I4Copter: An Adaptable and Modular Quadrotor Platform

Peter Ulbrich
Chair of Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

Rüdiger Kapitza
Chair of Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

Christian Harkort
Chair of Automation Control
Friedrich-Alexander University Erlangen-Nuremberg

Reiner Schmid
Siemens Corporate Technology, Munich

Wolfgang Schröder-Preikschat
Chair of Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

ABSTRACT

Quadrotor helicopters are micro air vehicles with vertical take-off and landing capabilities controlled by varying the rotation speed of four fixed pitch propellers. Due to their rather simple mechanical design they have grown to popularity as platform for various research projects. Despite most of them being individually highly successful, they are typically tailored to a specific purpose making it hard to utilise them for further research and education.

In this paper, we present the novel design of the *I4Copter* quadrotor. It has been developed to provide a stable demonstration quadrotor platform for various kinds of research and education projects targeting cross-field challenges in real-time and embedded systems, distributed systems, robotics and cybernetics. The modular and open architecture of our platform allows an application-specific, fine-grained extension, adaption and replacement of software and hardware components. The safe extensibility is supported by strict temporal and spatial isolation between the software modules. We validated our approach by two distinct cross-field use cases: an evaluation platform for modularised control algorithms enabling trajectory tracking and an implementation that is resilient to transient hardware errors.

1. INTRODUCTION

Quadrotor helicopters (quadrotors) are a special variant of micro air vehicles with vertical take-off and landing capabilities. Their mechanical design is rather simple and relies on four fixed pitch propellers, pair-wise spinning in the opposite direction, and – in most cases – a simple gearless drive. Their attitude[1] is solely controlled by varying the rotation speed of the engines and the construction, operation and maintenance is, therefore, relatively easy and cheap.

[1]Angle of the quadrotor in regard to a reference point

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

Due to these benefits, quadrotors have grown to popularity among governmental, industrial, academic and private users around the world. Here, they demonstrate their abilities for personal diversion, flying camera mount or for serious surveillance and reconnaissance tasks performed by the police and military. While the mechanical setting of a quadrotor is rather simple, the challenge is to reliably control its inherently unstable flight characteristics. This has to be realised using inertial measurement and software flight control. Therefore, a quadrotor helicopter is a challenging example of a safety-critical embedded real-time system, making it a attractive evaluation platform to explore new ideas in the fields of control theory [5, 11, 10, 1], navigation [3] and real-time systems [**?**, 2], just to name a few. In summary, quadrotors have been implemented by various interest groups that can be categorised into hobbyist, companies and researchers.

The resulting solutions have their individual strengths, but all share a common weakness. They are designed for a special purpose; for example, to provide maximal flight qualities, to enable reliable surveillance or to demonstrate an individual new technique in research. In our opinion, a solid base system that can be flexibly tailored to current and future research as well as new application areas is missing. While there are popular open source platforms, with the MikroKopter project[2] as a protagonist, we made the experience that it is comparably hard to extend them (e. g., by indoor navigation support), as the control software is highly integrated. Commercial platforms such as provided by microdrones[3] show a remarkably good overall performance, but they are typically not available to the community, which inhibits their use as a basis for research. Finally, research projects usually focus on specific questions and naturally try to achieve this in the fastest possible way. This may lead to good results from an individual research perspective but slows down the general progress, as individual researches have to jump through the same hoops of building a base system to make progress in their specific field again and again.

We therefore decided to go a different way by building a modular quadrotor platform that facilitates research and education in the fields of real-time and embedded systems, distributed systems, robotics and cybernetics. This approach raises new aspects and challenges in designing such a modular platform. While having

[2]MikroKopter homepage – `http://www.mikrokopter.com`
[3]Microdrones homepage – `http://www.microdrones.com`

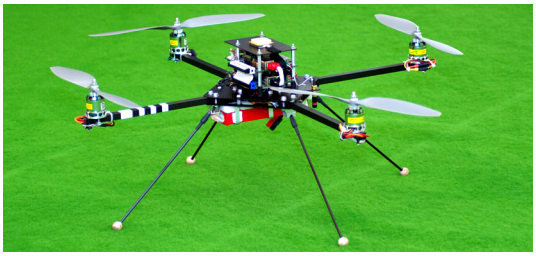

Figure 1: The Apollo *I4Copter* prototype.

modularity in mind, we made an – at first sight – surprising choice by consolidating the hardware setting compared to the state of the art and combining all tasks on one central microcontroller. Colocating the time-critical flight control with less critical application software (e. g., communication) requires strict isolation and clearly defined interfaces between all tasks.

