vehicle's attitude[1] and setting the thrust. Additionally, an outer control loop was used for altitude control. In both cascades a Luenberger [20] observer and state feedback control created an observer-based compensator. Incoming sensor signals with the same physical meaning were combined by internal fusion blocks. It turned out that a good quality of the observer estimates can be achieved at an execution rate of three milliseconds. The monolithic controller directly provided the four engine setpoint signals.

In the following, we describe the transformation of this monolithic design to its modular version. The resulting layered control architecture is depicted Fig. 2.

Porting the sensor device drivers to the *Sensor Layer* was straightforward and incorporated their extension by rate conversion filters and annotation of bandwidths and MUs, representing the sensor performance within the given system. With this individual representation sensor components can be flexibly mapped onto real-time tasks.

The *Sensor Fusion Layer* was obtained from the monolithic PCS by splitting fusion blocks from observers. It has been emphasized in Section III-B that components of the sensor fusion layer that depend on internal states must be separated dynamically from the control layer. This requirement was satisfied using Luenberger observers [21] for the fused signals (the angles about the three axes as well as its altitude and height). More specifically, for the purpose of noise suppression, these are implemented as Kalman filters, whose covariance matrices are derived from the MUs of the sensor layer.

[1] Yaw, pitch, and roll angles of the aircraft in regard to the body frame

Since the observers and the fusion of corresponding sensor signals are now separated from the control layer, it only consists of the actual attitude and the altitude controller that implement state feedback control with integral part. Due to the inertia of the vehicle a satisfying control performance is now achieved at an execution rate of nine milliseconds. Hence, compared to the monolithic setup, only *one third of the execution rate* is necessary for the controller, yielding computation-time savings. The controllers' outputs are demand signals for the torques about the three axes as well as thrust. Thus, they have a physical meaning, rendering it easy to interpret the controller's behaviour by inspecting its outputs. For the controller design it is important to note that executing the control layer and the sensor fusion layer in different tasks has the consequence that, compared to the monolithic PCS, an additional time delay is inserted into the signal flows. This delay was taken into account by appropriately extending the controller's system model.

The *Actuator Fusion Layer* maps the nominal signals onto the engine setpoints. For this task the inverse characteristics of the engines was implemented so that the engine specifics are hidden from the control layer. Note, that the actuator fusion does not contain dynamics, as claimed in Section III-B.

C. Run-time Adaptivity

As described before, two different physical quantities can be used for estimating flight levels: The *height above ground level* (AGL) describes the distance between ground and quadrotor. This quantity can be precisely measured with time-of-flight distance sensors (e. g., sonar, infrared), which have low sampling rates (50 ms). The *relative height* (RH) indicates the vertical distance between the vehicle and its starting point and can be measured at high sampling rates (3 ms) by very sensitive pressure sensors. During normal operation, RH is used due to the lack of disturbance caused by terrain. In contrast, take-off and touch-down demand for AGL measurement, since there is no way to determine the distance to ground by RH. In consequence, in-flight adaptivity is necessary to switch between AGL and RH controllers depending on the situation.

In the original monolithic design controllers, the observers for AGL and RH, and the state machine for tracking the operational mode were combined in the overall system model. The entire PCS had to be executed at the highest sampling rate leading to unnecessary execution time overhead. Any change in the vehicles's behavioural state machine also implied modification of the control models.

In our layered variant, we separated the observers. Each of them is now running at the same rate as the corresponding sensor (i.e., 3 ms for the pressure and 50 ms for the distance sensor). We merged both, previously dedicated, controllers for AGL and RH into one universal state feedback controller for altitude, running at a rate of 20 ms.

Since the redesigned altitude controller is modelled without internal states, its inputs can be switched in every step, as long as the new input state (in our case vertical position, vertical velocity and vertical acceleration) is observed with a suitable (i. e., low enough) MU. In other words, we can arbitrarily

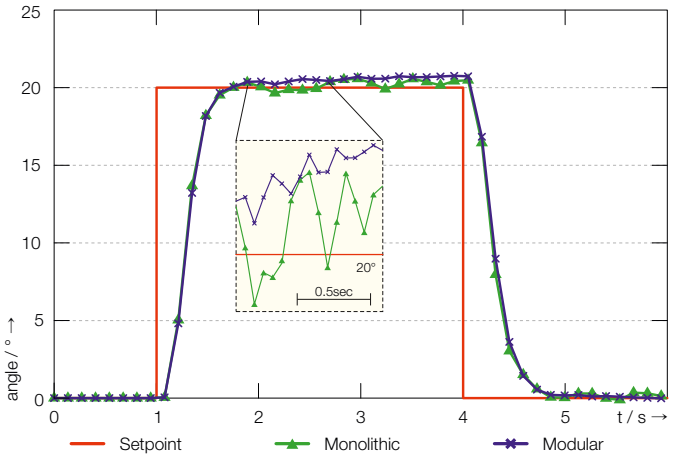

Fig. 3: Angle of the *I4Copter* about the pitch axis under use of the monolithic controller and the alternate modular controller.

