A Flexible Real-Time Control System for Autonomous Vehicles.
Johannes Meyer, Armin Strobel
Institute of Flight Systems and Automatic Control, Technische Universität Darmstadt, Germany

Abstract

In this paper we present a framework for the real-time control of lightweight autonomous vehicles which comprehends a proposed hard- and software design. The system can be used for many kinds of vehicles and offers high computing power and flexibility in respect of the control algorithms and additional application dependent tasks. It was originally developed to control a small quad-rotor UAV where stringent restrictions in weight and size of the hardware components exist, but has been transferred to a fixed-wing UAV and a ground vehicle for in- and outdoor search and rescue missions. The modular structure and the use of a standard PC architecture at an early stage simplifies reuse of components and fast integration of new features.

1 Introduction

In recent years the interest in autonomous unmanned vehicles has grown significantly, taking into account various applications. The spectrum reaches from surveillance and reconnaissance [1, 2], environmental monitoring [3], collection of geospatial data [4] to the support of search and rescue forces in disaster mitigation and prevention [5, 6]. Depending on the situational requirements either unmanned ground vehicles (UGV), aerial vehicles (UAV), surface vehicles (USV) or even underwater vehicles (UUV) or a mix of these are considered. The usage of unmanned vehicles has a high potential in reducing the risks for humans in hazardous scenarios and helps to save time and money compared to existing approaches. This especially holds for autonomous systems that are able to accomplish a predefined mission or at least subtasks without human interaction.

From a system engineer’s perspective many challenges need to be tackled when designing and implementing such a system. Even though many disparities exist when looking at the mechanical design, propulsion, sensors and actuators of different type of vehicles, some problems are conspicuously similar. Basic prerequisites for an autonomous vehicle are the ability to localize itself, to build an internal model of what happens in the environment based on sensor readings, to plan its future steps and to control the actuators effectively. All of these components need to play together in a robust and reliable way and in most cases time constraints exist to a greater or lesser extent. Another important aspect that needs to be addressed is the communication between the vehicle and an operator station or between different vehicles.

In this paper we present a framework for the design of such a control system. It was originally developed as an onboard controller for a small quadrotor aircraft of approximately 1.2 kg takeoff weight (Fig. 1) but soon has been adopted for the integration into other vehicles as well. In the meantime it has successfully been deployed in a fixed-wing airplane and a ground vehicle used for search and rescue missions. As opposed to specialized microcontroller-based designs we decided to come up with a solution, that relies on a commercial off-the-shelf onboard computer with PC architecture at a very early stage in the control process. Only basic sensor and actuator I/O is done by a microcontroller which communicates with the computer using an Ethernet connection. The choice of this link technology was motivated by the high availability of single board computers

Figure 1: Quadrotor UAV controlled by the proposed system

---

http://www.fsr.tu-darmstadt.de/
equipped with that interface and because of its high bandwidth and real-time capability, when not having to cope with packet collisions. A modular software design makes it easy to replace only some of the components or simply load another configuration file for obtaining a different system behavior, e.g. when moving to another vehicle. Many existing UVs are equipped with onboard computers as well, but we know no other projects that integrate it in the lowest control loop on platforms equally sized than ours.

Limitations related to payload, power consumption and timing constraints have to be taken into account when designing an onboard control system. The developer has to find a suitable trade-off between the increased capabilities and flexibility and those limiting factors. Weight and size limitations are most important on small and lightweight vehicles, especially aerial vehicles. Lighter systems usually have less accurate sensors and possess less computing power. Also the vehicles have to carry batteries for energy supply with them leading to an additional increase of weight for each new piece of hardware and reduction of available runtime. However, during the last years there has been substantial progress on the market of small and lightweight single board computers which consume little energy and can be used as capable onboard computers. Another frequently underestimated parameter is the tolerable transport latency in the control loop. Small vehicles usually have faster dynamics and higher eigenfrequencies and therefore latencies of more than a few milliseconds can have a negative impact on control quality and even cause instabilities for some systems.