This is achieved by using a real-time operating system (RTOS), a stringent analysis of the real-time components and a spatial isolation via hardware memory protection. Thus, the consequence is a slightly more complex system design; however, it offers various benefits for further extension. In many cases, the application-specific high-level control has to interact with the low-level flight control in a specific way to avoid undesirable effects caused by the subsidiary control. An indoor collision avoidance may, for example, want to disregard the basic motion control, as it is not adequate for evasive manoeuvres. Furthermore, any high-level control should have access to the sensors raw data at maximum sampling rate as it might require its own filtering and sensor fusion algorithms. Again, an appropriate solution to support adaptability at this level is to integrate those two parts onto a single but powerful microcontroller.

In this paper, we present the novel design of the 3rd generation of the *I4Copter*[4] quadrotor family that combines the experience of 4 years of research and hundreds of test flights. This final desgin has been developed to provide a stable and extensible research, education and demonstration quadrotor platform by featuring not only modular software but also a modular hardware and controller design. We have validated our approach by two distinct use cases: an evaluation platform for modularised control algorithms enabling trajectory tracking and a quadrotor implementation that is resilient to transient hardware errors. While the former demonstrates the capabilities of our approach to incrementally extend the controlling of a quadrotor making new ideas in control theory practical evaluable, the latter exemplifies a profound enhancement and explores a novel design for providing safety critical applications. In sum we consider the *I4Copter* as the first deeply extensible quadrotor platform that enables to address cross-field challenges in the domain of real-time and embedded systems, distributed systems, robotics and cybernetics. This has been achieved by a careful design process and a combination of operating system, software engineering and control theory concepts as well as the use of industry grade hardware from the automotive sector.

The paper is structured as follows: First, we outline the hardware setting including microcontroller, sensors and all mechanical parts. Section 3 details our software architecture and describes the controller design. Thereafter, we illustrate the benefits of our system architecture on the basis of the two use cases. Finally, we discuss related approaches in Section 5 and conclude.

[4] *I4Copter* homepage – http://www4.informatik.uni-erlangen.de/Research/I4Copter

2. HARDWARE PLATFORM

In this section we describe the basic hardware design of the *I4Copter* including the airframe with the propulsion, the sensor system and the microcontroller. In general, the system has been designed and developed to resemble embedded real-time systems as employed in real-world industrial scenarios such as the automotive sector and to be adaptable to new mission scenarios.

2.1 Hardware Requirements

When selecting the individual hardware components, we took the following requirements into account:

- **A microcontroller that** (cf. Section 2.2)
  - supports hardware memory protection,
  - offers sufficient memory and computing resources
  - and free I/O to support additional use cases.
- **A sensor system that** (cf. Section 2.3)
  - is based on commercial off-the-shelf components,
  - offers a direct connection between each sensor and the microcontroller (providing raw data),
  - supports stable flight in its basic configuration
  - and is extensible to support additional use cases.
- **A propulsion system that** (cf. Section 2.4)
  - has a minimal flight time of 15 min
  - and can carry at least 500 g of additional payload.

While the first set of requirements enables us to safely consolidate basic flight control and more complex behaviours on one controller, the second set offers unrestricted access to sensor values providing the freedom to flexibly filter and process these in the context of new use cases. The last set of requirements basically ensures a practical system setting.

2.2 Microcontroller

To meet these requirements, the *I4Copter* is equipped with an Infineon TriCore® TC1796 microcontroller [6], as commonly used in automotive ECUs and a custom-made sensor periphery board featuring a wide range of sensors (8 in total) and a variety of interfaces. This choice enables us to utilise automotive standards such as OSEK (open systems and their interfaces for the electronics in motor vehicles), which offers powerful industry-grade tool and operating system support.

The TC1796 is a 32-bit RISC microcontroller running at 150 MHz, featuring 2 MB flash and 64 KB internal memory and, important for the aspired application isolation, a full-featured memory protection unit. In addition, it provides several versatile peripheral units, such as four serial controllers, two analogue-to-digital converter units (with 32 inputs each) and two general-purpose timing arrays (64 I/Os each). Out of the commercially available evaluation boards, we decided to use HighTec's[5] EasyRun Board as a basis for our development as it features 1 MB MRAM and an additional Ethernet on-board controller combined with a small form factor.

