COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle

Christian Berger, Al Mamun Md Abdullah, Jörgen Hansson

COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle
(Invited Paper)

Christian Berger, Md Abdullah Al Mamun, and Jörgen Hansson

Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden
[christian.berger,abdullah.mamun,jorgen.hansson]@chalmers.se

Abstract. The international competition 2007 DARPA Urban Challenge significantly fostered the research and development on intelligent vehicles. Recent prototypes from Google for instance are able to drive fully autonomously through the San Francisco bay area for more than 450,000km. Even if self-driving vehicles are not yet available for purchase for customers, technology spin-offs are increasingly finding their ways into today's vehicles as pedestrian collision warning system, automated parking, or adaptive cruise control with temporary lane following. Thus, the upcoming generation of software and computer engineers needs to be accordingly educated to be already familiar with the concepts and pitfalls of these complex systems. In this article, we describe the architecture of a self-driving miniature vehicle based solely on commercial-off-the-shelf (COTS) components for the hardware and the real-time operating system ChibiOS/RT for the software/hardware interface.

Keywords: COTS, self-driving vehicle, real-time OS, software engineering

1 Introduction

The 2007 DARPA Urban Challenge facilitated the research and development of self-driving cars and safety systems in today's vehicles [1]. Even if the technology of recent prototypes is still not non-intrusive due to the fact of prominently visible sensors as known from the Google cars, spin-offs of this technology with dedicated sensors are nowadays already used to protect passengers, pedestrians, and bicyclists [2]. Thus, upcoming computer and software engineers are faced with challenges from these systems in industry after their studies.

In the higher education, these developments and industrial demands need to be considered to prepare these students accordingly. Thus, in the 2013 project course for 2nd year students from the bachelor program of "Software Engineering and Management" of the Chalmers — University of Gothenburg, a self-driving miniature vehicle at 1:10 scale was realized over a period of three months. Under the supervision of the authors, the goal for the students was to adapt a given sensor architecture and to design algorithms for a self-driving miniature vehicle, which should be able to reliably follow lane markings, overtake stationary obstacles, and park automatically.

The research question, which should be answered by applying design-research in that course, was to explore the possibility to realize a self-driving miniature vehicle that is designed based solely on commercial-off-the-shelf (COTS) components. Hereby, the term COTS does explicitly not only refer to hardware components but also the software/hardware interface to read data from sensors and to control the actors for steering and acceleration.

The rest of the paper is structured as follows: Firstly, an overview of the sensor layout and hardware architecture is presented followed by a description of the incorporation of the real-time OS ChibiOS/RT, which was evaluated as software/hardware interface. Afterwards, a short overview of the development process and the evaluation of the software is presented.

2 Related Work

In this section, we focus on currently available frameworks for the development of component-based embedded automotive and robotics software.

AUTOSAR (AUTomotive Open System ARchitecture) is a standardized architecture for automotive software systems. It was developed with a focus to standardize in-vehicle electronic control units (ECU), software interfaces modules, and the workflow to create and maintain them. AUTOSAR is a multi-layer architecture where it resides directly on top of a micro controller to abstract the ECU and to provide a runtime environment for the software components at the application layer. AUTOSAR primarily focuses on the system development process for ECUs and does not provide specific support for sensor/actuator-based autonomous system development [3]. An implementation of an AUTOSAR-compliant embedded software platform is Arctic Core [4] from ARCCORE AB, which offers a real-time OS, memory services, and communication services such as CAN and LIN. It supports various micro controller architectures like Freescale MPC5xxx, HCS12, and ARM-Cortex-M3/R4.

OpenJAUS is an open source implementation of DARPA's Joint Architecture for Unmanned Systems (JAUS) for developing autonomous air, ground, water, and subsurface systems [5]. The messaging system of the OpenJAUS comprises of interconnected remotely running node managers, which interchange messages. JAUS was also used by teams in the 2007 DARPA Urban Challenge like the entry from Virginia Tech [6]. In contrast to real-time OS, it focuses on the high-level system and communication architecture.

