Real-time system architecture design practices

Andrey Getmanskiy, Semen Sechenev, Mikhail Nenyukov, Igor Ryadchikov, Alexander Gusev, Dmitry Sokolov

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2020) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

14th International Symposium "Intelligent Systems", INTELS'20, 14-16 December 2020, Moscow, Russia

Real-time system architecture design practices

Andrey Getmanskiy[a], Semen Sechenev[a], Mikhail Nenyukov[b], Igor Ryadchikov[a], Alexander Gusev[a], Dmitry Sokolov[c,*]

[a] *Kuban State University, Krasnodar, Russia*
[b] *JSC Kalashnikov Concern, Krasnodar, Russia*
[c] *Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France*

Abstract

In this paper we give an overview of both hardware and software architectures of real time systems used in devices for various purposes — from lab bench contraptions to cars. The goal of the paper is to reveal separate classes of such architectures as well as to define preconditions for choosing a particular architecture.

Peer-review under responsibility of the scientific committee of the 14th International Symposium "Intelligent Systems".

Keywords: real-time systems; hardware; software

1. Introduction

Due to the intensive growth of applications of mobile robotic platforms in recent years, the development of on-board real-time control systems is coming to the fore. Increasingly, open-source robotics (OSR) platforms are the foundation for development, providing unified blueprints, schematics, software, and infrastructure to implement the final ideas of third-party developers. Like in the case of a non-anthropomorphic robot we are currently developing in the Laboratory of Robotics and Mechatronics of the Kuban State University (refer to Fig. 1). Thus, the task of selecting the architecture of the on-board control system from a large variety of ready-made alternatives comes to the fore, in accordance with the specific operating conditions of the mobile platform. The development of criteria and the construction of a methodology for selecting the architecture of the on-board real-time systems, thus, are an urgent research task.

In this paper we propose an overview of the hardware and software solutions for a selection of representative samples that cover very different areas varying from laboratory test benchs to electric cars, ATVs performing "aggressive maneuvers", humanoid robots, four-legged walking robots and manipulators. The idea is to reveal main architecture classes, summarize the identified strengths and weaknesses of the most popular solutions. This allows us to identify the main issues affecting the choice of a particular architecture.

So, let us start the overview of a selection of representative examples.

* Corresponding author. Tel.+33 3.83.59.20.77
E-mail address: dmitry.sokolov@univ-lorraine.fr

1877-0509
Peer-review under responsibility of the scientific committee of the 14th International Symposium "Intelligent Systems".

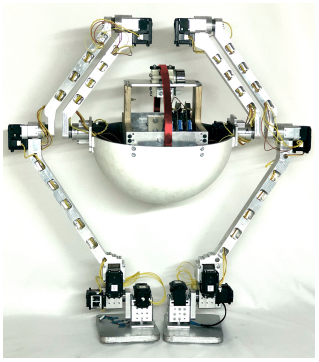

Fig. 1. We are developing a biped that uses uses reaction wheels (shown in red) as an auxiliary stabilization system, see [1]

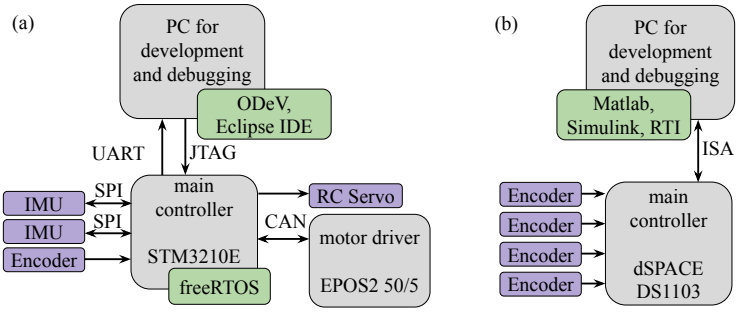

Fig. 2. **(a)** The architecture of Cubli laboratory test bench. **(b)** The architecture of the triple inverted pendulum lab test bench.