[5] HighTec homepage – http://www.hightec-rt.com

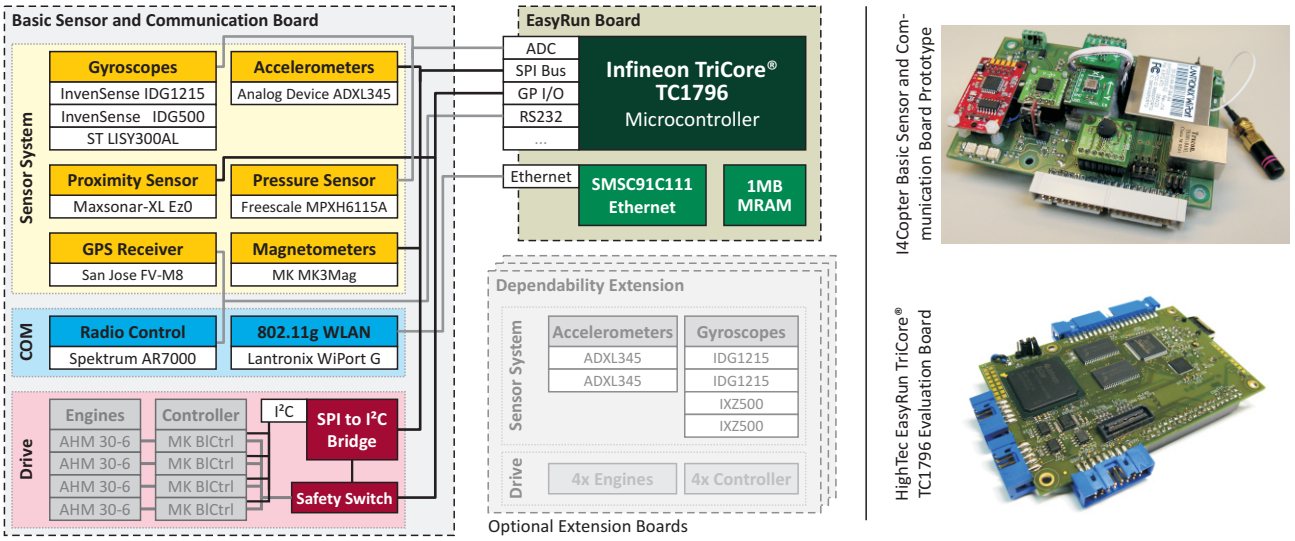

Figure 2: The *I4Copter* hardware platform consists of several modules that can be combined according to the mission scenario. The base system is built of the microcontroller and the basic sensor board.

2.3 Sensor and Communication System

The sensor and communication system is the core of the hardware platform (see Figure 2). It consists of three major parts, the *Sensor System* for attitude stabilisation, the communication facilities for steering and measurement data transmission (*COM*) and an interface to the engine controllers (*Drive*).

To avoid any built-in estimation and filtering algorithms, but to have full control of the sensor system, we decided to use basic sensors and commercial off-the-shelf components. Therefore, three gyroscopes and accelerometers each form the basic three-axis inertial measurement unit (IMU) sensing the angular rate and the acceleration of the vehicle around the roll, pitch and yaw axis. Additional sensors are used to support the pilot; for instance, magnetometers are utilised as a digital compass and a combination of proximity and pressure sensors is used to determine the altitude. Finally, using a GPS device, the vehicle can even support waypoint flight and maintain its position autonomously. The *Software Architecture* (cf. Section 3.1) does not necessitate the sensors to be of a certain type or interface. Thereby, they can be mixed or even replaced between hardware revisions as long as they provide the same type of information and an appropriate driver is developed. The latter has, for example, been verified by integrating different types of accelerometers (ADXL202, ADXL335 and ADXL345[6]). In the same way, the sensor system can be extended by adding optional extension boards (cf. Section 4.1).

The communication facilities consist of two independent channels, a standard 802.11g WLAN and a 2.4 GHz radio control. The WLAN adapter is connected via a bridge to the Ethernet interface and offers a flexible and high-bandwidth facility for sending measurement data and receiving control commands. The radio control is unidirectional; however, the effective communication range is higher compared to the WLAN and it can be used with off-the-shelf remote controls.

2.4 Airframe and Propulsion

The airframe has a conventional cross-shaped design made of carbon-fibre square tubes and aluminium interconnects. The *I4Copter* is equipped with 12x3.8 in propellers, resulting in an airframe span of 65 cm and an overall span of 91 cm.

The propulsion system consists of four brushless engine controllers and engines that are powered by a 14.8 V lithium polymer battery. In this configuration, the system is capable of constantly generating a thrust of 36 N in total. Based on a thrust margin of 40 % required for steering and holding off the vehicle, the maximum take-off weight is about 2120 g. The current prototype of the *I4Copter* weighs approximately 1400 g, leading to more than 700 g payload, and can maintain flight for more than 20 min. In sum, this makes the *I4Copter* competitive with the state-of-the-art.