For robotic applications, several component-based frameworks have emerged like Orca [7], the Carnegie Mellon Robot Navigation Toolkit (CARMEN) [8], and Middleware for Robotic and Process Control Application (MiRPA) [9], which focuses on real-time communication for distributed embedded systems. However in recent years, the Robot Operating System (ROS) [10] incorporates the simulation environment Player/Stage/Gazebo [11], and it has been widely adopted as a meta-OS for many popular robotic platforms. The abstraction of physical sensors and actuators enables ROS to support a wide range of robotics platform and reuse of components. However, it requires another OS like Ubuntu Linux or Windows where it runs on top and thus, it cannot be used to interface with the hardware on highly embedded systems.

The results outlined in this article are a continuation of the work previously described by Berger et al. [12]. In contrast to that work, the design and realization, which is described here, focuses solely on COTS components both on the hardware architecture but also on the software/hardware interface in terms of a real-time OS. The followed principles throughout the design and realization of the entire system and its software were previously elaborated here [13].

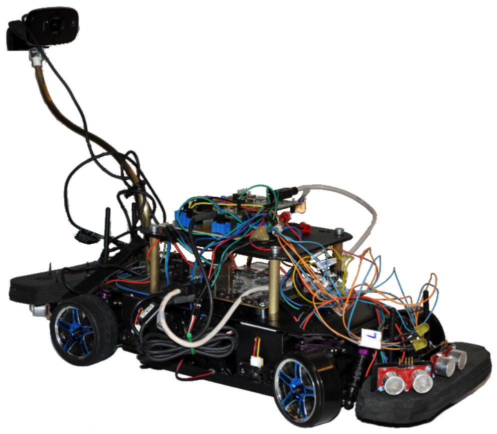
3 COTS-Architecture for the Hardware

3.1 Overall Architecture

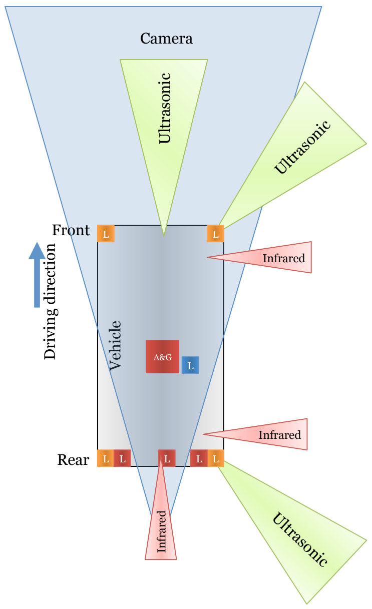
Various hardware components e.g., sensors, devices, and boards are used for a self-driving miniature vehicle. When selecting the COTS components, the general selection criteria were efficiency, low-power consumption, low-price, availability and extendability. All sensors and actors and their connections are described in this section. An overview is also available in Appendix A.

Fig. 1 shows one developed self-driving miniature vehicle from the bachelor course together with the layout of different sensors and camera setup that was realized in the vehicle. In order to detect obstacles, the vehicle is equipped with three infrared sensors (SHARP GP2D120) and three ultrasonic sensors (Devantech SRF08). A webcam (Logitech c525) is used for lane marking detection. The layout of the sensors and webcam is shown in Fig. 1(b). A brushless motor and servo are used to drive and steer the vehicle. An electronic speed controller (ESC) is used to control the motor and the steering. The Razor-9DoF-IMU board (Razor Board) is used to measure accelerations and angular velocities. It comprises a triple axis accelerometer (ADXL345), single/dual axis gyros (LY530ALH and LPR530ALH), and a triple-axis magnetometer (HMC5843). The output of the sensors is already processed onboard and are accessible through a standard serial connection.

An STM32F4 Discovery Board is used to process sensor data and to interface with the actors. An ARM-based PandaBoard ES is used to run the vehicle application software. These two boards are further described in the following. Fig. 2 shows how the different sensors, actors, and boards are connected.



(a) Self-driving miniature vehicle based on COTS components.



(b) Sensor layout of the self-driving miniature vehicle.

Fig. 1. One self-driving miniature vehicle from the 2^{nd} year bachelor course of the program "Software Engineering & Management" at Chalmers | University of Gothenburg.

