Industrial Robot: An International Journal

Emerald Article: A software framework for agricultural and forestry robots

Thomas Hellström, Ola Ringdahl

Research article

A software framework for agricultural and forestry robots

Thomas Hellström and Ola Ringdahl
Department of Computing Science, Umeå University, Umeå, Sweden

Abstract

Purpose – The purpose of this paper is to describe a generic software framework for development of agricultural and forestry robots. The primary goal is to provide generic high-level functionality and to encourage distributed and structured programming, thus leading to faster and simplified development of robots. A secondary goal is to investigate the value of several architecture views when describing different software aspects of a robotics system.

Design/methodology/approach – The framework is constructed with a hybrid robot architecture, with a static state machine that implements a flow diagram describing each specific robot. Furthermore, generic modules for GUI, resource management, performance monitoring, and error handling are included. The framework is described with logical, development, process, and physical architecture views.

Findings – The multiple architecture views provide complementary information that is valuable both during and after the design phase. The framework has been shown to be efficient and time saving when integrating work by several partners in several robotics projects. Although the framework is guided by the specific needs of harvesting agricultural robots, the result is believed to be of general value for development also of other types of robots.

Originality/value – In this paper, the authors present a novel generic framework for development of agricultural and forestry robots. The robot architecture uses a state machine as replacement for the planner commonly found in other hybrid architectures. The framework is described with multiple architecture views.

Keywords Robots, Agriculture, Forestry, Computer software, Agriculture industries, Industrial robotics, Forestry industries, Software architecture, Architecture view, Middleware

Paper type Research paper

1. Introduction

Construction of software has become an increasingly complex and time-consuming part of robot development. A major reason for this is increasingly higher demands on accuracy and functionality. While early generations of industrial robots were often governed by simple PID controllers, high-precision robot arms of today need much more complex control algorithms to achieve promised speed and positional accuracy. The robots of today also offer much more advanced functionality than just a few years ago. Robots are often expected to perform not only one single task, but to combine several tasks in a flexible manner depending on varying environmental conditions. Sensing has become increasingly important, with accompanying problems and requirements related to data acquisition, data analysis and modelling. A robot that is expected to do useful work in common unstructured outdoor environments has to act timely and intelligently to external factors such as humans, animals, wind, and sudden changes in light and ground conditions. The factors mentioned above are particularly relevant for construction of agricultural and forestry robots.

Development of agricultural and forestry robots typically requires software for several different parts of the robots, and for different purposes; custom designed sensors often contain embedded computers for data acquisition, data processing and communication. Robot arms and grippers often contain low-level controllers accepting set values for joint angles and/or coordinates. The top-level control program typically contains driver routines for communication with sensors and actuators, and algorithms for sensing, planning, motion control, and gripping. An advanced robot system may also offer off-line programs for systems configuration, calibration, data analysis and learning.

In this paper, we present a software framework for development of agricultural and forestry robots within the CROPS project (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 is currently used for the development of several robots with slightly different tasks, which means that it has to be general and possible to configure in

The current issue and full text archive of this journal is available at **www.emeraldinsight.com/0143-991X.htm**

Industrial Robot: An International Journal
40/1 (2013) 20–26

[DOI 10.1108/01439911311294228]

This work is partly funded by the European Commission (CROPS GA no 246252). Thanks to Peter Hohnloser for the C++ implementation of the state machine used in the robot architecture. This paper is revised and updated from an original article published at RHEA-2012, Pisa, Italy, September 19-21, 2012.

several respects. The main contribution in the paper is the hybrid architecture of this framework. It uses a state machine to replace the planner commonly found in other hybrid architectures, and coordinates execution of behaviours that can be purely computational and/or acting, in the sense that they control physical actuators. 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.

This paper is organized as follows. In Section 2, classical paradigms for existing robot architectures are described together with an overview of architecture views. Section 3 describes the developed architecture using these views. Section 4 summarizes the paper with conclusions and directions for future work.