[6]Analog devices homepage – `http://www.analog.com`

Table 1: Events and their period

Event	Type	Period
Sensor processing	periodical	4.7 ms
Attitude control	periodical	12 ms
Monitoring	periodical	25 ms
Command	aperiodical	25..250 ms

2.5 Temporal Properties

The temporal properties of the quadrotor are the result of the analysis of the physical events (e. g., control, commands or interference) affecting the vehicle, the physical cause and the consequences. These properties have a significant impact on the real-time properties of the software system. In our case, we have identified the four major events depicted in Table 1. Several associated, in particular aperiodic events (e. g., emergency shutdown) are not listed due to brevity of the paper. To analyse their temporal properties, we have measured the causing physical parameters (e. g., weight and engine response time) using various test rigs and thereof build a physical model and derived the unmeasurable parameters (e. g., vehicle inertia).

In detail, *sensor processing* is a periodical event caused by pending data and is triggered, according to the sampling theorem, with twice the sensor output frequency. This is 105 Hz in our case, leading to a period of 4.7 ms. The *attitude control* is caused by a change in motion and interference (e. g., wind) and depends on the engine response time of ~120 ms, leading to a 12 ms period with ten times overapproximation. In the same way, *monitoring* the vehicle's flight

condition is a periodical event; its period of 25 ms is derived from the vehicle inertia. Finally, the user operating the quadrotor causes aperiodic *command* events. Their minimal interarrival time is determined by the communication channel and the reaction time by the human response time.

3. SOFTWARE ARCHITECTURE

This section details the software architecture of the *I4Copter*. It has been designed with modularity as a key requirement, to support high level application integration while providing real-time and isolation properties. The latter has to be preserved even in the presence of shared resources like the serial communication bus.

Consolidating all tasks on a single real-time system does not only demand for isolation support but also the handling of various types of periodical (e. g., attitude control) and sporadic (e. g., command) events that have an impact on the system. A central aspect of our architecture is therefore to be analysable, which is achieved by a completely static setting.

3.1 Software Architecture Requirements

Shifting the separation of the time-critical basic flight control and the more complex application logic from hardware to software by colocating the former two on a single microcontroller raises further requirements that have to be addressed by the software architecture as well as the underlying system software:

- Strict temporal and spatial isolation of all software components
- Modular and configurable core architecture that supports predictable runtime behaviour
- Measurement data provision at actual sampling rate
- Software signal processing, filtering and estimation

3.2 Core Architecture

We have chosen a system and software engineering centric approach to tackle the aforementioned requirements. Therefore, the *I4Copter* software architecture has been designed as a lightweight and predictable framework and features a component-based design written in C++. A real-time operating system is used for the execution of the software components and for instrumenting the hardware memory protection offered by the TriCore microcontroller. Overall, the base system – including the flight control – comprises approximately 26.000 LOC (without the OS). The flight-control system with all available features enabled utilises less than 1/3 (28,4 %) of the CPU, leaving enough spare capacity for user provided tasks and features. The memory footprint of 164 KB is less than 10 % of the available system memory.

To support the real-time analysis and predictability, wherever possible, periodic activation and polling is used to minimise the coupling and impact of events on the rest of the system. The base software components feature high coherency and minimal coupling among them, and their dependencies are reduced to simple data flow dependencies wherever possible.

In total, the core architecture (see Figure 3) consists of the base *System* managing the hardware, the *Ethernet* and *SerialBus* resource handlers operating the communication interfaces, the *Signal Processing* component for data acquisition and the *Flight Control* and *Copter Control* components implementing the low level control and the high level behaviour of the quadrotor.

3.3 System

The *System* layer provides the base framework for the *I4Copter* software architecture. On the one hand, it acts as an abstraction layer towards the hardware. On the other hand, it provides services for data exchange, exception handling and consistent operational mode management for all other software components.

At the moment, two real-time operating systems can be used to execute the application system, whereas the OS-specific calls are covered by a thin abstraction layer: The commercial PXROS-HR[7] and the CiAO OS [9], which is developed at our lab. PXROS-HR is a stable and well supported RTOS platform and used in SIL-4 applications. HejDa is open source, highly configurable and features an AUTOSAR[8] API. The latter enables the integration of automotive software developments for evaluation purposes. An outstanding feature both have in common is the support for hardware-based isolation using memory protection.

In the same way, the TC1796 microcontroller-specific control of the periphery modules is encapsulated in a thin, statically configured *TriCore Hardware Abstraction Layer*, which supports the *Copter Hardware Device Drivers* controlling the sensors (e. g., gyroscopes) and actuators (e. g., engine controllers) directly attached to the microcontroller. Thereby, we perform the signal filtering at the level of the device drivers, as their parametrisation is highly device specific.