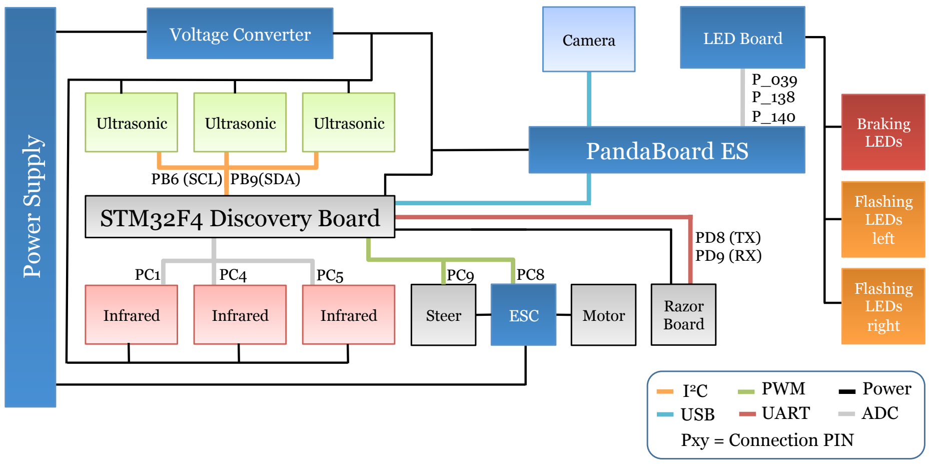


Fig. 2. Schematic diagram of the hardware architecture including the connection plan for the STM32F4 Discovery Board of the self-driving miniature vehicle.

3.2 Application Board

An ARM-based PandaBoard ES is used as the platform to run the applications developed for the self-driving miniature vehicle. It is a low-cost and low-power single-board development platform based on the OMAP4460 System-on-Chip from Texas Instruments. The OMAP4460 has a dual-core ARM Cortex-A9 MPCore with up to 1.2 GHz each, 1 GB low-power DDR2 RAM, dedicated POWERVR SGX540 graphics core, which supports the major APIs (OpenGL ES, OpenVG, and EGL), HDMI and DVI connectors for displays, Bluetooth and 802.11b/g/n wireless connectivity, HDMI audio out, SD/MMC card slot, onboard 10/100 Ethernet, and various expansion slots including camera, LCD, USB, MMC. Furthermore, it also features general purpose input/output (GPIO), which is used to control brake and flashing lights with LEDs in a self-assembled LED driver board. The PandaBoard ES is powered externally by 5V and runs the Ubuntu Linux 11.10 operating system for the ARM platform.

3.3 Hardware/Software Interface Board

The STM32F4 Discovery Board is used as the interface between the sensors and actors as mentioned in Sec. 3.1 and the PandaBoard ES. It runs the ChibiOS/RT real-time OS to schedule its processes in real-time for handling sensors and to interface with the actors.

The STM32F4 is a member of the STM32 family of micro controllers that are based on the RISC ARM-Cortex cores. The STM32F4 contains a STM32F407VGT6 micro controller featuring a high performance ARM Cortex-M4 core at the frequency of up to 186 MHz with 1 MByte flash, DSP with a FPU, an adaptive real-time accelerator and Ethernet. The key reasons for using STM32F4 Discovery Board are its cheap price combined with its high manufacturing quality, its low-power consumption, its extensibility through many GPIO pins, and the standard connection through USB to the PandaBoard ES.

4 COTS-Architecture for the Real-Time Software with ChibiOS/RT

ChibiOS/RT [14] is used to realize the scheduling and execution of the code for the embedded system to read and process sensor data and to interface with the steering and acceleration motors. It is not merely a scheduler but a whole real-time OS, which is designed for deeply embedded systems. ChibiOS/RT is open source and features a wide set of kernel primitives. It is lightweight, efficient in its implementation, and offers concepts like threads and mutexes to enable concurrent programming. It has been successfully used for different applications in the embedded domain for example to control a quad-copter [15].