2. Background

In the field of mobile robotics, several "robot architectures" have been proposed over the years. They all describe different approaches to how to organize sensing, planning, motion, cognition, and control. Most architectures follow one of these classical paradigms:

- *Deliberative (1967-1990)*. Inspired by classical artificial intelligence, world models, and classical planning. Sensor data is used to create a world model, in which a planner solves the current problem. The resulting plan is then executed through the robot's actuators. A typical deliberative architecture is the Nested Hierarchical Controller (NHC) (Meystel, 1990).
- *Reactive (1988-1992)*. Inspired by animals' behaviour. Emphasizes that complex and fast behaviours can be achieved by simple control mechanisms. Actions are directly computed and performed as a function of current sensor data. Typical reactive architectures are the subsumption architecture (Brooks, 1986) and DAMN (Rosenblatt, 1997).
- *Hybrid (1992-present)*. Combination of the two approaches above. Actions are reactive and are activated and adjusted by deliberative functions. Typical hybrid architectures are AuRA (Arkin and Balch, 1997), and Saphira (Konolige and Myers, 1998). Like most hybrids, both AuRA and Saphira include a planner in the deliberate part.

For an overview of these, and other robot paradigms, see for instance (Matarić, 2002). In this paper, we propose a hybrid robot architecture that uses a state machine instead of a planner to coordinate the necessary sub-tasks necessary to execute a certain specified task. State machines were suggested for behaviour-based robotics by Brooks (1986). An overview of how they can be used for sequencing behaviours is given in (Arkin, 1998, pp. 81-82). Our design and usage of state machines is partly inspired by SMACH, a hierarchical concurrent state machine library implemented in Python (Bohren *et al.*, 2011).

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.* Definition of top-level functionality, general modules and their relationship.
- *Development view.* Programming and software development. For instance object orientation, component based systems, model driven design.
- *Process view.* Issues related to concurrency, distribution (versus centralization), and communication modes (such as point-to-point or publish-subscribe).
- *Physical view.* Placement of physical components and physical connections between these components.

These architecture views describe important and complementary aspects of a system. In addition to the architecture of the entire system, each individual subsystem in a robot, for instance a vision system, may have its own architecture.

3. Proposed architecture

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.

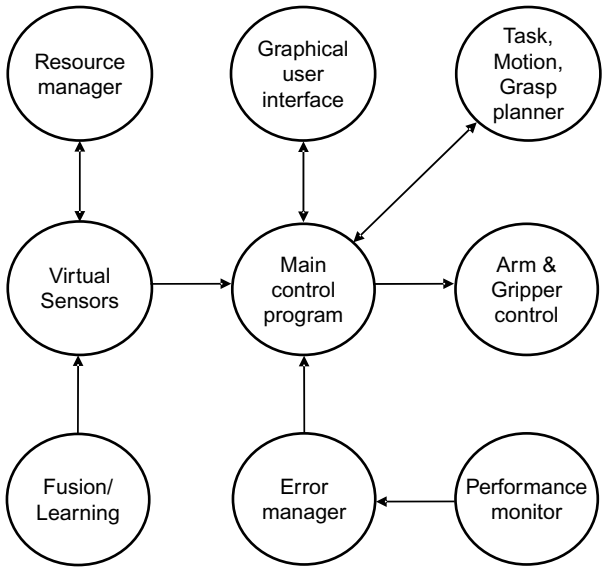
3.1 Logical view

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 in 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 (GUI)*. 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).

The listed modules and primary flow of communication is shown in Figure 1.

Figure 1 Logical view of the architecture for the developed framework, describing top-level functionality, general modules and their relationship



3.2 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 robot operating system (ROS) 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). The example in Figure 2 shows a small part of the system configured for a fruit harvesting robot in the CROPS project.

3.3 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 (Figure 3). 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 (red box) is very flexible. This means that processor-intensive computations, if necessary, can be moved to separate laptops without any changes in the programs. All laptops, sensors, and actuators communicate through a standard ethernet bus. A gateway to CAN equipment is, if necessary, available through a CAN/ethernet interface.

3.4 Robot architecture

We further develop the architecture by specifying and designing how the different modules function and interact. This development is affected by design choices described in the logical, development, process and physical views. The result is a robot architecture that follows the hybrid paradigm. The logical view of this architecture is shown in Figure 4.

