Challenges in the software architecture design for autonomous legged robots

Marco Frigerio, Claudio Semini and Darwin G. Caldwell
Dept. of Advanced Robotics,
Istituto Italiano di Tecnologia (IIT),
via Morego, 30, 16163 Genova
<first name>.<last name>@iit.it

Jonas Buchli
Agile & Dexterous Robotics Lab,
ETH Zurich,
Tannenstr. 3, 8092 Zürich
buchlij@ethz.ch

***Abstract*— HyQ is a fully torque controlled quadruped robot for research on legged locomotion. This extended abstract gives an overview of the current software system mounted on the robot, which allows us to perform successful experiments such as fast trotting and squat jumping. However, the complexity of autonomous legged locomotion, and the consequent need to incorporate additional functionality such as vision, demand for a more general software architecture. Some of the challenges and the topics to be addressed in the architectural design will be briefly discussed.**

I. Introduction

At the Advanced Robotics Department of the Istituto Italiano di Tecnologia, our research group is developing HyQ, a high performance quadruped robot with hydraulic actuation [8, 3, 1, 5]. The software system we developed for the robot allows us to perform experiments in various fields such as legged locomotion in complex terrains, whole body control (see Figure 2).

Our system is the realization of a specific architecture that has proved to be successful, and can be applied also for other articulated robots (Section II). However, the final goals of making the robot autonomous and versatile enough to perform tasks such as fast running and careful navigation on rough terrain, demand for a more sophisticated software architecture. Building on our experience, Section III briefly illustrates some of the limitations of the existing system and proposes some ideas for the future work on this topic.

Here and in the following, with *software architecture* we mean the design about the overall organization of the components of a software system, which highlights properties like the computational complexity, the frequency of execution, the entity of information flows among components [7, 2].

II. The current control system

The diagram in Figure 1 shows the gross organization of the hardware and the software of HyQ, which is also a quite general layout that can be applied to different articulated robots. We implemented such an architecture on our robot and achieved promising results [1, 3].

At the base of the diagram we find the hardware and the operating system, which obviously host all the rest of the software modules. A general purpose computer is equipped with input–output (I/O) boards that allow the communication with sensors and actuators. The most important requirement

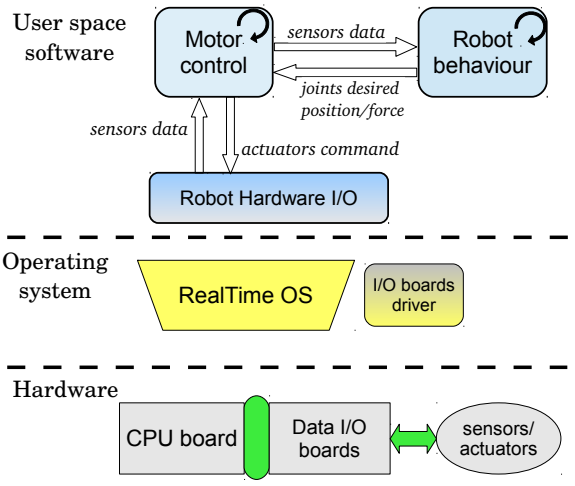

Fig. 1. Logical view of the current computing setup of the HyQ robot. Data acquisition boards enable the interaction with the robot hardware (e.g. sensors). The real–time capable operating system includes a driver giving access to the facilities of the I/O board. The user level code includes a library to abstract the hardware and two other active components: the motor control module and the robot behaviour module, both running as processes. The sensor data as well as the desired trajectories include the position and the force at the joints.

for the operating system is to be *hard real–time capable*, i.e. support processes that need to be triggered with reliable timing, without being preempted.

The robot-hardware I/O layer (or "hardware abstraction layer" – HAL) deals with the sensor measurements being transmitted to the computer (input), and the commands destined to the drivers of the actuators (output). This component is a software library that interacts with the I/O boards that provide the physical connections to the devices. It abstracts the low level details of the communication and of the specific sensors, and provides an interface to the numerical control algorithms. We implemented this component enforcing modularity, to bound the required software modifications in case of some change in the actual hardware.