Onboard systems for the control of unmanned vehicles are a well researched area. While larger systems like the cars used for the DARPA Urban Challenge with less constraints in size and weight often use off-the-shelf components e.g. from industrial automation [7], specialized autopilot solutions exist for small-scale UAV and model aircrafts [8]. They have sensors, processors and peripheral circuits integrated into one single board and control laws are more or less fixed except for parameter tuning. The integrated functions are usually restricted to waypoint navigation, attitude and air speed hold and some support auto-takeoff and landing. A bunch of middleware solutions for mobile robots exists that run in a Java virtual machine or as native tasks in Unix-based systems [9, 10], but none of them supports hard real-time operation and they instead rely on underlying controllers implemented in hardware.

In the following section we give a short overview over the proposed system structure and the different target vehicles. In section 3 the hardware components and their interaction will be described and section 4 introduces the associated modular software framework. Afterwards some results are given in section 5 and the paper is concluded in section 6.

2 System Overview

In contrast to most existing solutions, which are purely based on microcontroller boards, our main goal was to come up with a hard- and software framework that simplifies the realization of new navigation and control algorithms while at the same time offers enough computing power for the additional onboard processing of computer vision, obstacle avoidance, mapping and cooperative control schemes. With the choice of a capable onboard PC and Linux as operating system together with a real-time enabled kernel this leads to a powerful solution to this problem. Unfortunately, most available embedded PC boards do not incorporate common sensor interfaces like analog ports, SPI and I²C busses directly. For this reason we could not eliminate the need for a microcontroller board, which serves as a hardware abstraction for sensors and actuators and interfaces the computer via an Ethernet link. However, this does not imply huge constraints in respect to flexibility, as the interface communication is very abstract and all the processing is done by the onboard computer.

2.1 Target Platforms

2.1.1 Quadrotor

The first platform is a quadrotor helicopter developed at TU Darmstadt (Fig. 1). Quadrotors are able to take off and land vertically and can hover at a fixed position without rudders or pitch-control, which motivates their application for search and rescue missions [11]. The propulsion system using four independently controlled motors and propellers allows the carriage of comparatively heavy payloads. Our airframe can carry up to 500g of cameras and other sensors and has a total weight of 1200 - 1400g including the control system and batteries for an endurance of approximately 20 minutes. With a diameter of 80cm it can be easily deployed in outdoor missions as well as in indoor scenarios. As the rotational sense of two adjacent drives differ, the moments about all three axes and the total thrust can be controlled independently by simply varying the speed of the individual motors.

For image acquisition a Logitech QuickCam Pro9000 camera is mounted to the quadrotor which transmits video images to the ground station via the onboard computer using the wireless network. Additionally, up to five frames per second are stored on an onboard flash media for after-mission analysis including references to the available navigational data.

2.1.2 Fixed-wing aircraft

Our hardware and software has also been used in a fixed-wing propeller driven airplane. The airframe (Fig. 2) is based on the Graupner Elektro Kadett, an almost ready to fly consumer model with 1.6 m wing span and made of wood. Fully equipped with a brushless motor, a four cell
LiPo battery (4.8 Ah) and our hardware platform it weighs about 2.3 kg.

![Airplane](image1)

**Figure 2: Airplane**

2.1.3 Ground vehicle

The ground vehicle is based on a 1:8 scale R/C Monstertruck model (Fig. 3). It is modified with a four wheel steering and an additional gear to drive slow and increase precision. On top of this chassis a box containing the interface board with the same sensors as on our flying systems and a additional odometers in every wheel. The laserscanners and cameras (one daylight and one thermal camera) are directly connected to a high performance mobile PC unit. This vehicle was used in RoboCup Rescue competition [12] where the goal is to autonomously find victims in a simulated disaster scenario.

![Ground Vehicle](image2)

**Figure 3: Ground Vehicle**

3 Hardware Description

3.1 Computing Unit

The structure of our onboard hardware is shown in Fig. 5. As PC unit nearly every PC- or compatible platform can be used that comes with a supported Ethernet controller. It runs a Xenomai real-time enabled Linux kernel with RTnet as an alternative Ethernet driver [13]. For our quadrotor and airplane we use a small single board computer with Intel Atom Z530 processor in Pico-ITX format (100 × 72 mm). With 400g total weight of all electronic components the system fulfills the restrictions of our flying vehicles. The ground vehicle is equipped with a Mini-ITX Core 2 Duo mainboard and an additional CUDA-compatible GPU, which enables parallel image processing, accurate simultaneously localization and mapping and runs the complete software framework with communication and behavior control.

