Robotics **2014**, *3*, 207-234; doi:10.3390/robotics3020207

OPEN ACCESS

robotics

ISSN 2218-6581

www.mdpi.com/journal/robotics

Article

Towards an Open Software Platform for Field Robots in Precision Agriculture

Kjeld Jensen [1,*], Morten Larsen [2], Søren H. Nielsen [1], Leon B. Larsen [1], Kent S. Olsen [1] and Rasmus N. Jørgensen [3]

[1] Faculty of Engineering, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark; E-Mails: soeni05@gmail.com (S.H.N.); lelar09@student.sdu.dk (L.B.L.); keols09@student.sdu.dk (K.S.O.)

[2] Conpleks Innovation, Fælledvej 17, 7600 Struer, Denmark; E-Mail: morten.larsen@conpleks.com

[3] Institute of Engineering, Aarhus University, Nordre Ringgade 1, 8000 Aarhus, Denmark; E-Mail: rnj@iha.dk

* Author to whom correspondence should be addressed; E-Mail: kjen@mmmi.sdu.dk.

Received: 20 September 2013; in revised form: 12 May 2014 / Accepted: 18 May 2014 / Published: 13 June 2014

Abstract: Robotics in precision agriculture has the potential to improve competitiveness and increase sustainability compared to current crop production methods and has become an increasingly active area of research. Tractor guidance systems for supervised navigation and implement control have reached the market, and prototypes of field robots performing precision agriculture tasks without human intervention also exist. But research in advanced cognitive perception and behaviour that is required to enable a more efficient, reliable and safe autonomy becomes increasingly demanding due to the growing software complexity. A lack of collaboration between research groups contributes to the problem. Scientific publications describe methods and results from the work, but little field robot software is released and documented for others to use. We hypothesize that a common open software platform tailored to field robots in precision agriculture will significantly decrease development time and resources required to perform experiments due to efficient reuse of existing work across projects and robot platforms. In this work we present the FroboMind software platform and evaluate the performance when applied to precision agriculture tasks.

Keywords: field robots; precision agriculture; FroboMind; ROS

---

1. Introduction

Robotics in precision agriculture has the potential to improve competitiveness and increase sustainability compared to current crop production methods [1], and has become an increasingly active area of research during the past decades. Tractor guidance using Global Navigation Satellite System (GNSS) based sensor systems for route following and local sensor systems for accurate in-row navigation in row crops and orchards have already reached the market. One example is the John Deere iTEC Pro which supports GNSS based steering in straight and curved rows and at headlands while controlling speed and performing active implement guidance. Another example is the Claas Cam Pilot system which navigates a tractor through row crops using 3D computer vision to detect the location of the crop rows. Early prototypes of smaller field robots performing precision agriculture tasks without human intervention also exist [2–6]. But research in advanced cognitive [7] perception and behaviour that are required to enable a more efficient, reliable and safe autonomy becomes increasingly demanding. The level of complexity in an unstructured, dynamic, open-ended and weather influenced environment like a crop field or an orchard is high, and the size and complexity of the software needed to perform experiments impose ever greater demands on research groups. A lack of collaboration between the research groups contributes to the problem. Scientific publications are published on findings and results in precision agriculture, but little field robot software has actually been released, published and documented for others to use. We hypothesize that a common open software platform tailored to field robots in precision agriculture will significantly decrease the development time and resources required to perform field experiments due to efficient reuse of existing work across projects and robot platforms. The aim of this work is to establish such a software platform and evaluate the performance when applied to different precision agriculture tasks and field robots.

Related Work

The literature contains numerous references to relevant proposals and implemented solutions within robot software architectures, frameworks, middlewares, development environments, libraries *etc*. Below is a brief review of some of the best known and widely used solutions. Table 1 compares the middleware specifications.

CARMEN Robot Navigation Toolkit from the Carnegie Mellon University (CMU) [8] is a modular software library for mobile robot control. It provides interfaces to a number of robot platforms and sensors, a 2d simulator and algorithms for localization, mapping, path planning *etc*. The architecture features 3 layers, the lowest layer contains hardware interfaces and collision detection, the middle layer localization and navigation, and the highest layer contains all high level tasks. Inter Process Communication (IPC), another project by CMU, is used for sending data between processes based on TCP/IP sockets.

CLARAty (Coupled Layer Architecture for Robotic Autonomy) [9,10] is a framework for generic and reusable software for heterogeneous robot platforms developed by the Jet Propulsion Laboratory. CLARAty is a two-tiered coupled layer (decision and functional) architecture. The functional layer is a modular software library which provides an interface to the robot system and contains algorithms for

low- and mid-level autonomy. The decision layer builds on top of this adding high-level autonomy to achieve mission goals. CLARAty consists of a public and a private repository.