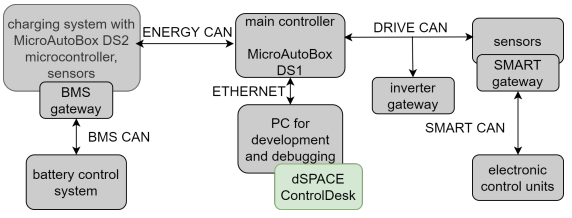

Fig. 3. Hardware and software architecture of the car from [2].

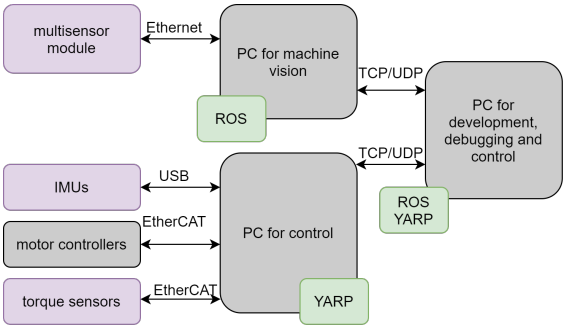

Fig. 4. Architecture of the Walk-man biped [3].

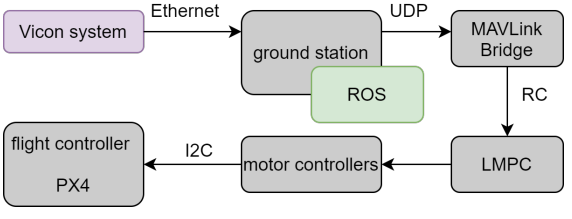

Fig. 5. System architecture of a quadrocopter performing "aggressive maneuvers"

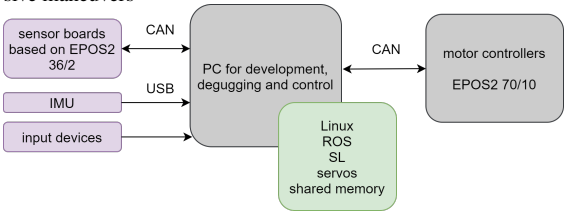

Fig. 6. Hardware and software architecture of a 4-legged robot from the article [4]

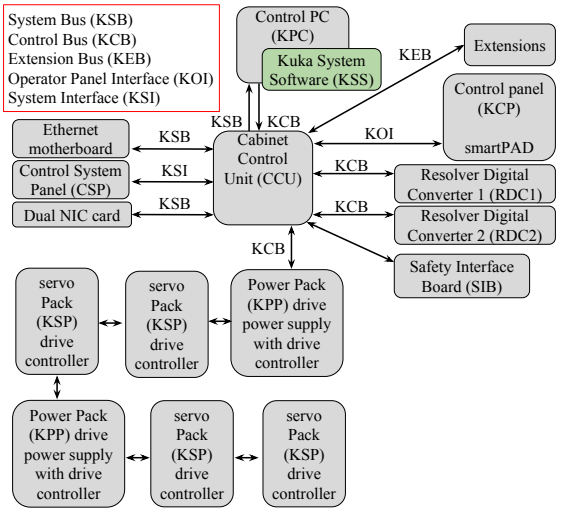

Fig. 7. Hardware and software architecture of KUKA KR C4 extended [2]

2. Architecture desing practices

2.1. Lab test benches

We start this overview with a one-dimensional prototype of a balancing cube bot (Cubly)[5]. The Fig. 2a shows schematically the main elements of the system and their interaction. In this document we provide such schemes for each of the architectures to clearly show their distinctive features. The degree of schemes' details depends on the completeness of the system's description in public sources and also on the expediency of mentioning some elements, especially for complex systems.

The system consists of two controllers connected by a CAN interface, the main controller and the motor controller. IMU sensors, encoder and brake motor servo are connected to the main controller. IMU sensors are connected to the master controller via SPI interface. The main controller is the STM3210E debugging board for STM32F103E microcontrollers with Cortex M3 core. The advantages of this choice are the possibility of rapid prototyping and availability of community support.

FreeRTOS real-time system is used as an operating system. As advantages of FreRTOS the authors note the functionality that allows to perform multitasking and small kernel size - only 4KB. There is a wide choice of real time operating systems: the article[6] contains a rather complete list of them. FreeRTOS is described there as one of the most common free open source systems, along with eCos. The list of officially supported freeRTOS microcontrollers is quite impressive[7]. It is similar to openRTOS (for commercial use) and safeRTOS (for industrial and medical use).