The two modules at the top of Figure 1 control the robot according to the user plans. The *motor control* is primarily responsible for the processing of the sensory data and of the input commands, in order to produce the appropriate output commands for the actuators, in a closed-loop fashion (e.g. a

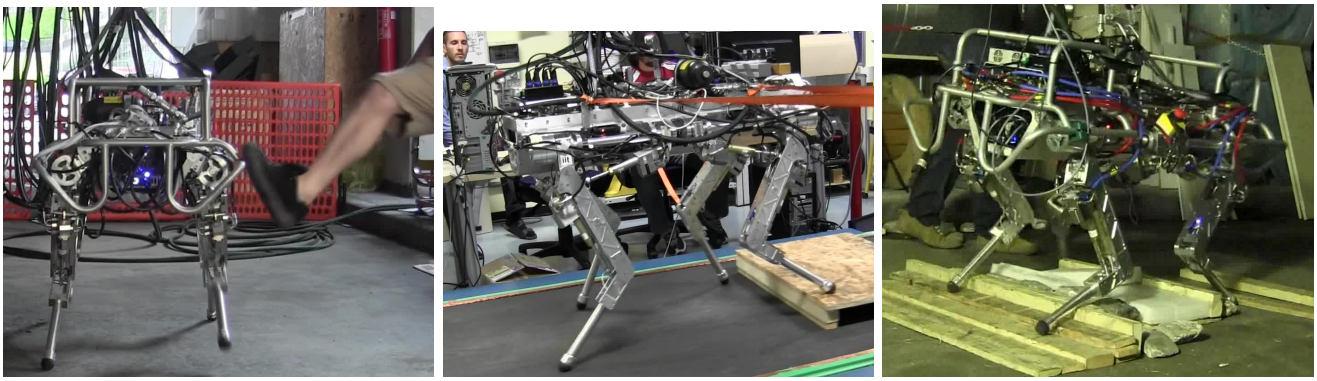

Fig. 2. The HyQ robot performing some interesting motions. From left to right: reacting to lateral disturbances, trotting over obstacles on a treadmill, trotting outside on rough terrain. All these experiments could be performed also thanks to the software system represented in Figure 1.

PID controller on the joint torque).

The second module – *robot behaviour*, in the figure (we will also refer to it as the *task* module) – generates commands for the motor control module, in the form of desired positions or forces at the joints. It is basically a trajectory generator, even though this term is a bit generic and may hide quite a broad set of sub-modules interacting according to complex patterns. The users implements their own behaviours within this module.

In general, it is highly desirable for the motor control to be as much as possible agnostic with respect to the task module, and limit the interaction between the two modules according to a simple and specific interface. A clear identification of such interface, i.e. of the information flow, is part of the definition of the architecture.

The SL *simulator and motor controller package* basically provides an implementation of the motor control and the robot behaviour blocks, so we adopted it for our system [9]. SL can be configured to have the motor control interact with a simulator (also included in the package) instead of a real robot. The task module is unaware of the current mode of execution, hence it is possible to try behaviours in simulation and then move to the real robot, without changing anything in the implementation of the behaviour itself. This very effective feature is possible thanks to the separation between the motor-control and the trajectory-generation, two distinct activities. This is a general principle for designing a software architecture for an articulated robot, independently from the use of SL.

III. The need for a more general software architecture

Achieving useful behaviours and autonomy in articulated robots is a challenging goal. Legged locomotion on rough terrain is a notable example – and it is also the main topic of research in the HyQ project.

For such applications, the system described in the previous section has limitations, some of which are listed here:

- A single-process access to hardware resources (as the motor control process using the I/O library) does not handle well information flows that vary in the amount of data and the frequency. There might be no point, for example, to sample an Inertial Measurement Unit at the same high frequency required for position sensors. With cameras, sampling at e.g. 500Hz is simply unfeasible.
- The robot behaviour module can be arbitrarily complicated. A single component incorporating the whole logic to generate joint trajectories is not sufficient. This part of the system depends significantly on the specific application.
- A single computer for the deployment of the whole system. Weakly coupled activities or with very different requirements (e.g. hard real–time position control versus image processing) might very well be more effective (and safe) if executed on different machines. Constraints like the available space on the robot or the costs of development on different hardware may bias this analysis; however, the compromises required to fit the hardware constraints should ideally be made with the purely problem-driven architecture available as a reference.[1]

In general, it is necessary to devise a proper software architecture identifying and quantifying the information flows and the logical activities the system has to perform. A logical architecture must be designed with no reference to any implementation technology, which is a subsequent step of the process [2]. It sometimes happens in the robotics community, that the availability of libraries or middlewares that might have gained a certain fame – sometimes because they are actually effective – leads to technology driven–systems, which however is a conceptually flawed approach.

One of the interesting challenges in the development of the architecture would be to identify the role of controllers within the whole system. For example, research has shown that force control (plus model based control) enables compliant interaction with the environment, which leads to benefits at the level of the final application, for instance to cope with unperceived obstacles during locomotion [4, 3]. Which are the interactions required between low level controllers

[1] In this case we are referring primarily to the *deployment architecture* [2].

and higher level components such as planners, to achieve autonomy and robustness? For instance, should the controller notify the presence of an obstacle (e.g. detected by unexpected position of the foot) even though it can be handled by the compliance of the leg? Is it sufficient for the controller to expose an interface limited to the setting of the gains?