Besides the abstraction, the *System* layer provides service primitives for component interaction. This is, base classes for system and application components as well as a type-safe, template-based connector mechanism used to implement data flow between memory protection domains at different levels of protection. Depending on the actual data flow scenario and the data type, the connectors are reduced at compile-time to simple pointers or are mapped to OS-based messaging. The software component support is completed by a basic exception handling and an operational mode management. The former enables components to signal exceptions of different severity (e. g., warning, error and panic) within predictable time limits. The operational mode manager takes care of a consistent view on the system's operational mode and state among the components.

3.4 Shared Media Handlers

In general, hardware devices can be attached to the microcontroller using either point-to-point connections (e. g., I/O pin) or shared media via a communication bus. In any case, the software interfaces of the device drivers should be fixed even when changing the connection type. Therefore, the serial periphery bus (SPI) and the Ethernet interfaces are operated using the *SerialCom* and *Ethernet* handler components, respectively. They have been designed to maintain the isolation of the components that are using bus devices.

The *SerialCom* design is to store the serial message within each bus device driver individually, granting the handler exclusive memory access only when actually communicating. By the time the answer has been received, the handler releases the memory again, thereby also implementing synchronisation with the component. Further implementation details regarding the temporal isolation are discussed in Section 3.8.

The *Ethernet* handler is used for parsing incoming and for building outgoing packets. The protocol used is a custom design on top of the UDP protocol. Its uniform packets are configured statically at compile time. In combination with a strictly periodical transmission the resource consumption is constant at runtime.

3.5 Signal Processing

The sensor data acquisition and filtering is done by the *Signal Processing* component. Therefore, it consists of several subcomponents that represent either the different kinds of sensor classes attached to

[7]Freely available for educational use (`http://www.hightec-rt.com`).

[8]AUTOSAR homepage – `http://www.autosar.org`

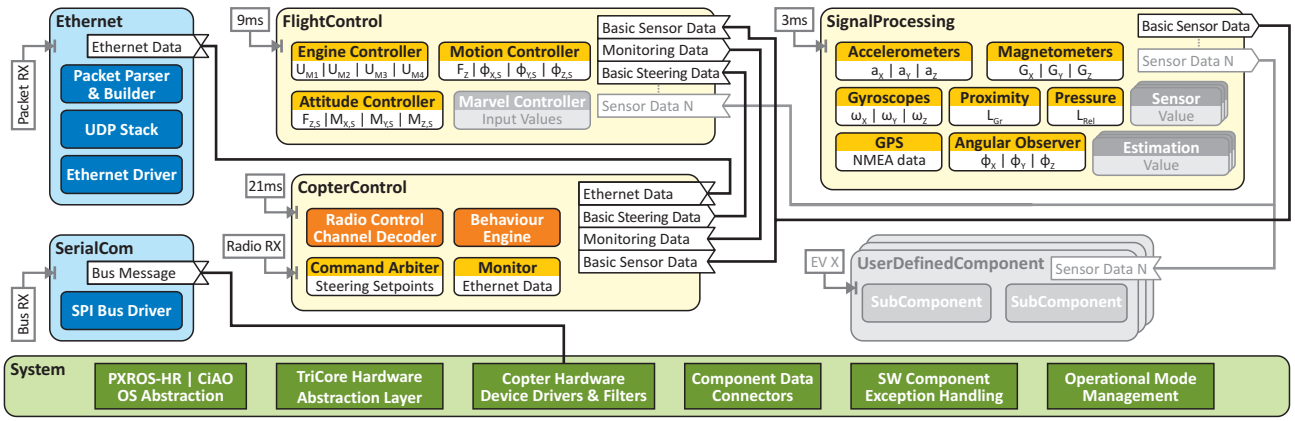

Figure 3: The *I4Copter* software architecture

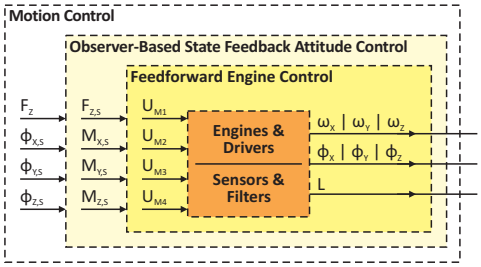

Figure 4: The cascaded attitude controller of the *I4Copter* enables to hook up additional controllers

the system or estimation filters. All measurement values are stored and distributed in a type-safe manner, using the appropriate unit.

The device drivers that belong to a certain sensor class are encapsulated by a subcomponent, thereby providing a single, stable interface to the rest of the system. This is especially useful for multi-axes sensors or for multiple sensors covering a sensing range. Furthermore, the scheduling of (lower) divergent sampling rates is also handled within the sensor class subcomponents.