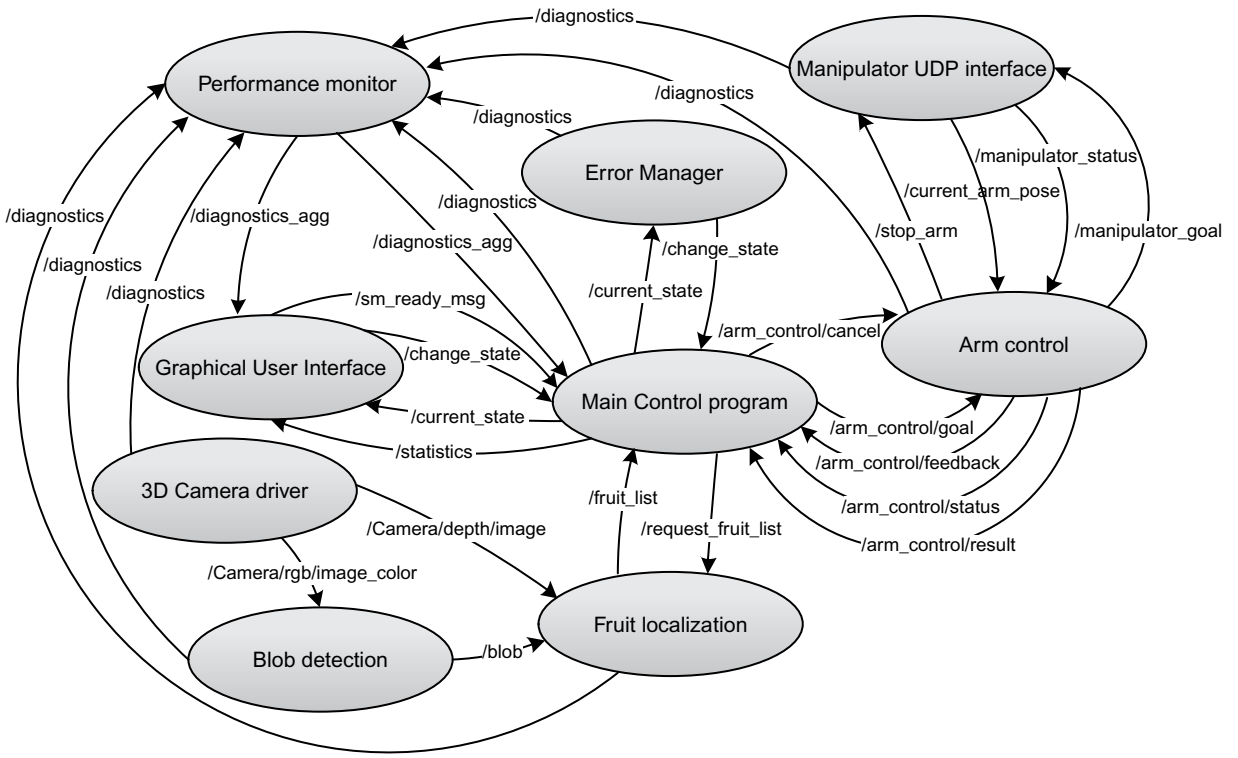
The major refinement concerns the design of the Main Control Program (Figure 1). A state machine replaces the planner component commonly found in hybrid architectures. This is viewed as a good solution for the CROPS robots, and for other robots where there is no need for planning in the sense of problem solving or finding novel solutions to new tasks. This is the case if the behaviour of the robot is simple and well defined, and can be described in for instance a flow diagram. A part of a simplified flow diagram for a sweet-pepper harvesting robot is shown in Figure 5. This flow diagram is transformed into the state diagram in Figure 6. The complete state diagram is implemented as a state machine in the robot. At each time step, the current state returns an "outcome". Depending on the "outcome", the state machine moves to another state or remains in the same state at next time step. In Figure 6, the label of each arrow represents a specific "outcome" of the state at the root of the arrow. The tip of the arrow points at the state that will become active at next time step.

The proposed architecture differs from the standard hybrid model also 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 integrating user interfaces and error handling. Behaviours can be computational and/or acting. The computational ones are not connected to any actuators while the acting ones are.