It has a portable preemptive kernel and a hardware abstraction layer for different embedded boards, which enables the reusability of software components in the case of exchanging a board for example. It also supports a wide variety of CPU architectures (ARMv6_M-ARMv7_ME, ARM7, MegaAVR, MSP430, STM8, and Power Architecture e200z), compilers (GCC, RVCT, IAR IAR, Cosmic, and Raisonance) and platforms (AT91SAM7x, AVR, LPC11/13xx, LPC214x, MSP430, SPC56x, STM32F0-F4xx, and STM32L1xx).

On the STM32F4 Discovery Board, ChibiOS/RT requires only 0.40μs for a context switch and the entire software, which is used on the self-driving vehicle on this board, weights only 49kByte. An overview of the software component size is provided in Table 1.

Component	Size (LoC)	Purpose
Sensors		
ACC	69	Read internal accelerometer sensor.
IMU	254	Read Razor-9DoF-IMU board and IMU filter algorithms.
IR	81	Read infrared sensors.
US	163	Read ultrasonic sensors.
Actors		
Motor	116	Interface to the acceleration and steering motors.
Misc		
Conf	414	ChibiOS/RT board configuration header files.
Protocol	284	Protocol encoder/decoder for PandaBoard communication.
UI	198	Interactive user interface.
USB	215	Realizing serial-over-USB connection.
main.c	27	System initialization.
Total	1,821	

Table 1. Overview of the software component size for the various parts of the embedded software from the self-driving miniature vehicle running on ChibiOS/RT.

The overall high-level system architecture containing key hardware and software components for the developed self-driving miniature vehicle is shown in Fig. 3.

5 Development and Evaluation of Self-Driving Software

From a software engineer's point of view the utilization of COTS to let software interface with its reality enables the developers to focus mainly on the application layer of the system. Therefore, the software design followed two main goals: 1) Design, realize, and evaluate the actual driving behavior as the application, and 2) ensure reusability for future versions of self-driving miniature vehicles.

The first goal was realized based on the experiences described in [12]: The students received at the beginning of the course a pre-installed and pre-configured

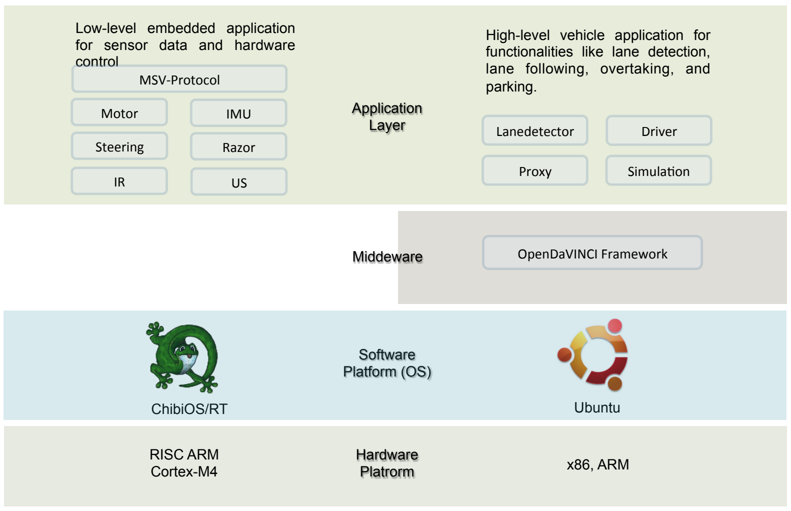


Fig. 3. High-level system architecture of the self-driving miniature vehicle.

development environment based on Ubuntu Linux 11.10 in a VirtualBox virtual machine. In that environment, the simulation components from the distributed middleware "OpenDaVINCI" as shown in Fig. 4 were also provided together with a set of scenarios. The simulation consisted of the components `vehicle` for simulating the vehicle motions, `camgen` to generate images from a virtual camera, and `irus` for providing relative distance data from virtual infrared and ultrasonic sensors.

These components were used to provide both virtual input data to the component `lanedetector` and to have a feedback loop for the control commands sent from the component `driver`. Thus, a validation of first algorithmic ideas was possible without having the real vehicle completely assembled. Furthermore, due to the usage of the pre-configured virtual machine, the students could directly start with making themselves familiar with the simulation environment without spending valuable time in just setting up the right OS and libraries.