The estimation filters located in the *Signal Processing* can directly access the raw data and work with the same sampling rate. They are used not only for Kalman filtering but also for sensor fusion and reconstruction of unmeasurable system states. The most important estimates are the vehicle angles around the pitch and roll axes.

3.6 Copter Control

The *Copter Control* component is responsible for receiving operator commands, sending measurement data and for the high level behaviour of the quadrotor. The steering data is provided by either the radio control device driver or the *Ethernet* component or both of them. In the latter case, the data is merged and prioritised. Whereas the data sent by the radio control is fixed, complex flight and waypoint plans can be submitted using the Ethernet interface. Both connection types are strictly periodic and used as a heartbeat at the same time to detect connection loss. In the opposite direction, *Copter Control* forwards relevant measurement data to the ground station via Ethernet. Being the vehicle's monitor, *Copter Control* also implements the basic behaviour control for emergency responses in the case of a cut-off communication, low battery and other exceptions including an immediate emergency stop.

Table 2: Events and their appearance

Event	Component/Task	Period	WCET
Sensor processing	*SignalProcessing*	3 ms	422 µs
Attitude control	*FlightControl*	9 ms	260 µs
Monitoring & Steering	*CopterControl*	21 ms	472 µs

3.7 Flight Control

The software component with the highest impact on the user's perception is *Flight Control*, as it implements the basic control system for maintaining a stable and easy to control flight. Its design (see Figure 4) is modular and organised in a cascaded structure of three layers. This has the advantage that not only high level motion control is available for the operator but that also the inner control loops can be accessed. The attitude controller is available as a MathWorks™ Simulink®[9] model and the controller code is generated using Simulink's Real Time Workshop.

3.8 Real-Time System Design

Finally, the software components and physical events outlined in Section 2.5 have to be mapped onto tasks that are scheduled within the real-time operating system. As the base components have been designed with events in mind, this mapping is straightforward, as shown by the selected set in Table 2. All time-critical tasks are executed strictly periodically. However, the aperiodic events have to be handled asynchronously by using for example (short) interrupt handlers or background execution. Soft real-time tasks, for example the Ethernet task, are scheduled for background execution.

One may notice that the period of the periodic tasks is harmonic. This helps to determine appropriate phase offsets for the tasks and to avoid overlapping tasks, which is especially important when dealing with shared resources like the serial communication bus. Accordingly, the resource allocation can be done ahead of runtime implicitly using a static schedule. Therefore, the allocation protocol restricts each device driver to one bus message per task activation. For calculating the phase offset for a certain setting, we have determined the worst-case execution time (WCET) of each component and interrupt handler using the Absint[10] aiT WCET analyser. In the resulting static schedule the tasks do not interfere with each other.

[9] MathWorks homepage – `http://www.mathworks.com`
[10] AbsInt homepage – `http://www.absint.de`

4. USE CASES

Next, we will outline two distinct use cases extending the *I4Copter* base platform. Firstly, a variant supporting redundant execution that is resilient to transient hardware errors. Secondly, an evaluation platform for engineering and evaluating control algorithms.

4.1 A Soft Error Resilient Quadrotor

Future hardware designs for embedded systems will exhibit more parallelism at the expense of being less reliable. Detecting transient faults (also known as *soft errors*), and tolerating their effects, has been a popular topic in the context of mission-critical systems (e. g., space computers) for decades. However, there is limited knowledge on how these faults will be handled once off-the-shelf components are afflicted with these kinds of faults. Conveniently, recent work on software protection mechanisms was done in cooperation with Siemens [12]. We therefore decided to implement an error resilient version as an use-case.

As a first explorative step, we extended the *I4Copter* by triple modular redundant execution [4]. At the hardware level this was achieved by a second extension board that duplicates the essentially needed sensor values by a second set of accelerometer and gyroscope sensors. As we require another set of values to enable voting, we decided to derive these values using sensor fusion algorithms. In sum, we gain three sets of inputs that are handed over to an interfacing flight control task which stores them in different memory regions for gaining fault independence. Next, we tripled the flight control task. Each of the tasks using its own set of sensor input values. Finally, the outputs of the three instances are combined and voted. We designed the voter to be as simple as possible to reduce the risk of soft errors at this critical stage.

The extension of our first prototype *Icarus* to the soft-error resilient version *Apollo* (see Figure 1) went quite smoothly. As our system design was already very modular and component-based we only had to implement the new interfacing task and the voter. The actual replication step of the flight control can be considered as matter of extended configuration. This included building a new schedule. Compared to the basic variant the system load increased by only 10.4 %. Furthermore, the temporal and spatial isolation provided by the OS and supported by the microcontroller was very beneficial for the development and for achieving fault independence. Hardware-wise the extension board could be easily attached to the existing system and the core microcontroller had enough input ports to handle the additional sensor signals. Aside from the positive experience, we also identified the former *SerialCom* to hinder the background execution and to be obstructive when setting up the schedule for the system. The reason was the easy to analyse but inefficient synchronous operation of the serial bus, leading to gaps in the schedule and waste of CPU time. For this reason, we extended the *SerialCom* to provide asynchronous operations (see Section 3.4).