The relation between states and behaviours is shown in Figure 7. In the figure, the "Move_home" state is connected to "Move arm behaviour", which in turn communicates with the "Manipulator" node, which is connected to the physical manipulator. The outcome and other results computed in a behaviour are communicated to the corresponding state and can also be stored such that they are made available to other states and behaviours. One example is the computation and usage of coordinates of fruits. The "Locate_fruits" state communicates with "Fruit localization behaviour", which computes locations of fruits by using data delivered by the "Camera" node and the "Range sensor" node. The result is stored such that the states that decide on picking order (Select fruit) and move to a fruit (Pick fruit) can access them (Figure 6).

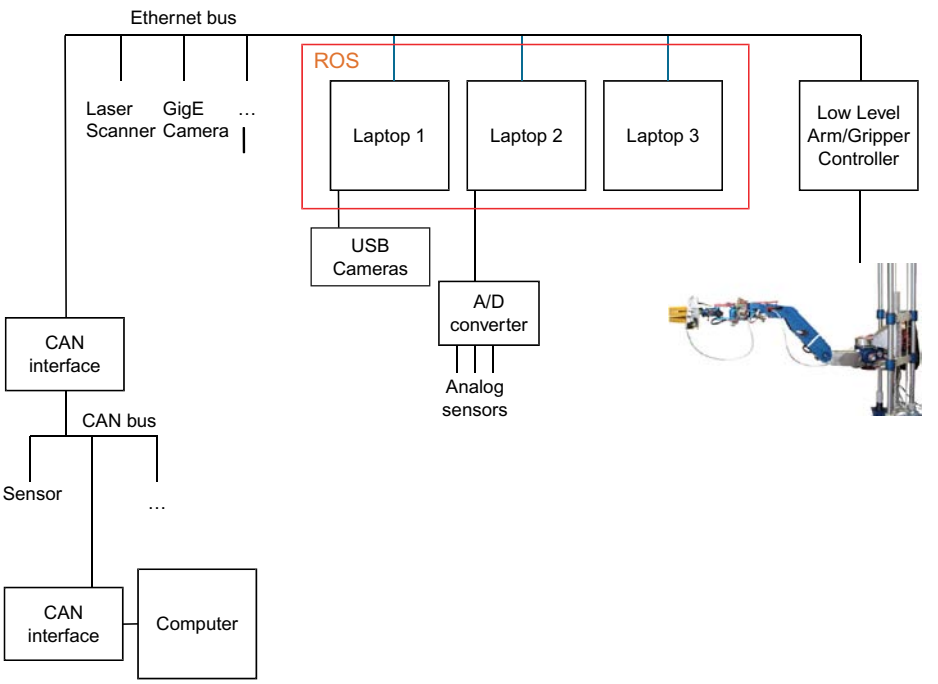
Properties of states, and conditions for transitions between states, are defined in a "Transition file" in text format. This file is read by the state machine at run time and provides a flexible and easy to use configuration tool when developing new robots. One example is shown in Figure 8. The contents of the transition

Figure 2 Development and process view of the architecture for parts of the framework



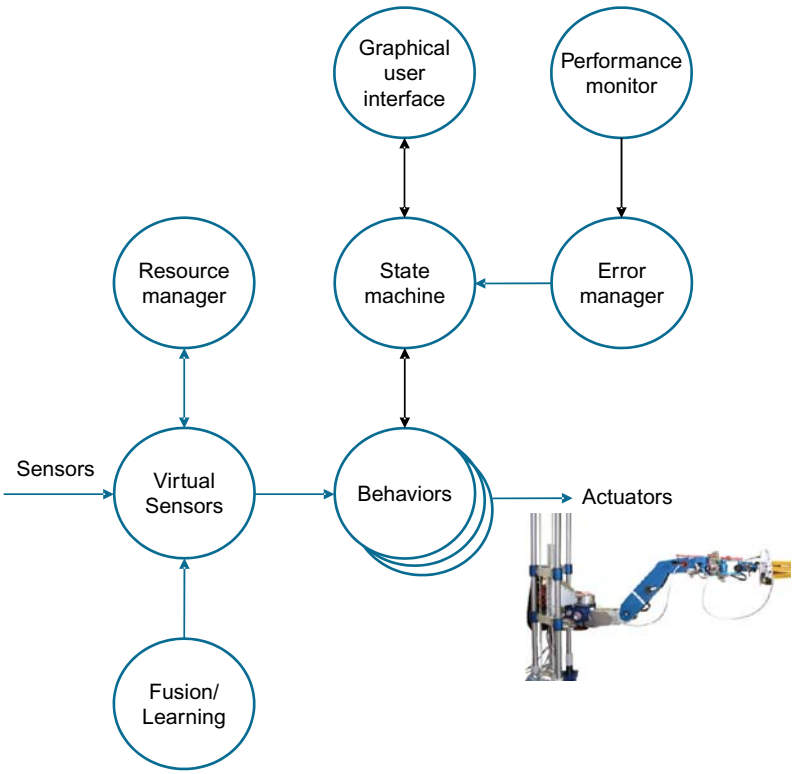
Note: Circles represent ROS nodes and arrows ROS message passing