In this case the network of controllers (CAN) is used for communication between the main controller and the motor drivers. CAN is a popular decision both in research, and in the industry. Its advantages are high resistance to interference, simplicity, low cost and ample documentation. However, the disadvantages of this bus are speed (a small amount of data can be transmitted in one package, because a lot of work for service data), limited working distance (30m for 1Mb/s), lack of predictability and limited reliability. There are corresponding higher-level protocols: CANopen, DeviceNet, FTT-CAN and TTCAN[8].

This is a quite typical architecture, let us consider another speciment of the family, namely the triple inverted pendulum on a cart[9]. There is one important point: while the final version of this system is controlled via a dedicated controller (dSPACE DS1103[10] control module), all initial experiments were performed in Matlab/Simulink environment. Some values were pre-calculated, interpolated and stored in look-up tables. Using dSPACE DS1103 is convenient because all I/O configuration can be done in Simulink and then compiled, loaded and started automatically. This can greatly speed up the prototyping of the system as well as reduce the experiment time. The architecture diagram is shown in Fig. 2b.

2.2. Electric cars

Distinctive feature of hardware and software architecture of cars is grouping of electronic systems into electronic control units (ECU). ECU can control one or more electrical systems of the car. Modern cars have a large number of ECU, which requires finding ways to reduce the number of wires connecting these systems. The most common solution is CAN Network[2,11,12,13]. It is a reliable way of data transmission even in case of electromagnetic interference, with a transmission speed of up to 1 Mb/s, which may be sufficient for a certain range of tasks. CAN Nwtwork is used for communication with all devices of the system described in the article[2], except for GPS. Due to the requirements of some devices it was necessary to divide the described system into several subnets: some devices require different network settings, some should be isolated, anb in some cases it is necessary to filter messages. Network gateways (such as ADFweb and DSPACE MicroAutoBox) are used to communicate between these networks.

The dSPACE MicroAutoBox II is used as the main microcontroller that controls all ECU in the vehicle[2]. Car PC (Intel P8700, 1 GB RAM and 500GB HDD) is connected to dSPACE MicroAutoBox II via Ethernet and used for logging, experimenting and development via dSPACE ControlDesk software. In its turn, dSPACE ControlDesk is used in conjunction with Simulink for rapid prototyping. The Fig. 3 shows a simplified scheme of the architecture.

2.3. Humanoid robots

Two-legged walking robots also often use the CAN bus[14,15,16]. Another popular solution is EtherCAT[3,17], which solves the CAN bus problem, namely the inability to provide high poll frequency of low level controllers[17]. As stated

in the article[18], PCI, PCI104 or PCIe are often used to connect motherboards, network cards and analog-to-digital converters. The advantage of tboth CAN and EtherCAT is reliability, and they are both used to connect motors and encoders. However, the CAN bus has a limited bandwidth (1 Mbit/sec) and limited number of devices connected to the bus. For this reason, several buses are often used in parallel in humanoid robots[15,14]. Some robots[19] also use SERCOS III, which is similar to EtherCAT principles and has a similar performance[8].

Let's consider the hardware and software architecture of the Walk-man robot[3]. COM Express computing module based on Pentium i7 quad core processor was used to control the robot's movements. It runs the Ethercat Master device manager. It is responsible for the operation of Ethercat slaves, synchronizing them by exchanging information about their relative state in real time. YARP[20] was chosen as the middle layer software for remote access to the robot motors, meeting the requirements of high speed and low latency.

In the original version, the robot architecture YarpServer and RosCore could communicate directly with each other. But in the real world, their direct communication can result in frequent communication losses. Centralized YARP/ROS servers may not recover from this. Therefore, the controlling PC and the robot were separated and each running its own RosCore and YarpServer, as shown in Fig. 4.

EtherCAT is a rather popular solution, especially for complex systems with large numbers of sensors. For example, there are "robot skin" projects[21], four-legged[22,23] and six-legged[24,25,26] robots, medical rehabilitation robots and exoskeletons[27,28,29].