4.2 Trajectory Tracking Control

The applicability of the *I4Copter* platform for evaluating control engineering algorithms has been proven by implementing a trajectory tracking control featuring waypoint flight in the three dimensions and an automated take-off and landing control.

The former has been implemented by extending the cascaded controller by a fourth layer controlling the vehicle's velocity and position on the basis of the GPS data. To obtain the velocities, the *Signal Processing* has been extended by additional estimation filters for pitch and roll as well as for the yaw axis. The latter is realised by a sensor fusion of the yaw accelerometer and the pressure data, bypassing the existing filters and using raw data. An additional user task computes the motion paths for the waypoint heading. The autopilot is working in parallel to the existing controller and thereby is able to compensate the noisy signals caused by the ground effects. Therefore, it features an alternative implementation of the attitude controller replacing the original one during the starting and landing phase.

The modular system and controller design enabled a straight adaptation to the mission-specific needs. The connectors used for data exchange between the components proved to be useful for integrating the velocity estimation and the additional task calculating the motion paths. The mode management provided by the framework helped us to implement the different operational modes defined by the control engineers.

5. RELATED APPROACHES

Existing quadrotor projects can be roughly subdivided by their main interest focus: commercial, hobbyist and research.

Commercial solutions available today such as products provided by microdrones offer already a wide range of support for surveillance up to the point of on-demand development for a specific task. Despite these facts, access to their code is not possible or associated with high costs and licence restrictions. Therefore, their use in research and education is rather limited.

The community-based project MikroKopter is a representative for a hobbyist project. It builds the starting point for hobbyist and researchers enabling initial work in the field of quadrotors. This is fostered by the availability of cheap and working toolkits and an open-source implementation of the control software. As downside, this project realised the whole flight control in one main loop based on a small Atmel® ATMega644 and avoids using an operating system. From an engineering perspective this is a poor choice as it makes further extension and evolution of the system very hard. In consequence, researchers tend to extend the platform by an additional microcontroller and have difficulties to adapt the control software to the changed flight characteristics.

Researchers usually focus on their research topics and naturally select the easiest possible way to implement them. Some projects like the *OS4* [1] or *X-4 Flyer* [11, 10] quadrotors have been design for evaluating the basic controller theory of quadrotors and therefore have the same drawbacks regarding reusability as the community driven projects (e. g., lack of an operating system). Other research projects, such as the *STARMAC* [5], do address application-specific programming. Therefore, these projects incorporate operating systems but still rely on a strict separation of low and high level control on multiple microcontrollers. The *JAviator* [**?**, 2] features a system design similar to that of the STARMAC – although being a lot bigger and better integrated – and also uses a Robostix (Atmel ATMega128) for low level hardware control as well as a Gumstix (Intel XScale PXA255) for high level vehicle control. In contrast to all other projects, the JAviator developers have identified the need for spatial isolation. Accordingly, they are either using Real-Time Linux or their own *Tiptoe* RTOS and Java-inspired programming paradigms. Showing a promising solution, however, the separation into a low and a high level microcontroller still remains. A characteristic that OS4, STARMAC and JAviator have in common is the usage of integrated IMUs incorporating the basic filters and estimation algorithms. This impedes an application-specific reuse as there is no access to the individual sensor raw data at high sampling rates.

6. SAFETY ASPECTS

Despite being a model-size aircraft, a quadrotor can cause serious damage and must be considered under safety aspects. For the development of the safety-critical parts of the system – this is mainly steering and flight control – we used industry grade tooling and analysis techniques for both functional as well as real-time aspects. The basic flight control is model-based, developed and tested by control engineering experts using Simulink. So far, there are four *I4Copter* in total, two of them are used by our group and one each by Siemens Corporate Research in Munich and Princeton. There, the overall system design has shown its applicability and reliability through rigorous testing and hundreds of test flights.

However, we encountered an unexpected but retrospectively rather obvious safety aspect while evaluating the trajectory tracking control use case. A bug in one of its specific components jeopardized the aspired fault isolation, which almost lead to the loss of a quadrotor. To be specific, the bug indirectly affected the radio control component in case of a connection loss[11], where it impeded the transition to the prioritised emergency mode. In general, the *I4Copter* architecture implements component interaction by state messages [8] like data flow coupling. Apart from that, the components have to form and obey a consensus of the current operational mode of the system (e.g., ready, flight, emergency). Here, the issue is that the shared global operational mode leads to a distributed and parallel state machine [7] if more than one component triggers a certain mode – this cannot be mapped to state messages. In consequence, the correctness and the temporal order of the state machines involved for a certain transition has to be verified and can be seen as the TCB.