Table 1. Comparison of middleware specification. The year listed under *Update* indicates the latest official release or substantial update.

	CARMEN	**CLARAty**	**MRDS**	**ORCA**	**Orocos**	**Player**	**ROS**
Updated	2008	2007	2012	2009	2014	2010	2014
Main Languages	C	C++	C# VPL	C++	C++	C++ TCL Java Python	C++ Python Lisp
Primary Platforms	Linux	VxWorks Linux Solaris	Windows	Linux	Linux Windows OSX	Linux Solaris BSD OSX	Linux
Component Interface	IPC	Unknown	CCR, DSS	ICE	CORBA	TCP sockets	XMLRPC
License	GPLv2 /BSD	Proprietary /Closed	Commercial /Academic	LGPL/ GPL	LGPL	GPLv3	BSD 3-Clause

MRDS (Microsoft Robotics Developer Studio) [11] is a development environment for robot control and simulation. MRDS has support for a number of programming languages including it's own platform specific language Visual Programming Language (VPL). It supports a wide range of robotics hardware platforms and integrates a fully featured simulation environment. The component interface is based on the .NET Concurrency and Coordination Runtime (CCR) library for managing asynchronous, parallel tasks using message-passing and Decentralized Software Services (DSS), a lightweight .NET-based runtime environment.

Orca [12] is an open-source software framework for developing component-based robotic systems. The aim of Orca is to promote software reuse through definition of interfaces, providing component libraries and maintaining a public component repository. The intended use is for both commercial applications and research environments. The Orca project is a branch out from Orocos, one main difference is that Orca uses the Internet Communications Engine (ICE) [13].

Orocos (Open Robot Control Software) [14] contains portable C++ libraries for advanced machine and robot control. Orocos builds upon the open source Common Object Request Broker Architecture (CORBA) middleware. Orocos is in active development and contains extensions that support other frameworks including ROS.

Player [15] is one of the most used robot control interfaces and supports a wide variety of robots and components. The Player architecture is based on a *network server* which runs on the robot platform and provides a TCP socket interface to the robot. The *client program* is thus able to read data from sensors, write commands to actuators *etc*. by connecting to the socket. Player coexists with Stage which is a 2d multiple robot simulator and Gazebo which is 3d multiple robot simulator based on a physics engine.

ROS (Robot Operating System) [16] is a flexible framework for writing robot software. It provides services, libraries and tools for building robotics applications. Examples are hardware abstraction and

device control, interprocess communication, multi-computer environment support *etc*. ROS is in active development and is maintained by Open Source Robotics Foundation who provides access to a large repository of available components and maintains a list of external repositories provided by the growing community. The ROS core code and most ROS packages are released under the BSD 3-Clause License which facilitates commercial use of the code.

In terms of robot software architectures the above mentioned *Carmen* and *CLARAty* each use their own architecture, wheres a *MRDS*, *ORCA*, *Orocos* and *ROS* don't specify or endorse a certain architecture. *Player* does not specify an architecture beyond the two-layer *Network server* and *Client program*. Several relevant but lesser known robot software architectures are described in the literature:

Ref [17] argues that the sense-model-plan-act paradigm used in most architectures is unable to react in a dynamical environment because it depends heavily on the model, and that a behaviour-based approach such as the subsumption architecture [18] is limited by the purely reactive behaviours and does not perform well when carrying out complex tasks. A hybrid is thus presented containing three layers: Hardware layer, Reactive layer and Deliberative layer. The reactive layer is grouped into a perception module (localization and mapping) and an action module (navigation and actuation). The deliberative layer contains high level mapping and path planning.

Ref [19] introduces Agricultural Architecture (**Agriture**), a control architecture designed for teams of agricultural robots. Agriture consists of three layers, the physical layer which is either the robot or the Stage/Gazebo simulators, an Architecture middleware based on *Player*, *Java Agent Development Framework* and *High Level Architecture*, and a Distributed application layer for the application software.

Ref [20] proposed **Agroamara**, a hybrid agent of behaviour architecture for autonomous farming vehicles. Here *agent of behaviour* describes computational and control processes addressed to reach or to maintain a goal, with perceptual, deliberative and reactive abilities. The architecture groups the agents into perceptual and motor agents and uses layers to describe the information flow and hierarchy of activation of the agents. Data sharing is handled through shared memory and peer to peer message parsing. Multiprocessing is supported using winsocket.

Ref [21] describes the software architecture of the Autonomous Mobile Outdoor Robot (**AMOR**) which won the first price in autonomous driving in the European Land Robot Trial (ELROB) 2007 for urban and non-urban terrain. The robot software is not publicly available, however the paper does provide a good introduction to the design as well as the navigation modules. Intercommunication is handled by a *Virtual Sensor Actor Layer* that use TCP sockets and unifies the access to all sensors and actors. The main modules of the high level software is *global model* (map database), *global path planning* (deciding plans for the indended behaviour), *local model* (based on laser scanner and camera data), *local path planning* and *basic reflexes*.