A typical example of a humanoid robot is Talos Pyréne, developed by PAL-Robotics[17]. It uses the architecture is built around ROS and EtherCAT. The robot is equipped with two 8-core computers, one for the control and the other is responsible for vision and high-level computations. All PAL-Robotics robots, including Talos Pyréne, are deeply integrated with the Robotic Operating System (ROS). They active use ros_control, thus quickly switching between simulations and tests on a real robot. The authors, however, note that although ros_control is easier to use than openRTM, it is currently inferior to openRTM[30].

2.4. Aggressively maneuvering quadrocopters

The architecture of ATVs that perform "aggressive maneuvers" often includes the Vicon motion capture system. The frequency of system operation and the number of sensors used can vary. A typical example is provided in[31], where the Vicon system with the frequency of 200Hz is chosen. Vicon connects via Ethernet to a ground station where ROS is installed. Then the data is sent via UDP bridge over MAVLink (micro air vehicle communication) protocol to Pixhawk px4 autopilot from which I2C commands to engine drivers are sent. The architecture diagram is shown in the Fig. 5. There are also variants using ROS-Matlab Bridge[32].

2.5. Manipulators

Many KUKA industrial manipulators are controlled with Kuka KR C4[33]. The central element of the controller is the Cabinet Control Unit (CCU), the main board that is the interface for all components of the controller. You can see the wiring diagram of the devices in the Fig. 7. The user interface is located on the KUKA Control PC (KPC). The KUKA Servo Pack (KSP) is a motor controller. KUKA Power Pack (KPP) - robot power system with its own controller. Resolver Digital Converter (RDC) is responsible for collecting data on motor position and temperature. The Safety Interface BOARD (SIB) is responsible for the option to safely operate the robot. In addition, an Ethernet card, Controller System Panel (CSP), Dual NIC and SmartPAD are connected to the CCU, which is the operator's console. These elements are connected to the CCU via the following five interfaces: 1) KUKA Extension Bus (KEB), 2) KUKA System Bus (KSB), 3) KUKA Service Interface (KSI), 4) KUKA (KCB), 5) KUKA (KOI). Via the KEB interface, the controller can be connected to industrial Internet networks such as PROFIBUS and DeviceNet. Support for PROFINET, INTERBUS, EtherCAT, Ethernet/IP and VARANBUS is also claimed[34]. For comparison, the Universal Robots E-series collaborative control robots use a control box that supports only 3 industrial Internet standards: ModbusTCP, ProfiNet and EthernetIP[35]. And all these standards are characterized by soft real time, non-deterministic, without guarantees about the transfer time[8].

2.6. Four-legged walking robots

An architecture to control a complex and inherently unstable system such as StarlETH in an unknown environment is described in the article[4]. One of the main requirements for such an architecture is reliability. The basis of the

architecture chosen by the authors is a centralized computer responsible for all high-level management. Schematic representation of the architecture is shown in the Fig. 6. The authors use real time environment for simulation and management SL [36], which simulates the dynamics of a large number of objects, visualizes the robot and the environment, as well as communicates between different services via shared memory. It also includes tools for logging and post-processing.

Motor tracking system is responsible for synchronization and coordination of separate executive motors and consists of low-level regulators (position, speed, torque, etc.) and signal filters. The motor tracking system uses a shared memory to access the simulation and a CAN interface to access the actual robot. This makes it possible to experiment with the robot and run the simulation with exactly the same controller.

The authors note that the factor limiting the speed of the system is the connection between the centralized computer and the motor controllers via CAN interface. Despite parallel channels (one channel per leg), the 1 Mb/s bandwidth limits the total polling frequency to 400 Hz.

