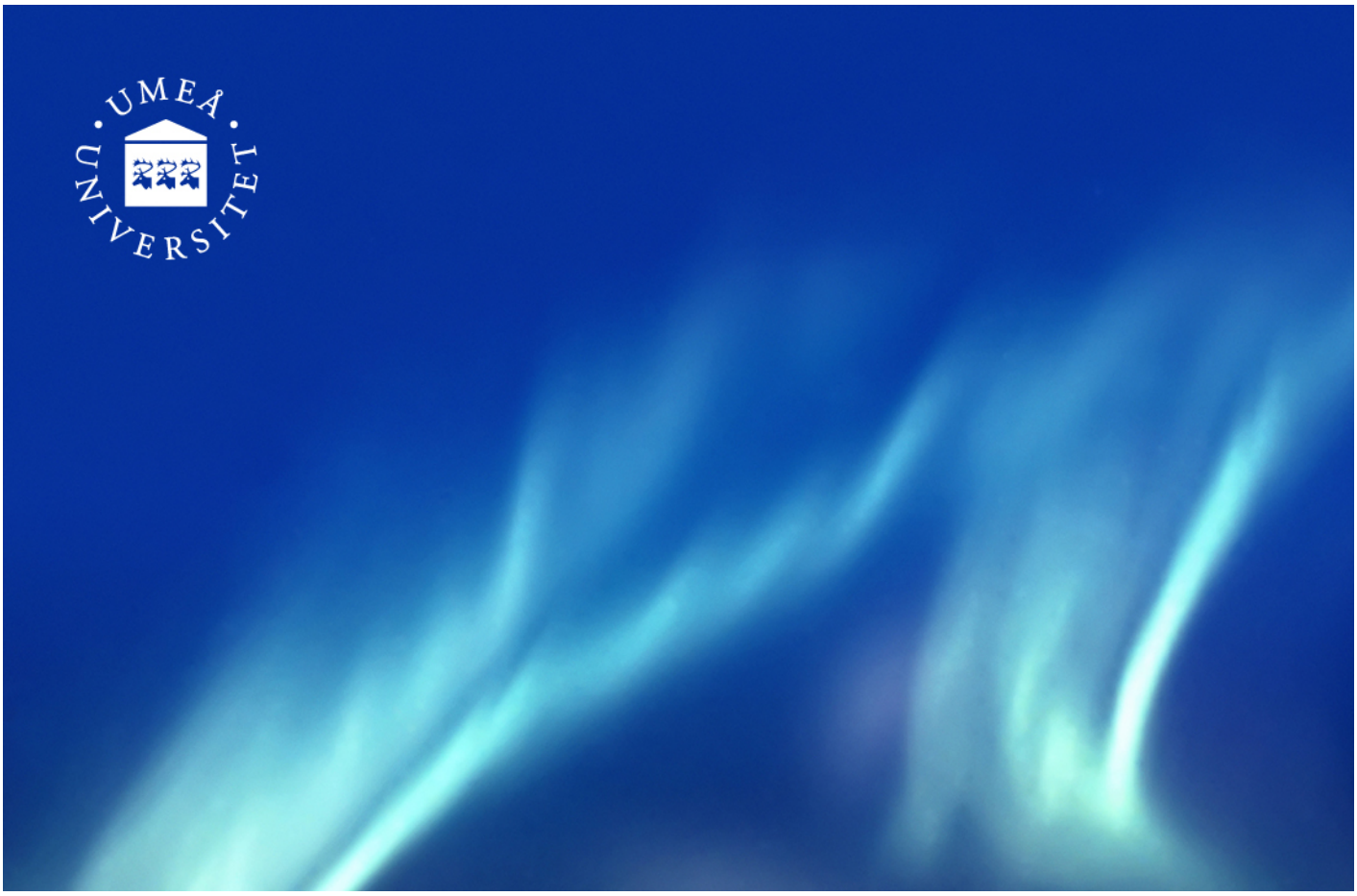

First RHEA International Conference on Robotics and associated High-technologies and Equipment for Agriculture
Hosted by the University of Pisa
Pisa, Italy, September 19-20-21, 2012

A software framework for agricultural and forestry robotics

Thomas Hellström, Ola Ringdahl

Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden

e-mail: thomash@cs.umu.se ringdahl@cs.umu.se

Keywords robot architecture, architecture view, middleware, ROS

Abstract:

In this paper we describe on-going development of a generic software framework for development of agricultural and forestry robots. The goal is to provide generic high-level functionality and to encourage distributed and structured programming, thus leading to faster and simplified development of robots. Different aspects of the framework are described using different architecture views. We show how these views complement each other in a way that supports development and description of robot software.