Figure 3 Physical view of the architecture for the developed framework



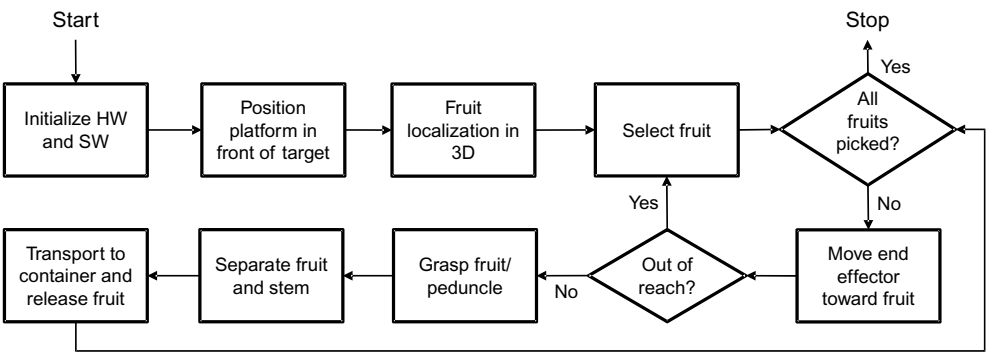
Notes: All software within the red box communicates via ROS messages and can be easily physically relocated to different laptops without any program changes; CAN equipment can be added through the CAN/ethernet interface

Figure 4 Logical view of the hybrid architecture for the developed framework



Note: A state machine that configures and activates behaviours replaces the planner found in most hybrid architectures

Figure 5 Part of a flow diagram for a sweet-pepper harvesting robot



file, shown in the rectangle, specify that the state machine will stay in state "Move_home" as long as the outcome of the state is "Moving to home". When the outcome changes to "Home", the state will change to "Locate_fruits".

The "GUI" 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. For the fruit harvesting robots, one important virtual sensor communicates with one or several cameras and identifies fruits in the images. The output of this virtual sensor is read and further processed by the "Locate fruits" behaviour. The "Resource manager" provides customization of the virtual sensors such that the system can be adapted to varying environmental conditions (such as changing weather) or changing tasks (such as harvesting a new type of fruit). 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. Another example is if the robot arm fails to reach