3.2 Link Technology

Two main issues have to be considered when selecting a link technology for connecting sensors with a SBC, namely
the real-time capability and the achievable bandwidth and latency. Additional issues to consider are the expected availability on small embedded PC units now and in next future and the existence of a suitable driver implementation (Table 1).

<table>
<thead>
<tr>
<th></th>
<th>Ethernet</th>
<th>USB</th>
<th>Serial Ports</th>
<th>SPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time operation</td>
<td>using</td>
<td>no</td>
<td>possible</td>
<td>possible</td>
</tr>
<tr>
<td></td>
<td>RTnet/Xenomai</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical bandwidth</td>
<td>10/100 MBit/s</td>
<td>12/480 MBit</td>
<td>&lt; 1 MBit/s</td>
<td>10 MBit/s and more</td>
</tr>
<tr>
<td>Availability</td>
<td>longterm</td>
<td>longterm</td>
<td>midterm</td>
<td>unknown</td>
</tr>
<tr>
<td>Spreading</td>
<td>very good</td>
<td>very good</td>
<td>often</td>
<td>poor</td>
</tr>
</tbody>
</table>

Table 1: Expected properties of link technologies

We decided to use Ethernet with RTnet/Xenomai as real-time driver implementation on the PC side. MAC controllers are either already integrated in the microcontroller or can be easily connected as peripheral device.

### 3.3 Interface Board

The computing unit is connected to the interface board via Ethernet which sends sensor data on request and receives the output commands for the actuators. The interface is equipped with a Philips/NXP LPC2138 microcontroller, a high-speed 16 bit A/D-converter, a 6 degree of freedom inertial measurement unit (IMU), an ENC28J60 Ethernet controller and additional components for power management. Supplementary sensors and actuators can be connected to the analog inputs or via external ports according to the requirements for the respective vehicle. The available ports include two bidirectional serial ports, a SPI, two I²C connectors and several GPIO ports.

A special expansion board for aerial vehicles is equipped with a barometer, a differential pressure sensor for measuring airspeed and a 3D magnetometer. These devices have been mounted on the expansion board because they are very prone to electromagnetic fields and need to be placed where little interference is expected.

For safety and legal issues we have also connected a module that reads a PPM signal from a standard model aircraft R/C-receiver and has several outputs for servos. This allows to actuate the basic functions of all vehicles directly with a remote control. Due to this the airplane can be flown by a safety pilot even in the case of a total failure of the interface board or PC unit.

The two serial ports on the interface board are usually connected to a GPS receiver and a Maxstream XBee data modem. The data modem complements the wireless network of the PC unit and is used for long range and reliable communication with limited bandwidth in order to transmit status and mission information and commands.

### 3.3.1 Firmware

The interface board initiates sensor measurements on request only and sends the results back to the onboard computer. Which sensor information is requested is determined by a bit field in the request packet. After all sensor data are read, they are packed into corresponding messages and forwarded to the computing unit. For the serial ports the interface board buffers all incoming data and relays it on request. No pre-computation is made on the interface board, e.g. conversion of raw data to the real physical quantities or filtering. However, we plan to sample wide above the cut-off frequency of the IMU sensor and implement a digital filter in the next release.

### 3.3.2 Hardware Modularity

All connected expansion boards, sensors and actuators are identified by the interface board during an initialization phase. Information about available devices can be requested. Various additional sensors are supported by now, as to mention ultrasound range finders, temperature sensors and wheel encoders. On the output side different brushless motor controllers and PWM or RS485 servos are recognized as actuators.

### 4 Software Framework

This section presents details about the software framework in general and some of the components we implemented for autonomous vehicles.

![Software component structure and appropriate data flows](image)

Figure 6: Software component structure and appropriate data flows

It is well known that the control of complex autonomous vehicles requires a structured approach to the design of a control software. Integration of the whole functionality in a monolithic core would exacerbate the maintenance significantly, especially when multiple developers are involved. The Orocos Open Robot Control Software (Orocos) provides the basic functionality for a modular component-base design [14]. Orocos comes with a real-time capable core library, the Real-Time Toolkit (RTT),
and several useful libraries and ready-to-use components. RTT serves as an abstraction layer and uses the concept of components, which are entities with distinct functionality providing data flow ports, attributes, properties, methods, commands and events as external interface. By using RTT the developer does not have to cope with real-time thread management and thread-safe data exchange directly but can implement the required functionality in a more descriptive manner. RTT in general requires a real-time enabled kernel like Xenomai or RTAI as target platform, but also runs in native Linux mode or even on Windows hosts without real-time guarantees.