In Fig. 5, an example for a scenario is depicted that was used to evaluate the software. With scenarios like these, the application software from the students was evaluated for a valid driving behavior for lane following[1], automated parking on a sideways parking strip [2] and during overtaking maneuvers[3].

The second goal to ensure reusability of the software both on the real miniature vehicle but also with future hardware environments was achieved by two architectural means: Firstly, the interface for the aforementioned two application components `lanedetector` and `driver` was realized transparently between the

[1] Video of the lane following algorithm: http://youtu.be/lYlfy3tB93w
[2] Video of the autonomous parking algorithm: http://youtu.be/e0hgjo-8XL4
[3] Video of the overtaking algorithm: http://youtu.be/zJVEWaoT3t8

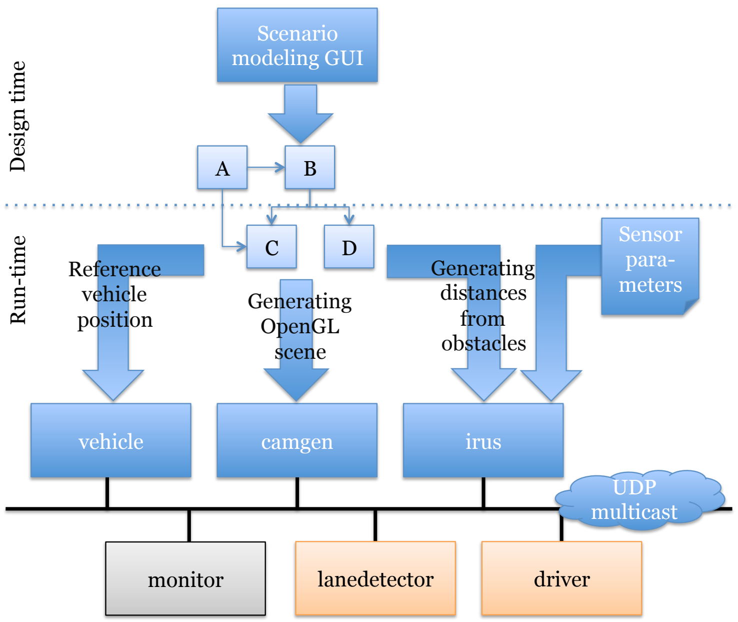


Fig. 4. Architecture of the components in the simulation environment.

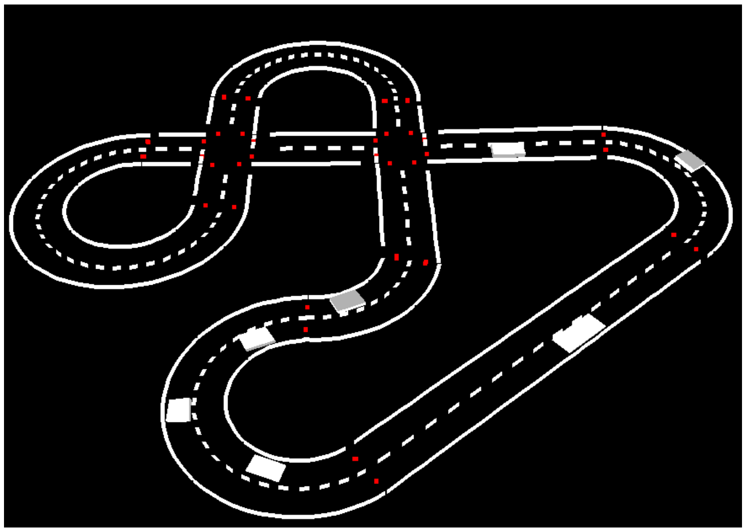


Fig. 5. Model of the test track in the simulation environment for the overtaking scenario.

simulation environment and the final hardware platform on the PandaBoard ES so that both components could be reused without any further changes. Hereby, the additional component `proxy`, which is already previously shown in the high-level software architecture by Fig. 3, converts the commands from `driver` into the protocol that is used between the PandaBoard ES and the STM32F4 Discovery Board and maps the measured distances from the infrared and ultrasonic sensors to the data structure used in the application.