Figure 6 State diagram representing the flow diagram in Figure 5

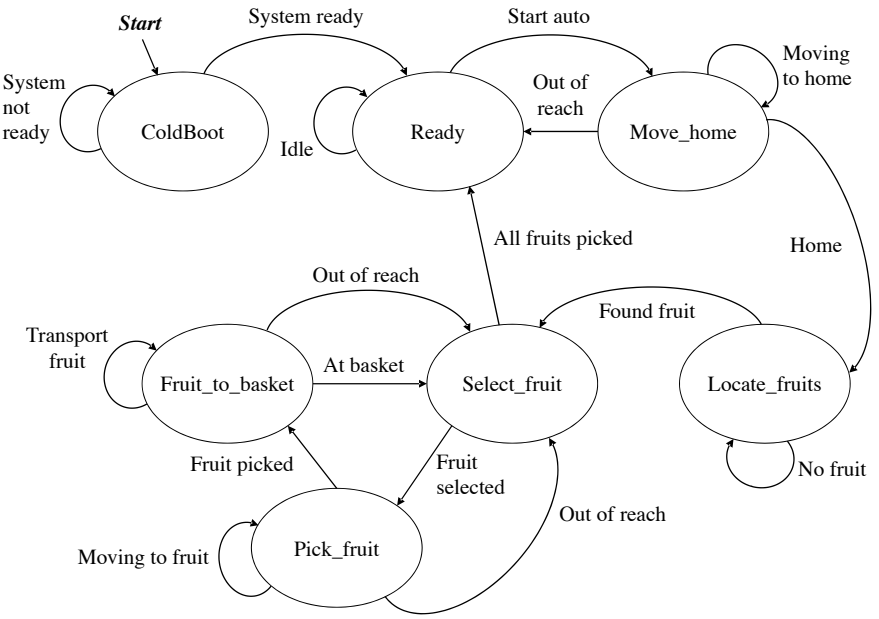
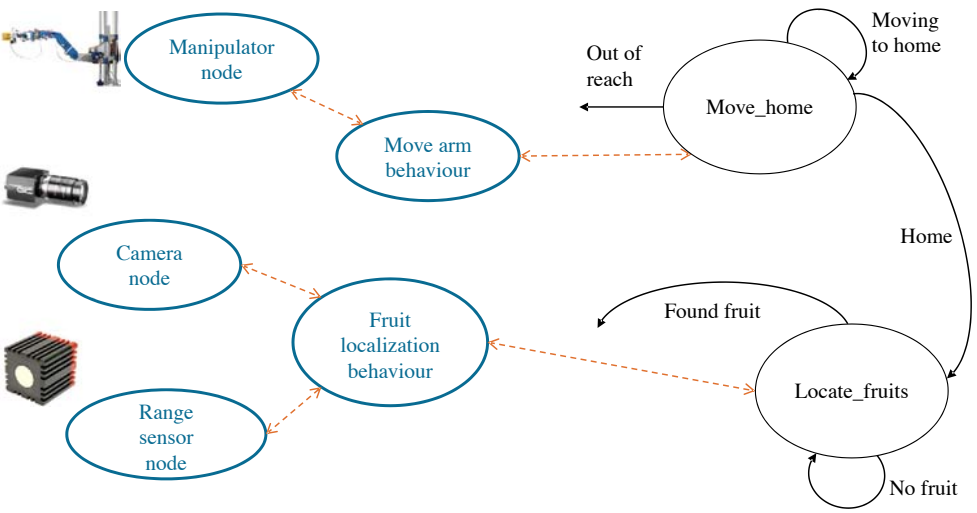


Figure 7 Each state in the state machine is normally connected to a behaviour, which performs computations and/or communicates with sensors and actuators

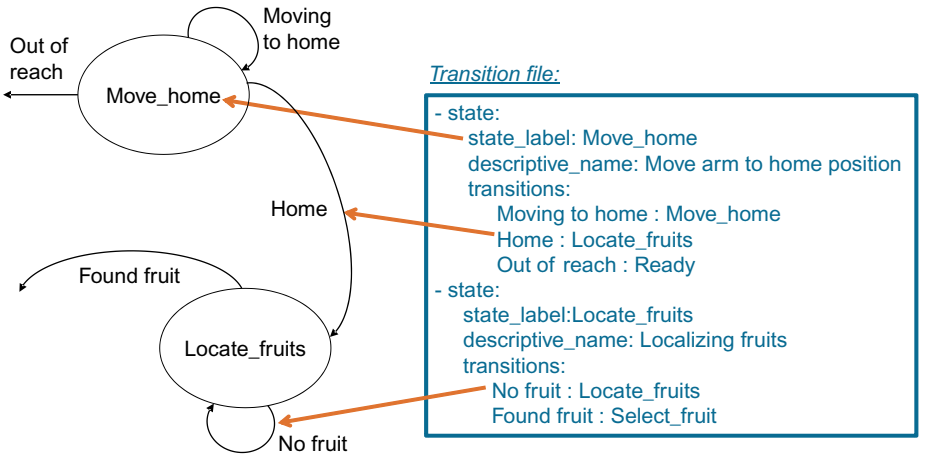


Notes: Blue ovals are ROS nodes while black ovals are states in the state machine; red arrows represent ROS message passing

the coordinates of a fruit selected for harvesting (for instance if an obstacle blocks the motion, or if the fruit is no longer visible for the robot). As shown in Figure 6, the state "Pick_fruit" may detect this error by returning outcome "Out of reach", causing a state shift to "Select_fruit" where another fruit is selected for harvesting.