The following points refer more specifically to the application of autonomous locomotion in unstructured environments, and they also represent issues to be addressed to devise a coherent architecture.

- Terrain and world modelling: sensors such as cameras and laser range scanners are useless if their data is not used to construct some sort of description of the terrain and the environment the robot has to move in.
- Robot task model: a description of the possible high level commands for the robot should be identified, e.g. how to ask the robot to reach a certain position, with a certain gait and/or a certain velocity, with time constraints, etc.
- Robot model: the control logic could exploit information like the current gait and speed of the robot, the power availability, the estimated quality of the various sensor measurements.

Other interesting challenges in the design of the software system refer to the representation of time and of the own computing capabilities of the robot (a sort of *Quality of Service* measurement [10]); imagine a running robot that perceives a moving and approaching obstacle (e.g. a car): avoiding crashing into the obstacle involves reasoning about its speed and the speed of the robot – which pertains the *real* physical time – possibly taking into account the available computing resources, that is how long the robot can afford to "think" before taking a decision.

As far as the actual implementation is concerned, it is important to adopt sound technologies. For example, the code for low level controllers must be hard real–time capable; kinematics and dynamics computations are fundamental to achieve complex behaviours, but it is not straightforward to implement them, especially when efficiency is important. Our work described for instance in [6] addresses these issues, and provides sound tools for the implementation of important components of the whole architecture.

IV. Conclusions

The software system currently controlling the HyQ robot enabled to start a state-of-the-art research in the field of torque controlled, high performance locomotion. Different behaviours in the form of generators for the desired joint positions and forces can be developed and tried on the robot.

However, due to the complexity of the domain such a system is not the definite solution to actually enable versatility and autonomy for the HyQ robot or other machines of similar complexity; some limitations and a direction for future development about the software architecture design have been highlighted.

The points discussed in this short paper will be the subject of future research.

Acknowledgements

This research has been funded by the Department of Advanced Robotics, Istituto Italiano di Tecnologia, via Morego, 30, 16163 Genova. The authors would like to thank Alessio Margan for his valuable help in the development of the low level software.

References

[1] Victor Barasuol, Jonas Buchli, Claudio Semini, Marco Frigerio, Edson R. De Pieri, and Darwin G. Caldwell. "A Reactive Controller Framework for Quadrupedal Locomotion on Challenging Terrain". In: *IEEE International Conference on Robotics and Automation (ICRA)*. [accepted for publication]. 2013.

[2] Diego Bernini and Francesco Tisato. "Explaining architectural choices to non-architects". In: *4th European conference on Software architecture*. ECSA'10. Copenhagen, Denmark: Springer-Verlag, 2010, pp. 352–359.

[3] Thiago Boaventura, Claudio Semini, Jonas Buchli, Marco Frigerio, Michele Focchi, and Darwin G. Caldwell. "Dynamic Torque Control of a Hydraulic Quadruped Robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2012.

[4] Jonas Buchli et al. "Compliant quadruped locomotion over rough terrain". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2009, pp. 814–820.

[5] Michele Focchi, Thiago Boaventura, Claudio Semini, Marco Frigerio, Jonas Buchli, and Darwin G. Caldwell. "Torque-control Based Compliant Actuation of a Quadruped Robot". In: *12th IEEE International Workshop on Advanced Motion Control (AMC)*. 2012.

[6] Marco Frigerio, Jonas Buchli, and Darwin G. Caldwell. "Code Generation of Algebraic Quantities for Robot Controllers". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012.

[7] David Garlan and Mary Shaw. "An introduction to software architecture". In: *Advances in Software Engineering and Knowledge Engineering*. Ed. by V. Ambriola and G. Tortora. Vol. 1. World Scientific Publishing Company, 1993, pp. 1–39.

[8] Claudio Semini et al. "Design of HyQ – a Hydraulically and Electrically Actuated Quadruped Robot". In: *IMechE Part I: J. of Systems and Control Engineering* 225 (2011), pp. 831–849.

[9] Stefan Schaal. *The SL simulation and real-time control software package*. Tech. rep. CLMC lab, University of Southern California, 2009.

[10] Andreas Steck and Christian Schlegel. "Towards Quality of Service and Resource Aware Robotic Systems through Model-Driven Software Development". In: *1st International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob)*. Sept. 2010.