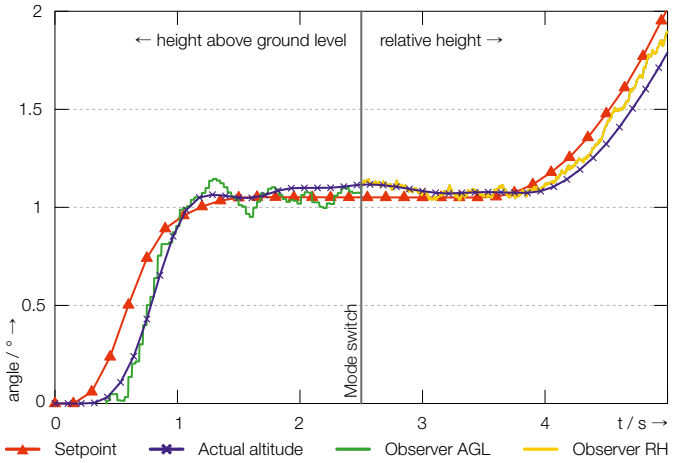

Fig. 4: Change of operating mode during flight at $t = 2.5$ s. After take-off, using slow sampling distance sensors, the altitude control employs faster pressure sensors without changing the controller's parameters.

exchange the underlying observer, depending on the system mode, as long as its corresponding MU is acceptable.

However, in the modular design, the observers are executed only when necessary. Thus, their internal states are likely to differ when changing the mode. Subsequently, the activated observer would have to settle, probably leading to a violation of the MU threshold. We therefore leveraged the state transfer interface provided by the architecture to synchronise the internal states of the two observers during mode change.

Finally, neither the observers nor the altitude controller implement a dedicated state machine but can rely on the application-level behavioural model, simplifying the design.

D. Experimental Results

To demonstrate the feasibility and the effectiveness of our approach we compared our concept of a modular PCS with a classical approach for sensor fusion, controllers, and actor fusion combined in one monolithic block.

In a first experiment we examined the attitude control, the most challenging control application of the *I4Copter*.

The transformation of the monolithic PCS to the modular concept requires only small adjustments of the control layer design, besides the bare rearrangements. The behaviour of both implementations is compared in Fig. 3. It shows the resulting trajectories of the vehicle's pitch axis angle that correspond to an acceleration process. Apparently, the difference between the operation of both PCS is marginal, confirming that the modularisation does not affect the closed-loop performance.

Next, we focused on the run-time adaptivity. Fig. 4 shows the plot for the described mode switch. At $t = 2.5\,\mathrm{s}$ the controlled quantity is switched from AGL to RH. Though the observed values differ at switching time, the state transfer ensures the motion to remain stable and continuous.

Moreover, by executing only those observers actually needed, a lot of processing time can be saved. In our experiments one combined observer running at the more pessimistic sampling rate caused almost twice as much processor load.

Additionally, the development of the height observers was straightforward since the output of other observers (in this case the attitude observer) could be reused. The integration of the observers into the existing system was strongly supported by using the existing actuator layer as the only feedback interface to the attitude controller. In the new architecture the existing attitude controller could be reused without any modification.

V. Discussion and Conclusion

In this paper, we presented a modularisation approach for the design of robotics process control systems, aiming not only for variability at design time but also for adaptivity at run-time. We implemented our approach by means of a layered control architecture that provides a separation of concerns in terms of independent sensor, filter and fusion, controller and actuator components. A tailored interface – comprised of measurement, measurement uncertainty, and delay – allows for reintegrating complex interdependencies that are naturally considered in a holistic and monolithic design approach. Our approach not only facilitates a modular design of the PCS but also a fine-grained mapping to the real-time system. Thus, we bridge the gap between computer science and control engineering.