3. How to choose an architecture

The considered architectures of real-time systems can be classified by the area — aggressive flight copters, lab test benches such as inverted pendulums, four-legged walking robots and so on. The are popular architecture solutions for each use case, however, there are no clear boundaries between these architectures. Prior to choosing the architecture for a specific task we need to answer a number of questions, and only these answers would allow us to make a good choice. To give an example, the answer to the question whether it is planned to use a large number of different sensors gives a clear tip if you need to think about using ROS. Here we list all the questions that we found relevant:

1. *Are there highly specialized hardware and software solutions for the task you are solving?* A positive answer to this question can narrow down the search area — specialized solutions can use the same technologies as general purpose solutions, but will take into account special requirements. For example, most specialized flight controllers are lightweight and small in size, while specialized controllers for industrial manipulators support a large set of standards such as industrial internet for more flexible integration into existing production lines. Many ready-made software solutions for robotic tasks become available using ROS or Matlab.
2. *Do you plan to connect a large number of different sensors?* When you connect a large number of sensors, the network load grows. In this case you can use EtherCAT or SERCOS III. However, if you connect a large number of sensors from different manufacturers, you should pay attention to ROS, which already has interfaces for many different sensors in its open database.
3. *How complex is the system: how many drives and how many non-motorized degrees of freedom?* If the system is complex, you can divide it into separate logical units. This will help to avoid some problems and increase reliability of the system. For example, centralized ROS and YARP servers may not recover after losing communication with each other [3]. To increase speed, it may be worth parallelize channels into separate groups of motors [4]. For example, it is possible to partially compensate for the lack of CAN transmission speed. But you should be careful, as even in spite of paralleling CAN, the bus may become a bottleneck of the system and limit the speed in general.
4. *Do you need quick prototyping possibilites?* If you need to experiment a lot, you should consider using Matlab/Simulink and systems that support it, such as dSPACE DS1104, MicroAutoBox and external PCs for development and debugging. If you plan to test different algorithms for the same hardware architecture, using Matlab/Simulink can significantly speed up their testing cycle. For example, there is HDL coder — a Matlab package that allows you to translate Matlab/Simulink code and models into C++ code and run it on microcontrollers from different manufacturers. It also allows SIL, PIL and HIL testing to be performed quickly.
5. *How critical is the reliability of the system?* If one of the main requirements of the architecture is reliability (e.g. the brake system of a car or a complex dynamic system), you should pay attention to the CAN Network, which has established itself as a network with high reliability [8]. In general, of course, the reliability of the system depends on many factors — all nodes of the system and the connection between them must be stable. As a rule of thumb, industrial solutions offered by large companies are more reliable than free and open source solutions. A software solution can be to program the behavior of the device in the event of a system failure. It is also worth

making sure that the system can recover from a failure - as, for example, in[3] the authors have divided RosCore and YarpServer into different devices so that after the loss of communication these servers could recover.

6. *What computing power is required for the system operation?* If a large computing power is required, and for some reason it cannot be placed inside the device under development, it can be transferred to an external calculation module. Also, calculations can be divided logically between different nodes of system (for example, preprocessing of the data received from sensors on the controller near to sensors). If large computing power is only needed in the experiment phase, it is reasonable to use a separate external computer for the experiments.
7. *What are the maximum bandwidth requirements?* It is necessary to calculate the worst case of the channel load (for example, in the project[2] the author performs such calculations) and to make sure that the chose network type will allow to transmit this amount of data at the speed suitable for you. If not, you should either choose another network (the bandwidth of different networks are given in[8]) or parallelize the data transmission to separate nodes as done in[4]. It is also important to pay attention not only to the bandwidth, but also to the delivery time. It may be crucial to use deterministic systems guaranteeing the time of delivery such as EtherCAT, EPL, Profinet IRT and SERCOS III.
8. *What is the planned carrying capacity of the system?* The question of limited payload is crucial for self-contained/autonomous devices, whereas in other cases some system elements can be externalized. If the system's payload is small and the computing capacity is large, you can take some of the computations to an external computer. Then the controller on the device itself may be smaller, with less power consumption.
9. *Do the scenarios consider the interaction of the device with another device, do you need access to some shared information?* In this case one should think about using cloud services or consider a choice of data transmission channel between the devices which will not become a bottleneck of the real-time system being developed.
10. *What is the maximum planned speed of movement of the device?* Quite often an action or task must be performed under a hard deadline, otherwise the result will have no meaning[8]. One of the factors that determine this deadline is the velocity of the (components of the) device. The system must have time to react to the changes in the environment. Otherwise, for example, an ATV flying at a high speed will not have time to react to an obstacle on its way and may crash.
11. *Will device operate in a known or unknown environment?* In an unknown environment, much more sensor data needs to be processed. This should be taken into account in the calculations as a time spent on data processing. Correspondingly, the network load will also increase.
12. *Are the hardware dimensions critical?* If so (e.g., if it is a satellite to be placed into orbit), then smaller microcontrollers can be used. For example, the use of System on a Chip allows us to greatly reduce the system dimensions.
13. *Is this device susceptible to interference?*. CAN is a popular solution if noise immunity is critical. However, other industrial standards (EtherCAT, SERCOS III) also have good immunity.
14. *What is the distance between the interacting modules?*. This must be taken into account, as some networks have serious limitations in this respect. For example, CAN, while having undeniable reliability advantages over short networks, has considerable speed losses for longer (typ. 30 meters) networks.