As a first step to regain the intended modularity and the isolation properties, we introduced the actual operational mode management, which is a turn-based arbiter for the operational mode. With this, the parallel, component-local state machines are synchronised at a specific point in time, and the consistency of a suggested mode transition can be checked. To reduce the TCB for the vital emergency transition we decided to build a simplified and provable version of this part of the state machine as a watchdog. In addition, we are currently working on a provable version of the local state machines picking up the approach described in [7] and using MathWorks™ StateFlow®.

7. CONCLUSION

We presented the novel design of the *I4Copter* quadrotor family. Unlike related approaches that basically target to provide maximal flight qualities or demonstrate an individual new technique in research, we combined the experience of 4 years of research and numerous test flights as well as various prototypes to provide a stable and modular platform. Thereby addressing cross-field challenges in the fields of real-time and embedded systems, distributed systems, robotics and cybernetics. This has been achieved by consolidating all control software to a single but powerful microcontroller and the spatial isolation via hardware memory protection. Furthermore, our architecture exhibits a modular software and controller design as well as an adaptable communication infrastructure while being analysable due to its strictly static setting. We validated the benefits of our approach by two challenging use cases: an evaluation platform for modularised control algorithms enabling trajectory tracking and a soft-error resilient variant of the *I4Copter*. While the first case makes formal control theory practically applicable the second explores the design space of safety critical applications. Preliminary results from the associated research projects at Siemens substantiate the experience we made with the two use cases. In sum, the overall system design resembles industry and automotive applications which makes it a great basis for research and educational use to attack and solve cross-field challenges.

[11] To be considered as a model aircraft, a manual override is mandatory by law in many countries

8. ACKNOWLEDGMENTS

We would like to thank Daniel Christiani for his restless support in constructing and wireing the *I4Copter* prototypes, Fabian Scheler for the prolific discussions about real-time aspects as well as Florian Franzmann, Sebastian Harl, Martin Hoffmann, Fabian Kampfmann, Sebastian Kotulla, Torsten Müller, Markus Götze, Tobias Klaus, Jens Schedel, and Adriaan Schmidt for implementing parts of this project. This research is partially supported by Siemens Corporate Technology, Munich under the CoSa grant.

9. REFERENCES

[1] S. Bouabdallar, A. Noth, and R. Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJInt. Conf. on Intelligent Robots and Systems*. IEEE, 2004.

[2] S. S. Craciunas, C. M. Kirsch, H. Röck, and R. Trummer. The JAviator: A high-payload quadrotor UAV with high-level programming capabilities. In *AIAA Guidance, Navigation and Control Conference*, pages 1–21, 2008.

[3] J. Eckert, F. Dressler, and R. German. Real-time Indoor Localization Support for Four-rotor Flying Robots using Sensor Nodes. In *IEEE Int. W'shop. on Robotic and Sensors Environments*, pages 23–28. IEEE, 2009.

[4] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante. *Software-Implemented Hardware Fault Tolerance*. Springer, Secaucus, NJ, USA, 2006.

[5] G. M. Hoffmann, H. Huang, S. L. Wasl, and E. C. J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *AIAA Guidance, Navigation, and Control Conference*, pages 1–20, 2007.

[6] Infineon Technologies AG, St.-Martin-Str. 53, 81669 München, Germany. *TC1796 User's Manual (V2.0)*, July 2007.

[7] F. Jahanian and A. K. Mok. Modechart: A specification language for real-time systems. *IEEE TOSE*, 20(12):933–947, 1994.

[8] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, 1997.

[9] D. Lohmann, W. Hofer, W. Schröder-Preikschat, J. Streicher, and O. Spinczyk. CiAO: An aspect-oriented operating-system family for resource-constrained embedded systems. In *2009 USENIX ATC*, pages 215–228, Berkeley, CA, USA, June 2009. USENIX.

[10] P. Pounds and R. Mahony. Design principles of large quadrotors for practical applications. In *2009 IEEE Int. Conf. on Robotics and Automation*, pages 1325–1330. IEEE, 2009.

[11] P. Pounds, R. Mahony, and P. Corke. Modelling and control of a quad-rotor robot. In *2006 Australasian Conf. on Robotics & Automation*, pages 1–10, 2006.

[12] U. Wappler and M. Müller. Software protection mechanisms for dependable systems. In *Conf. on Design, automation and test in Europe*, pages 947–952. ACM, 2008.