However, modularisation does not come for free. Computer scientists have to consider more aspects of the PCS. In addition to their usual tasks, they have to derive timing properties from MU and determine delays due to fine-grained PCS scheduling. Control engineers, on the other hand, have to adhere to the interface and to provide the MUs. Dynamics introduced by the additional rate conversion filters have to be eliminated or considered during observer design. The additional interfaces for state-transfer have to be provided in case of run-time adaptivity.

We consider it worth the effort and, in our experience, for both sides the overhead caused by our architecture is minimal. The benefits, however, are manifold. Application logic such as state machines is no longer part of controllers and observers, avoiding its tedious and error-prone reimplementation. Thus, the PCS as well as the real-time system become much simpler. Our interface allows for development of controllers, observers and system software like hardware drivers and the real-time system by independent teams, fostering interdisciplinary system development. We believe that using MU as a common ground between real-time and control systems opens up new opportunities for run-time adaptivity in future designs. We are also exploring ways of providing a control-aware adaptive real-time architecture [22] complementing the control architecture presented in this paper.

References

[1] M. Törngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems Journal*, vol. 14, pp. 219–250, May 1998.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *CACM*, pp. 1053–1058, Dec. 1972.

[3] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[4] D. L. Hall and S. A. H. McMullen, *Mathematical Techniques in Multisensor Data Fusion*, ser. Artech House information warfare library. Artech House, 2004.

[5] 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 of The Royal Aeronautical Society*, 1997.

[6] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An architecture for autonomy," *Int. J. Robotics Research*, vol. 17, no. 4, p. 315, 1998.

[7] M. Bedworth and J. O'Brien, "The omnibus model: A new model of data fusion?" IEEE *Aerospace and Electronic Systems Magazine*, vol. 15, no. 4, pp. 30–36, 2000.

[8] H. Durrant-Whyte, "Multi sensor data fusion," January 22nd 2001.

[9] C. Y. Chong, "Hierarchical estimation," July 1979.

[10] P. D. Groves, *Principles of* GNSS*, Inertial, and Multisensor Integrated Navigation Systems (*GNSS *Technology and Applications)*. Artech House Publishers, 1 2008.

[11] W. Elmenreich and S. Pitzek, "The time-triggered sensor fusion model," in *5th Int. Conf. on Intelligent Engineering Systems (INES '01)*. IEEE, 2001, pp. 297–300.

[12] W. Elmenreich, "Sensor fusion in time-triggered systems," Ph.D. dissertation, 2002.

[13] S. Zug, A. Dietrich, and J. Kaiser, "An architecture for a dependable distributed sensor system," *IEEE TIM*, vol. 60 Issue 2, pp. 408–419, Feb. 2011.

[14] J. Hightower, B. Brumitt, and G. Borriello, "The location stack: A layered model for location in ubiquitous computing," in *IEEE W'shop on Mobile Computing Systems and Applications*. IEEE, 2002, pp. 22–28.

[15] I. O. F. Standardization, "ISO/IEC 7498-1:1994 information technology – open systems interconnection – basic reference model: The basic model," *International Standard ISOIEC 74981*, p. 59, 1996. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/s025022_ISO_IEC_7498-3_1997(E).zip

[16] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *ASME J. Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[17] P. Bevington and D. K. Robinson, *Data Reduction and Error Analysis for the Physical Sciences*, 3rd ed. McGraw-Hill Science/Engineering/Math, 7 2002.

[18] P. Ulbrich, R. Kapitza, C. Harkort, R. Schmid, and W. Schröder-Preikschat, "I4Copter: An adaptable and modular quadrotor platform," in *26th ACM Symp. on Applied Computing (SAC '11)*. New York, NY, USA: ACM, 2011, pp. 380–396.

[19] P. Ulbrich, M. Hoffmann, R. Kapitza, D. Lohmann, W. Schröder-Preikschat, and R. Schmid, "Eliminating single points of failure in software-based redundancy," in *9th Eur. Dep. Computing Conf. (EDCC '12)*. IEEE, May 2012, pp. 49–60.

[20] D. G. Luenberger, "Observing the state of a linear system," *IEEE Trans, on Military Electronics*, pp. 74–80, 1964.

[21] S. Kailath, *Linear Estimation*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2000.

[22] P. Ulbrich, F. Franzmann, F. Scheler, and W. Schröder-Preikschat, "Design by uncertainty: Towards the use of measurement uncertainty in real-time systems," in *7th Int. Symp. on Industrial Embedded Systems*. IEEE, 2012.