1. Introduction

Construction of software has become an increasingly complex and time-consuming part of robot development. In this paper, we present on-going development of a software framework for development of agricultural and forestry robots within the CROPS project (http://www.crops-robots.eu/). The goal is to provide generic high-level functionality and to encourage distributed and structured programming, thus leading to faster and simplified development. The framework will be used for the development of several robots with slightly different tasks, which means that it has to be general and possible to configure in several respects. The main contribution in the paper is the hybrid architecture of this framework. Different aspects of the framework are described using different architecture views. We

show how these views complement each other in a way that supports development and description of the system.

1.1 Background

In the field of mobile robotics, several *robot architectures* have been proposed over the years. They all describe different approaches how to organize sensing, planning, motion, cognition, and control. Most architectures follow one of the classical paradigms Deliberative, Reactive, or Hybrid (Matarić 2002). In software engineering, the concept of architecture has been thoroughly investigated and developed. According to the definition in ISO/IEC/IEEE 42010 (International Organization for Standardization 2007), an architecture is the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Depending on which elements and relationships we consider, different types of *architecture views* are described. The following architecture views are often used to describe software-intensive systems (Kruchten, 1995): Logical view, Development view, Process view and Physical view. Each one of these architecture views describes important and complementary aspects of a system.

2. Materials and methods

In this section, the architecture of the framework developed for robots within the CROPS project will be described with reference to the different views described in the previous section.

2.1 Logical view

Defines top-level functionality, general modules and their relationship. In our case, this is mostly described in the specification documents for the CROPS robots. According to these, the robots should be able to determine locations of ripe and healthy fruit in the vicinity of the robot, determine appropriate picking order, plan a route to individual fruits, move to a fruit, determine how to grip a fruit, grip a fruit, determine how to cut a fruit, cut a fruit, and finally bring a fruit to a basket. All this should be done is a safe manner both for humans

working close to the robot, and for fruits and plants. The same software framework should work for development of different robots for harvesting of apples, sweet pepper, and grapes. Based on this specification, the following generic functional modules are identified: *Main control program*: runs the main loop that detects fruits, plans and executes motion, gripping, and harvesting. *Virtual sensors*: abstractions of sensors that do not only measure the physical world, but also process results from one or several physical, or virtual, sensors. *Planner*: generates plans for picking order, grasp patterns, and possibly also motion planning. *Arm and gripper control*: provides an interface to the robot arm (Baur et al. 2012), gripper, and cutter. *Error manager*: detects and handles situations when things go wrong. *Resource manager*: lets the user tune and configure the system for a task (choice of sensors, algorithms, parameters etc.). *Graphical user interface*: allows a user to start, pause, stop, and inspect the robot. *Fusion/Learning*: creates and adapts virtual sensors. *Performance monitor*: checks the health of all modules and communication channels (for instance physical connections and data flows).

2.2 Development and process view

The development view describes the organization of software modules and how they communicate to fulfil non-functional requirements such as performance and availability. This description specifies programming and software development paradigms such as object orientation, component based approaches, and model driven design. The process view describes issues related to concurrency, distribution versus centralization, and communication modes (such as point-to-point or publish-subscribe). In the CROPS project, most development is done using the ROS (Robot Operating System) environment (Quigley et al. 2009), with programming done in C++, which encourages object-oriented systems. ROS manages parallel execution of software modules (denoted nodes) and administrates Ethernet based communication between nodes. ROS also supports transparent physical relocation of nodes and contains other useful functionality. In our case, most aspects of

the development and process views are jointly described by a directed graph, with vertices representing ROS nodes, and links representing information flow (ROS message passing).

2.3 Physical view

This view describes placement of physical components and physical connections, and takes into account non-functional system requirements such as availability, reliability, performance, and scalability (Kruchten, 1995). In our case, this view is illustrated as a block diagram showing how laptops, sensors and actuators communicate through a standard Ethernet bus with possibility to connect also CAN based equipment. The hardware is used to implement the functionality described by the Logical view, and also the application specific functionality according to the specification documents. Thanks to the ROS environment, the physical location of software modules within the ROS domain is very flexible. This means that processor-intensive computations, if necessary, can be moved to separate laptops without any changes in the programs.

2.4 Robot architecture

We further develop the architecture by specifying and designing how the different modules function and interact. This specification is affected by the design choices described in the Logical, Development, Process and Physical views. The result is a robot architecture that follows the hybrid paradigm.