Secondly, the algorithms that are realized with ChibiOS/RT rely only on the hardware abstraction layer from that OS and thus, can be reused by any board, which is supported by the real-time operating system. Thus, future evaluations with other embedded systems that are either more powerful and energy-efficient or have more pins to connect more sensors are easily possible by remapping the connection pins in the protocol encoder/decoder.

Thus, the designed software architecture enables a seamless transferability of the application from the simulation to the hardware or from one existing hardware environment to a new platform.

6 Conclusion and Outlook

This paper presents the hardware and software architecture of a self-driving miniature vehicle based entirely on COTS components. In order to get immediate feedback on the partially developed software before the vehicle hardware was ready to run, the simulation environment "OpenDaVINCI" was used from the very beginning of the software development. The seamless transferability of the vehicle software from the simulation to the hardware and early feedback from the simulation environment helped in speeding-up the project development. Our experience from this project reflects that such an approach can significantly reduce the development time and thus, enables the development of an experimental self-driving miniature vehicle in only three months.

The choice of COTS hardware and software components has strategic values for the long term evaluation of self-driving miniature vehicles. Such vehicles highly depend on different sensors continually reading real-time data from their surroundings where the environment can be dramatically different in different geographical locations. Factors like traffic laws/signals, road traffic orientation (left vs. right), road structure (single-lane vs. multi-lane, availability of lane divider, etc.), and obstacle type (static, dynamic) would require that a vehicle uses specific types of sensors in specific mounting positions on the vehicle. The high variability of such factors requires a thorough selection of the COTS components for a good product line architecture for self-driving vehicles.

The selection of the STM32F4 Discovery Board and the used real-time OS has significant value from the architectural point of view. In order to achieve extendability and variability of the sensors used in the vehicle, the STM32F4 Discovery Board offers many pins and the possibility of using different types of sensors.

On the other hand, the real-time OS should support hardware abstraction from embedded boards to enable reusability of software components when hardware platforms are changed. Moreover, the real-time OS should be very efficient in terms of resource usage and runtime. The context switching performance is important because the number of sensors can be high and we need to read data from different sensors continuously in real-time. ChibiOS/RT fulfills all these criteria while providing a thoroughly designed API for developers. The features of this real-time OS enable to achieve many quality criteria for the system e.g., performance, reusability, changeability, and maintainability, which have significant value from the software engineer's perspective.

During the realization of the design for the self-driving miniature vehicle, it was apparent that maintaining, debugging, and testing of the software in ChibiOS/RT to interface with the sensors and actors is error-prone, time-consuming, and tedious. Since sensor types and their layout can vary with changes in the surrounding environment of the vehicle and also with the increased functionalities of the vehicle, it is worthwhile to adopt a model-based approach to maintain the embedded software for the STM32F4 Discovery Board running on ChibiOS/RT. Hence, one of our future goals is to automate the generation of verified code from a sensor/actor model for the ChibiOS/RT environment to further save development time.

Acknowledgments

We would like to thank the various student groups who worked with the development of self-driving miniature vehicle. We specifically like to acknowledge our students Dur Abuzaid, Leila Keza, David Ustabowski, Sarah Jamil, Zlatan Habul, Scott Titang, Michal Maciej Dzienis, Maksim Smirnovs, Husam Abdulwahhab, Dionysios Papathanopoulos, and James Olatunde Omoya who developed the vehicle in the bachelor course supervised by the authors. Furthermore, students from CRF provided valuable support during the hardware assembly process. This project is associated with the Gulliver Test-Bed (http://www.gulliver-testbed.net) for miniature vehicles.