4. Choosing the architecture for a non-anthropomorphic biped

Here we give an example of architecture choice for the non-anthropomorphic robot we are currently developing in the Laboratory of Robotics and Mechatronics of the Kuban State University (refer to Fig. 1). The biped has an auxiliary dynamic stabilization system which consists of two reaction wheels inside the robot's body and we are currently studying the improvement of the stabilization system on the basis of a scissored pair control moment gyroscope[37].

Let us see how answering the above questions narrows down the search area for the architecture choice.

1. There are there highly specialized solutions, mainly software. It is possible to use open drivers for sensors and ready algorithms for, say, motion planning, machine vision, SLAM and so on. We can choose ROS, as it supports a large number of different sensors, it uses code from many open sources, can automatically update it, apply patches and so on[38].

2. We do plan to use many sensors, but we are not talking about hundreds and thousands, so using EtherCAT or SERCOS III is not critical. However, completely different sensors — lidars, cameras, pressure sensors, IMUs and others — will be used. For this case, again, a popular solution is to use ROS with its Peer-to-Peer topology and support for a large number of sensors.
3. Our system is rather complex: it has 2 legs with 4 servomotors on each[39]. Inside the robot body there is a stabilization system with 3 motors. Part of the sensors are responsible for the stabilization of the robot — Hall sensors in the motors of the stabilization system, pressure sensors on the feet of the robot, gyroscope. Leg servo motors can also be included in the group of robot devices that affect stabilization. Sensors responsible for machine vision, SLAM can be separated into a different system. These two groups can be separated and connected to different computing modules, as done in[3]. Also, it makes sense to create parallel networks to servomotors to increase the available bandwidth.
4. The possibility of rapid prototyping is very important, as a number of studies are planned during the development process. This will require flexibility, the ability to quickly remake individual system blocks and to test in a new configuration. One of the popular means of rapid prototyping is Matlab.
5. The reliability is of course important but not crucial: while it is planned to commercialize the final product as an educational kit, it would not be very dangerous for the user.
6. Raspberry Pi 3B (ARM Cortex-A53 x64, 1.2 GHz) should suffice for the stabilization task; as for the total computing power for the final robot it is hard to tell at this stage, probably we well need something comparable to Walk man[3].
7. CAN bus with paralallel channels should suffice for the controller-legs communication, Ethernet interface suffices for the high-level control tasks.
8. The carrying capacity of the robot is only 3.8 kg, therefore, the control module need to be externalized.
9. No interaction with another robot/device is planned at this stage.
10. The speed limit is 5 km/h.
11. The robot will move in a known environment at first, but unknown environment is planned. We need to plan for an external acquisition system.
12. The hardware dimensions are very important, although it is less critical than for satellites.
13. The robot may operate in a very noisy environment, so fault tolerant interfaces are a must.
14. The maximum distance between system's modules is 5 meters, thus CAN bus is a reasonable choice.

While main hardware choices for the robot itself were made in [39] (carrying capacity, movement speed etc), the above answers orient our choice for control architecture to something very similar to Walk-man depicted in Fig. 4, the main difference is that we do not need EtherCAT, as CAN bus suffices for our needs.

5. Conclusion

The examples considered in this article show some of the most popular real-time hardware and software architectures. Many of the listed solutions similar characteristics (SERCOS III and EtherCAT), some can be (in some cases) mutually interchangeable, and many can be used in different combinations with each other. We believe that by answering the questions in section 3, you can define the basic requirements for the hardware and software architecture, choose one of the described solutions, or use these examples as a starting point for further searches.