The planner component commonly found in hybrid architectures is replaced by a static state machine. This is viewed as a good solution for the CROPS robots, and for other robots when there is no need for planning in the sense of problem solving or finding novel solutions to new tasks. This is the case if the given task for the robot is well defined, and the behaviour of the robot can be described in, for instance, a flowchart.

The proposed architecture also differs from the standard hybrid model in the way behaviours are defined and interact with the state machine/planner. Each state is typically associated with a behaviour. When the state machine moves to a new state, the corresponding behaviour is activated. Behaviours are

implemented as ROS nodes and execute independently of the state machine, such that the latter at each time step can decide to change state or stop execution. This is particularly important when implementing user interfaces and also error handling.

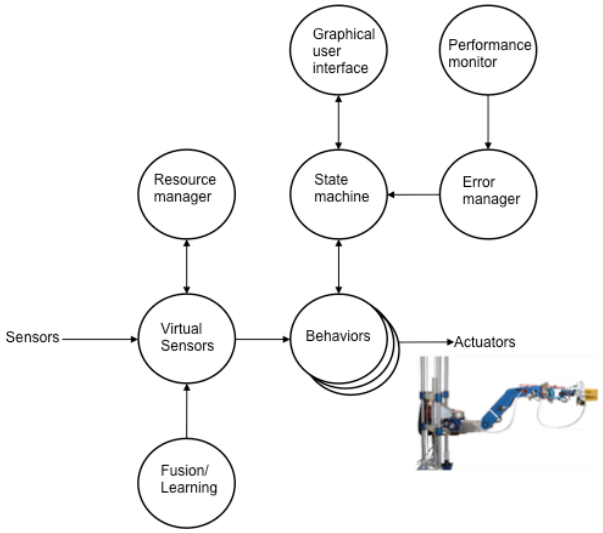

Figure 1. Hybrid architecture for the developed framework.

The Graphical user interface contains controls to start, pause, and stop the state machine. It also displays informative messages retrieved from the behaviours and other modules. A Virtual sensor is an abstraction of a regular sensor, and connects to one or several physical or virtual sensors. The Resource manager provides customization of the virtual sensors such that the system can be adapted to varying environmental conditions. At present, this kind of customization is done off-line and stored in configuration files that are read at system initialization.

Errors are dealt with at two levels in the system. Some errors are both detected and dealt with locally where the problem appears. One example is if a node responsible for fruit localization is not able to find any fruit in the image. The node may deal with this by acquiring a new picture and trying again, moving the camera or platform, or calling for human assistance. Other errors are independent of the current state, and are preferable detected and

dealt with at system level, by the Performance monitor and the Error manager. One example is a camera that suddenly stops functioning. This may be detected by the Performance monitor and forwarded to Error manager, which then stops the state machine and prints an error message through the GUI.

3. Results and conclusions

We have presented results from on-going development of a software framework for agricultural robots. The architecture is described from several views; Logical, Development, Process, and Physical. This way of describing a system displays complementary information that is not easily given in a single architecture view. This supports decisions in the design phase and enables a more successful end result.

The developed framework uses a state machine as replacement for the planner commonly found in other hybrid architectures. When applied to the development of a specific robot, the state machine is programmed to implement a flow diagram describing the top-level behaviour of the robot. The framework will be further developed and used when integrating work by the fourteen partners in the CROPS project.

Acknowledgements

This work is partly funded by the European Commission (CROPS GA no 246252). Thanks to Peter Hohnloser for implementing the state machine used in the robot architecture.

References

Baur J, Pfaff J, Ulbrich H, Villgrattner T (2012) *Design and development of a redundant modular multipurpose agricultural manipulator*, submitted to IEEE/ASME International Conference on Advanced Intelligent Mechatronics, July 2012

International Organization for Standardization (2007) *ISO/IEC 42010:2007 Systems and software engineering - Recommended practice for architectural description of software-intensive systems*, ISOIEC 42010 IEEE Std 14712000 First edition 20070715, International Organization for Standardization, available at http://ieeexplore.ieee.org

Kruchten P (1995) *Architectural Blueprints - The "4+1" View Model of Software Architecture*, IEEE Software, Vol. 12, No. 6. (November 1995), pp. 42-50

Matarić M J (2002) *Situated Robotics*, invited contribution to the Encyclopedia of Cognitive Science, Nature Publishing Group, Macmillan Reference Limited

Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Eric Berger RW, Ng A (2009) *ROS: An Open-Source Robot Operating System*, In ICRA Open-Source Software workshop