Ref [22] introduces **Mobotware**, a plug-in based framwork for mobile robots. Mobotware is divided into a hard- and a soft realtime section. Each section is decomposed into layers of increasing abstraction: Hardware abstraction, reactive execution, deliberative perception and planning. The framework has three core modules, a *Robot Hardware Daemon* providing hardware abstraction, a *Mobile Robot Controller* handling real-time closed loop controlling, and *Automation Robot Servers* handling sensor processing, mission planning and management.

Ref [23,24] propose a system architecture to enable behavioural control of an autonomous tractor based on the subsumption architecture [18]. This work is related to the Software Architecture for Agricultural Robots (**SAFAR**) project with the purpose to develop a set of designs, tools and resources to promote the development of agricultural robots. SAFAR is based upon MRDS and is closed source but supports a Python scripting engine.

In 2005 the Stanford robot **Stanley** won the DARPA Grand Challenge by driving autonomously for 131 miles along an unrehearsed desert trail [25]. The robot software is not publicly available, however the paper provides information on the software architecture and design considerations. Stanley contains approximately 30 software modules organized in 6 layers representing sensor interfaces, perception, planning and control, vehicle interface, user interface and global services. All modules are executed individually without interprocess synchronization and all data are globally time stamped. Interprocess communication is handled using publish-subscribe mechanisms based on the CMU IPC kit.

Table 2 compares some properties of the reviewed architectures with respect to this work. The data is based on the reviewed publications and information publicly available on the web. It should be noted that all the reviewed publications contains valuable and relevant knowledge, but at the same time the table underpins the need for collaboration between the research groups in precision agriculture with respect to software reuse as discussed in the problem description.

Table 2. Comparison of robot software architectures with respect to the problem domain and hypothesis of this work. In the columns Agricultural applications, Multiple platforms and Multiple users *Yes* means that practical field trials have taken place. (*Yes*) in Open source means that not all software has been released and/or the license is not permissive free.

	Agricultural Applications	**Multiple Platforms**	**Multiple Users**	**Open Source**	**Updated Recently**
CARMEN	Yes	Yes	Yes	(Yes)	No
CLARAty	No	Yes	Yes	(Yes)	No
Agriture	No	No	No	No	No
Agroamara	Yes	No	No	No	No
AMOR	No	No	No	No	No
Mobotware	Yes	Yes	Yes	No	Yes
SAFAR	Yes	Yes	No	No	No
Stanley	No	No	No	No	No

2. The FroboMind Software Platform

2.1. Design Goals

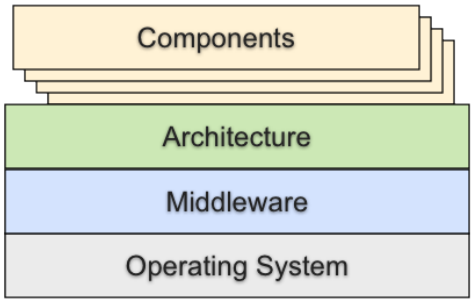
The software platform presented in this work has been named FroboMind which is a contraction of *Field Robot Mind*. It is based on design goals that promote reuse, support projects of varying size and complexity, and facilitate collaboration:

- **Modularity:** It must provide a proper modular structure to ease the development of new components and facilitate reuse. Interfaces between modules must be well defined, however still allowing researchers and developers to experiment freely.
- **Extensibility:** The design must support a wide variety of applications from a student working with signal processing or route planning algorithms to large scale field experiments conducted by research groups with several robots interacting online with each other and human operators.
- **Scalability:** It must support online computation of high load algorithms such as computer vision and mapping with respect to both memory and processing power requirements. This includes scaling to distributed applications on networked heterogeneous platforms.
- **Software reuse:** There must be a strong focus on minimizing resource requirements in the development of new applications by facilitating reuse of software. The future workload of maintaining the software platform is also an important consideration. When establishing the software platform, existing work should be reused to the extent possible. To prevent maintenance problems arising due to the existing work, this must be under active development and supported by a community.
- **Open source:** All core components of the software platform must be released under a permissive free software license to keep it free for others to use, change and commercialize upon. This is necessary to facilitate collaboration with industrial partners.

2.2. FroboMind

The FroboMind software platform structure consists of four parts as illustrated in Figure 1. The *Operating System* provides basic facilities for application execution, file handling, hardware interfacing, communication, networking *etc*. The *Middleware* provides services like timing and communication between software components in a distributed system which simplifies the software development significantly. The *Architecture* organizes the software components into layers and modules with well defined interfaces. The modularity eases the development proces and facilitate efficient software reuse. The *Components* are the actual building blocks used for field robot applications and reused across projects and robot platforms. In the following sections each of the four parts of the software platform structure is described in detail.