Other errors are independent of the current state, and are more efficiently detected and dealt with at system level. The "Performance monitor" monitors the status of communication lines and hardware such as sensors and computers. If an error is detected, messages are sent to the "Error manager" that takes appropriate action. One example is a camera that suddenly stops functioning. This may be detected by the "Performance monitor" and forwarded to the "Error manager", which then stops the state machine (and thereby the robot) and prints an error message through the GUI. Another example is an emergency stop of the robot arm if something or someone gets in the way. The "Error manager" may also tell the state machine to change state if an error is detected. The "Error manager" may sometimes need complex decision-making, and the correct action for error recovery may depend on the current state. Just as we use a static state machine to control the normal execution of tasks, the "Error manager" may be implemented as a state machine.

Figure 8 State transitions depend on "outcomes" from the states, and are defined in a "Transition file" in text format



Note: This file is read by the system at run-time

4. Results and conclusions

We have presented a software framework for development of agricultural robots. Although the design is guided by the specific needs of a harvesting agricultural robot, the result is believed to be of general value for development also of other types of 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 encourages and supports decisions already 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 approach encourages and supports a modular division of work, which is crucial especially for large development teams. The generic, yet configurable, modules for GUI, resource management, performance monitoring, and error handling save considerable development time. All modules, including the state machine, are implemented in C++ using ROS. The framework is currently used when integrating work by 14 partners in the CROPS project. It will also be used for a similar integration task within the INTRO project (http://introbotics.eu).

References

Arkin, R.C. (1998), *Behaviour-Based Robotics*, MIT Press, Cambridge, MA.

Arkin, R.C. and Balch, T. (1997), "AuRA: principles and practice in review", *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 9 Nos 2/3, pp. 175-89.

Baur, J., Pfaff, J., Ulbrich, H. and Villgrattner, T. (2012), "Design and development of a redundant modular multipurpose agricultural manipulator", paper presented at IEEE/ASME International Conference on Advanced Intelligent Mechatronics.

Bohren, J., Rusu, R.B., Jones, E.G., Marder-Eppstein, E., Pantofaru, C., Wise, M., Mosenlechner, L., Meeussen, W. and Holzer, S. (2011), "Towards autonomous robotic butlers: lessons learned with the PR2", *IEEE International Conference on Robotics and Automation* (*ICRA*), pp. 5568-75.

Brooks, R. (1986), "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, Vol. 2 No. 1, pp. 14-23 (legacy, pre-1988).

International Organization for Standardization (2007), *ISO/IEC 42010:2007: Systems and Software Engineering – Recommended Practice for Architectural Description of Software-Intensive Systems*, International Organization for Standardization, Geneva, ISOIEC 42010 IEEE Std 14712000 First edition 20070715, available at: http://ieeexplore.ieee.org

Konolige, K. and Myers, K. (1998), "The Saphira architecture for autonomous mobile robots", in Kortenkamp, D., Bonasson, R. and Murphy, R. (Eds), *Artificial Intelligence and Mobile Robots*, MIT Press, Cambridge, MA.

Kruchten, P. (1995), "Architectural blueprints – the '4+1' view model of software architecture", *IEEE Software*, Vol. 12 No. 6, pp. 42-50.

Matarić, M.J. (2002), "Situated robotics", *Encyclopedia of Cognitive Science*, Nature Publishing Group, Macmillan Reference Limited, London.

Meystel, A. (1990), "Knowledge based nested hierarchical control", in Saridis, G. (Ed.), *Advances in Automation and Robotics*, Vol. 2, JAI Press, Greenwich, CT, pp. 63-152.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Eric Berger, R.W. and Ng, A. (2009), "ROS: an open-source robot operating system", paper presented at ICRA Open-Source Software Workshop.

Rosenblatt, J.K. (1997), "DAMN: a distributed architecture for mobile navigation", *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 9 Nos 2/3, pp. 339-60.

Corresponding author

Thomas Hellström can be contacted at: thomash@cs.umu.se