References

1. Efimov, D., Aranovskiy, S., Fridman, E., Sokolov, D., Wang, J., Bobtsov, A.. Adaptive stabilization by delay with biased measurements. In: *IFAC 2020 - 21st IFAC World Congress*. Berlin, Germany; 2020, URL: `https://hal.inria.fr/hal-02634582`.
2. Miranda Bermejo, J.. *Design and Implementation of a Control System for Testing an Experimental Electrical Vehicle*. Ph.D. thesis; UPC, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona; 2010. URL: `http://hdl.handle.net/2099.1/14638`.
3. Tsagarakis, N.G., Caldwell, D.G., Negrello, F., Choi, W., Baccelliere, L., Loc, V., et al. Walk-man: A high-performance humanoid platform for realistic environments. *Journal of Field Robotics* 2017;**34**(7):1225–1259.
4. Hutter, M., Gehring, C., Bloesch, M., Hoepflinger, M.A., Remy, C.D., Siegwart, R.. Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion. In: *Adaptive Mobile Robotics*. World Scientific; 2012, p. 483–490.

5. Gajamohan, M., Merz, M., Thommen, I., D'Andrea, R.. The cubli: A cube that can jump up and balance. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE; 2012, p. 3722–3727.
6. Zhu, M.Y.. The next hundreds embedded real-time operating systems. 2012. doi:10.13140/RG.2.2.28990.69445.
7. Amazon Web Services, . freeRTOS supported MCUs. 2018. URL: https://freertos.org/RTOS_ports.html.
8. Wilamowski, B.M., Irwin, J.D.. *Industrial communication systems*. CRC Press; 2016.
9. Glück, T., Eder, A., Kugi, A.. Swing-up control of a triple pendulum on a cart with experimental validation. *Automatica* 2013;**49**(3):801–808.
10. dSPACE GmbH, . dspace ds1103 documentation. 2018. URL: http://www.ceanet.com.au/Portals/0/documents/products/dSPACE/dspace_2008_ds1103_en_pi777.pdf.
11. Nolte, T., Hansson, H., Bello, L.L.. Automotive communications-past, current and future. In: *2005 IEEE Conference on Emerging Technologies and Factory Automation*; vol. 1. IEEE; 2005, p. 8–pp.
12. Fredriksson, L.B.. Can for critical embedded automotive networks. *Ieee Micro* 2002;**22**(4):28–35.
13. Leen, G., Heffernan, D.. Expanding automotive electronic systems. *Computer* 2002;**35**(1):88–93.
14. Kaneko, K., Harada, K., Kanehiro, F., Miyamori, G., Akachi, K.. Humanoid robot hrp-3. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE; 2008, p. 2471–2478.
15. Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F.. The icub humanoid robot: an open platform for research in embodied cognition. In: *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM; 2008, p. 50–56.
16. Lim, J., Lee, I., Shim, I., Jung, H., Joe, H.M., Bae, H., et al. Robot system of drc-hubo+ and control strategy of team kaist in darpa robotics challenge finals. *Journal of Field Robotics* 2017;**34**(4):802–829.
17. Stasse, O., Flayols, T., Budhiraja, R., Giraud-Esclasse, K., Carpentier, J., Mirabel, J., et al. Talos: A new humanoid research platform targeted for industrial applications. In: *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*. IEEE; 2017, p. 689–695.
18. Stasse, O., Flayols, T.. An overview of humanoid robots technologies. In: *Biomechanics of Anthropomorphic Systems*. Springer; 2019, p. 281–310.
19. Lohmeier, S.. *Design and Realization of a Humanoid Robot for Fast and Autonomous Bipedal Locomotion:*. Robotik und Automation. Verlag Dr. Hut; 2010. ISBN 9783868537345. URL: https://books.google.fr/books?id=idAuygAACAAJ.
20. Metta, G., Fitzpatrick, P., Natale, L.. Yarp: yet another robot platform. *International Journal of Advanced Robotic Systems* 2006;**3**(1):8.
21. Baglini, E., Cannata, G., Mastrogiovanni, F.. Design of an embedded networking infrastructure for whole-body tactile sensing in humanoid robots. In: *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE; 2010, p. 671–676.