4.1 General Structure

The currently realized components and their data flow for our quadrotor application are shown in Fig. 6. The structure can easily be transferred to other autonomous vehicles. Every block represents an Orocos component with a well-defined interface which can be composed individually depending on the application. During the initial setup phase the components are connected to each other and assigned to an activity. RTT activities include periodic execution, which are mainly applied in the control loop, and event-triggered or non-periodic execution used for high-level mission control and communication component amongst others. The setup can also change during runtime, e.g. when the control source changes from autonomous guidance to manual control or vice-versa.

The main control loop realizes the essential functionality for the mobility of the vehicle. From the software point of view it is split up into four parts: the hardware interface, sensor abstraction, navigation/modeling and the controller. Each of these parts is represented by a special component. The interface component directly interacts with the system via network communication and driver APIs and provides methods and events for accessing analog inputs and outputs and other external interfaces. There is a strong coupling between this component and the subsequent sensor components which abstract the hardware details on an individual sensor level and provide data flow ports with values converted to meaningful physical quantities. As direct feedback of sensor information is not sufficient for control in many cases an additional modeling component is introduced, which infers non-observable state variables from the sensor readings by using probabilistic models. The controller uses the measured or estimated signals to calculate the commands for the actuators like servos and motors. Obviously the implementation of the controller highly depends on the vehicle’s kinematic and dynamics and needs to be adopted in the case of major changes or transfer to another platform.

On top of the basic control loop a supervisory component consecutively monitors the outputs and reacts on exceptional events by switching to manual control mode or executing an emergency procedure. The guidance component is an aggregate for several different control blocks which are switched depending on the current mission state. This modules generate a nominal trajectory or setpoints for the underlying controller during autonomous flight. The mission manager can execute preplanned complex missions described by a hierarchical state automaton. During manually controlled drive or flight the commands are read from the external R/C receiver in real-time and directly fed to the actuators or the controller.

The communication module collects data from different other components and uses the interface to the radio modem to send a periodic status message to listening agents. Furthermore, the vehicle’s state can be controlled by a set of commands which are forwarded to the autonomous guidance and mission control components. The counterpart of this interface is realized in our ground control software. It uses the same code as the communication component so that new functionality can be introduced at a single point. However, we will not go into details about the ground segment in this paper. Also we currently do not use the CORBA features of Orocos for communication for the sake of compatibility to previous projects.

Depending on the available payload additional components are loaded for interfacing cameras and laser range finders. Until now, these information are not processed in real-time but forwarded to other processes running on the same system. For indoor 2D navigation we implemented the simultaneous localization and mapping algorithm (SLAM) algorithm which has been successfully deployed in a search and rescue scenario in the RoboCup Rescue competition. The robot is able to find possibly injured humans and signs of hazardous materials autonomously using an object detector based on histograms of oriented gradients [15, 16] and evidence from the thermal camera.

4.2 Component description

4.2.1 Hardware Interface

The hardware component sends a request packet to the microcontroller board periodically. For each sensor the user can configure an individual rate divider so that only relevant data is transmitted in each timestep. The received information are forwarded to the sensor components for further processing. In most cases this includes transformation of raw values to SI units and coordinate transformations. Other modules, like GPS or the radio modem directly communicate with the connected devices and the serial data streams are forwarded over the Ethernet link. As soon as the controller has finished its processing step an event is emitted and the interface component writes out the new actuator commands immediately.

For networking RTnet is used as a real-time protocol stack which seamlessly integrates into Xenomai/RTAI. RTnet supports many popular NIC chipsets. As only one master and one slave device is present on the cable there is no need for a special media access control like in other applications of real-time networking.
4.2.2 Modeling and Navigation

For most vehicles self-localization is the most important modeling step. An Extended Kalman Filter (EKF) estimates the 3D position, velocities and Euler angles given the IMU sensor data according to the strapdown algorithm [17, 18]. Additional sensor information is needed to prevent the navigation solution from drifting. For outdoor operation, the GPS receiver is used as a source for the approximate position and velocities in the earth-fixed reference frame and the earth magnetic field serves as heading reference. The height estimation is further improved by pressure measurements from the barometric sensor. For indoor localization, positional and directional feedback is delivered by the external SLAM module using a 2D laser range finder.