References

1. Rauskolb, F.W., Berger, K., Lipski, C., Magnor, M., Cornelsen, K., Effertz, J., Form, T., Graefe, F., Ohl, S., Schumacher, W., Wille, J.M., Hecker, P., Nothdurft, T., Doering, M., Homeier, K., Morgenroth, J., Wolf, L., Basarke, C., Berger, C., Gülke, T., Klose, F., Rumpe, B.: Caroline: An Autonomously Driving Vehicle for Urban Environments. Journal of Field Robotics **25**(9) (September 2008) 674–724
2. Berger, C., Rumpe, B.: Autonomous Driving - 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System. In Goltz, U., Magnor, M., Appelrath, H.J., Matthies, H.K., Balke, W.T., Wolf, L., eds.: Proceedings of the INFORMATIK 2012, Braunschweig, Germany (September 2012) 789–798
3. Hošek, P., Pop, T., Bureš, T., Hnětynka, P., Malohlava, M.: Comparison of component frameworks for real-time embedded systems. In Grunske, L., Reussner,

R., Plasil, F., eds.: Component-Based Software Engineering. Number 6092 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (January 2010) 21–36

4. ARCCORE AB: Arctic Core – Open Source AUTOSAR Embedded Platform. http://www.arccore.com/products/arctic-core/ (June 2013)
5. OpenJAUS LLC: OpenJAUS. http://www.openjaus.com (July 2012)
6. Bacha, A., Bauman, C., Faruque, R., Fleming, M., Terwelp, C., Reinholtz, C., Hong, D., Wicks, A., Alberi, T., Anderson, D., Cacciola, S., Currier, P., Dalton, A., Farmer, J., Hurdus, J., Kimmel, S., King, P., Taylor, A., Covern, D.V., Webster, M.: Odin: Team VictorTango's Entry in the DARPA Urban Challenge. Journal of Field Robotics **25**(9) (September 2008) 467–492
7. Makarenko, A., Brooks, A., Kaupp, T.: Orca: Components for Robotics. In: International Conference on Intelligent Robots and Systems, Beijing, China (2006) 163–168
8. CARMEN: CARMEN - Robot Navigation Toolkit. http://carmen.sourceforge.net/ (June 2013)
9. Finkemeyer, B., Kröger, T., Kubus, D., Olschewski, M., Wahl, F.: MiRPA: Middleware for Robotic and Process Control Applications. (2007)
10. Willow Garage: ROS - Robot Operating System. http://www.ros.org/ (June 2013)
11. Gerkey, B.P., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proceedings of the 11th International Conference on Advanced Robotics, Coimbra, Portugal (2003) 317–323
12. Berger, C., Chaudron, M., Heldal, R., Landsiedel, O., Schiller, E.M.: Model-based, Composable Simulation for the Development of Autonomous Miniature Vehicles. In: Proceedings of the SCS/IEEE Symposium on Theory of Modeling and Simulation, San Diego, CA, USA (April 2013)
13. Berger, C., Rumpe, B.: Engineering Autonomous Driving Software. In Rouff, C., Hinchey, M., eds.: Experience from the DARPA Urban Challenge. Springer-Verlag, London, UK (2012) 243–271
14. Sirio, G.D.: ChibiOS/RT. http://www.chibios.org/ (June 2013)
15. Berkenbusch, M.K.: Quad with optical flow position hold. http://www.diydrones.com/profiles/blogs/quad-with-optical-flow (June 2013)

A Overview of Components

Component	Component ID	Voltage [V]	Electric current [A]	Connection to the Discovery-Board/Panda Board	Update frequency
RC Vehicle	1/10 scale electrically-driven on-road vehicle				
Infrared Sensor	SHARP GP2D120	5V	3×33=99mA	ADC	33ms - 57ms
Ultrasonic Sensor	Devantech SRF08	5V	3×15=45mA	I^2C	65ms
Camera	Logitech c525	5V		USB	40fps
Steer/Servo		5V		PWM	
ESC	1/10 Brushless ESC	7.2V		PWM	10ms
Motor	1/10 Brushless Motor	7.2V			
LED Board	Self-assembled			ADC	
Razor Board	Razor-9DoF-IMU	3V	25mA	UART	10ms
Discovery Board	STM32F4-Discovery	3V		USB	
Application Board	PandaBoard ES			USB	
RC-Handset	A3-STX Deluxe F.H.S.S	12V		2.4GHz	
RC Receiver	A3-RX Deluxe F.H.S.S	5V		2.4GHz	

Table 2. List of the COTS hardware components for the vehicle.