22. Semini, C., Goldsmith, J., Rehman, B.U., Frigerio, M., Barasuol, V., Focchie, M., et al. Design overview of the hydraulic quadruped robots. In: *The Fourteenth Scandinavian International Conference on Fluid Power*. 2015, .
23. Guo, K., Li, S., Huang, D.. Real-time quadruped robot control system based on xenomai. In: *Chinese Automation Congress (CAC), 2015*. IEEE; 2015, p. 342–347.
24. Gui, B., Wang, H., Chen, W.. Stability analysis for a hexapod robot walking on slopes. In: *Robotics and Biomimetics (ROBIO), 2015 IEEE International Conference on*. IEEE; 2015, p. 1888–1893.
25. Deng, H., Xin, G., Zhong, G., Mistry, M.. Object carrying of hexapod robots with integrated mechanism of leg and arm. *Robotics and Computer-Integrated Manufacturing* 2018;**54**:145–155.
26. Zhao, Y., Chai, X., Gao, F., Qi, C.. Obstacle avoidance and motion planning scheme for a hexapod robot octopus-iii. *Robotics and Autonomous Systems* 2018;**103**:199–212.
27. Allouche, B., Dequidt, A., Vermeiren, L., Hamon, P.. Design and control of a sit-to-stand assistive device via ethercat fieldbus. In: *Industrial Technology (ICIT), 2017 IEEE International Conference on*. IEEE; 2017, p. 761–766.
28. Grosu, V., Guerrero, C.R., Grosu, S., Leu, A., Ristic-Durrant, D., Vanderborght, B., et al. Real-time physical layer architecture for corbys gait rehabilitation robot. In: *Rehabilitation Robotics (ICORR), 2015 IEEE International Conference on*. IEEE; 2015, p. 606–611.
29. Rebelo, J., Sednaoui, T., den Exter, E.B., Krueger, T., Schiele, A.. Bilateral robot teleoperation: A wearable arm exoskeleton featuring an intuitive user interface. *IEEE Robotics & Automation Magazine* 2014;**21**(4):62–69.
30. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.K.. Rt-middleware: distributed component middleware for rt (robot technology). In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE; 2005, p. 3933–3938.
31. Bangura, M., Mahony, R.. Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes* 2014;**47**(3):11773 – 11780. URL: http://www.sciencedirect.com/science/article/pii/S1474667016434890. doi:https://doi.org/10.3182/20140824-6-ZA-1003.00203; 19th IFAC World Congress.
32. Turpin, M., Michael, N., Kumar, V.. Trajectory design and control for aggressive formation flight with quadrotors. *Autonomous Robots* 2012;**33**(1-2):143–156.
33. KUKA, . KUKA kr c4 extended documentation. 2018. URL: http://supportwop.com/IntegrationRobot/content/3-Armoires_de_commande/KRC4_extended/Norme_UL/English_manual_KR_C4_extended_NA_en.pdf.
34. KUKA, . KUKA kr c4 booklet. 2018. URL: https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_pb_controllers_en.pdf.
35. Universal Robots, . e-series documentation. 2018. URL: https://www.scottautomation.com/assets/Uploads/e-Series-Universal-Robots.pdf.
36. Schaal, S.. The sl simulation and real-time control software package. *University of Southern California* 2009;.
37. Aranovskiy, S., Ryadchikov, I., Nikulchev, E., Wang, J., Sokolov, D.. Experimental comparison of velocity observers: A scissored pair control moment gyroscope case study. *IEEE Access* 2020;**8**:21694–21702.
38. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., et al. Ros: an open-source robot operating system. In: *ICRA workshop on open source software*; vol. 3. Kobe, Japan; 2009, p. 5.
39. Ryadchikov, I., Sechenev, S., Sinitsa, S., Svidlov, A., Volkodav, P., Feshin, A., et al. Design and control of self-stabilizing angular robotics anywalker. *International Journal of Advanced Computer Science and Applications* 2017;**8**(9):29.