4.2.3 Controller

As mentioned above, each type of vehicle requires an individual structure of the controller. Our quadrotor uses a set of four PID controllers for stabilization and controlled flight, namely one for each Euler angle and one for the vertical speed, whose outputs are superposed afterwards to come up with an overall motor command. Further cascades can be activated on demand that control the horizontal speed, the position and the height of the vehicle. The other vehicles use simpler approaches, e.g. a linear speed controller and a steering angle control for directional control in the case of our unmanned ground vehicle.

An interesting alternative to the direct implementation in C++ is the graphical development of controller components. The Orocos Simulink Toolbox uses MathWorks Real-Time Workshop to generate Orocos compatible code from models in Simulink. This approach allows even users with poor programming skills to experiment with different control concepts and parameter tuning. We use this tool extensively together with a quadrotor testbed for research and education.

4.2.4 Autonomy and Mission Control

An autonomous mission is defined by a sequence of basic mission elements, which also can contain loops and conditional branches. The current mission state is controlled by the mission manager that basically implements a finite-state machine and permanently checks whether the preconditions for a transition are fulfilled or an exception occurred. Missions are loaded from a simple XML file or from models in Simulink. This approach allows even users with poor programming skills to experiment with different control concepts and parameter tuning. We use this tool extensively together with a quadrotor testbed for research and education.

4.2.5 Communication

The communication component provides the external interface of the control system. We use a message-based binary protocol based on the UBX protocol [19] known from GPS chipsets by u-blox AG. Every message is tagged with a timestamp and an identifier for the source and destination. A standard IP connection or the radio modem connected to the interface board are available as underlying transport layer. The communication component continuously broadcast status messages and reacts on a set of known commands. In a future step we plan to extend modularity by providing methods for the online registration of commands, so that each component can define its own external interface without touching the communication.

4.2.6 Logging and Replay

The logging component permanently monitors the data flow and writes a binary log file to a flash media. This dataset can be imported into Matlab for diagnostic purposes in the case of failures or debriefing after the mission. When running the software in a special replay mode, data is reinjected into the system from the log file in real-time, which simplifies debugging and testing without real hardware or sensor data available.

5 Results

The vehicles introduced in section 2 could successfully be controlled by the proposed system. Obviously, the quadrotor makes highest demands on real-time performance and behaves critical if latencies are too high. However we could achieve good flight performance even under harsh wind conditions. The control loop runs at 200 Hz with average latencies of 1.2 ms including data exchange with the microcontroller board, navigation filtering, control and logging. The onboard computer uses approximately 5 percent of CPU time, so that plenty of room remains for other tasks. The quadrotor gained the 1st place at the Outdoor Autonomy Competition of the European Micro Air Vehicle conference taken place in the Netherlands in late 2009. The system demonstrated its flexibility when transferred to the fixed-wing UAV and only the flight controller had to be adopted to the new environment. By resorting to Simulink and Real Time Workshop it cost no more than a few minutes to compile a previously implemented model, which was tested in a simulation environment.

Some small modifications have been made for the ground vehicle where in addition to the inertial sensor data a velocity feedback from odometry is used as an observable parameter and integrated into the Kalman filter. For operating indoor without GPS coverage, position and orientation information are delivered from the SLAM algorithm.
running in an external module. Beside of that the overall hard- and software structure remained the same.

6 Conclusion

In this paper we presented a control system which brings the advantages of a full-featured development environment and robotic middleware software down to small-scale systems and is appropriate to control very heterogeneous types of autonomous vehicles. The integration of all software components from low-level control to high-level autonomy and mission planning in a single framework allows fast development cycles and simplifies debugging and offline testing. Especially the navigation, mission management and communication components have proven to be easily transferable to different platforms.

We plan to make our code available to the public in near future. Also we work on interfaces to other robotic middleware suites like ROS [20] or Player/Stage [21]. These frameworks are supported by strong communities and a broad spectrum of ready-to-use tools and components exist.

Acknowledgements

This work has been funded, in part, by GRK 1362 of the German Research Foundation (DFG).

References