Figure 1. Overview of the FroboMind software platform structure.



2.3. Operating System

When choosing an operating system for FroboMind the design goals, in particular the *Software reuse* and *Open source* goals, must be taken into consideration.

In robotics software implementations a Real-Time Operating System (RTOS) dedicated to embedded systems is typically used. The reason is that RTOS improves the performance for components that have hard or near real-time requirements, and some of the available systems (e.g., VxWorks and QNX) improve the reliability significantly compared to traditional operating systems. However they are usually not open source, and software developed for the RTOS typically runs on the target system only and therefore requires cross compilation, and on-target debugging *etc*. Linux based open source RTOS exist in different varieties. Examples are dedicated distributions where the entire system run as preemptive processes (e.g., RTLinux), add-ons that create a small microkernel where linux runs as a task, and kernel patches or libraries that implement a near real-time environment like the RealTime Application Interface for Linux (RTAI) [26] and CONFIG_PREEMPT_RT [27]. Some of these are well functioning and are utilized in commercial products as well as academic projects. However they are trying to match an RTOS with a fully fledged operating system which is inherently a tradeoff between low latency requirements and the overall efficiency of the system [28]. Other typical challenges of the RTOS are small development teams and a limited community supporting the projects. The RTOS distributions therefore often lag behind regarding support for new software and hardware, and support options in case of problems are limited. In applications where the robot is connected to the internet, network security may be an issue as well due to delayed software updates.

Based on the above considerations the linux distribution Ubuntu was chosen as operating system for FroboMind. Ubuntu is one of the largest and most active linux distributions available, and the annual Long-Term Support versions have guaranteed support for five years after release. Choosing Ubuntu has the major advantage that one can effortlessly run the same operating system on the field robot and a standard laptop used for software development. There is also a huge community of linux software developers which is an advantage when support is needed for a particular problem. Ubuntu only supports soft real-time execution, but in field robotics real-time requirements at the order of 50–200 Hz typically only exist in relation to low level actuator controlling, and this task is often distributed to an external embedded controller that communicates via a serial or network interface. The use of Ubuntu in robot systems with near real-time requirements is evaluated in the experimental section.

2.4. Middleware

As described in the review of related work a number of middlewares designed for robot control exists already. However for FroboMind most of the frameworks and middlewares listed in Table 1 and described elsewhere must be disregarded because they do not comply with the *open source* design goal. Considering in addition the *software reuse* design goal stating that the middleware must be in active development this leaves only two attractive contestants: Orocos and ROS.

Taking a closer look at Orocos it is based upon a few principal components: The Orocos Toolchain which contains the tools required to build Orocos components, the iTaSC framework (generates robot motions using constraints). the rFSM toolkit (Finite State Machines) and the libraries KDL (Kinematics

and Dynamics) and BFL (Bayesian Filtering). The Orocos Toolchain includes the Real Time Toolkit (RTT) framework for developing real-time components in C++.

The ROS middleware offers inter-process communication through a set of facilities: Anonymous publish/subscribe message passing, recording and playback of messages, request/response remote procedure calls and a distributed parameter system. ROS also provides a set of robot libraries such as robot geometry, pose estimation, localization, mapping and navigation *etc*. ROS has a very good support for distributing components between networked computers. ROS is not a realtime framework and is thus not suitable for hard real-time critical software such low level control algorithms. There is, however, support for integration with the Orocos RTT.

Orocos and ROS both fit well with the FroboMind design goals. In favor of Orocos it supports real-time execution and the code base appears to be stable. The requirements for CPU power and memory appear to be lower than for ROS. But Orocos seems more oriented towards low level control than robot application building, and it is not as user friendly as ROS, which also has by far the largest supporting community and the most active development. ROS provides a very detailed documentation using a wiki, videos and examples. Fast prototyping using scripting is available with Python which may speed up application development in many cases. Based on these considerations ROS was chosen as middleware for FroboMind. The requirements with respect to computing power when running ROS and Ubuntu in robot systems with near real-time requirements is evaluated in the experimental section.

2.5. Architecture

The purpose of having a unified architecture shared across different projects and research groups is to facilitate software reuse at the application level. This allows the entire software platform to be transferred to new applications and field robots, the user needs only to add new high level behaviours and low level interface drivers needed for a particular application.

ROS does not propose or endorse a certain architecture or structure of the software, instead the different software components named ROS nodes are able to intercommunicate freely without restrictions to structure. As a result different users tend to use different designs and to some extent different interfaces between components. The user therefore often has to build a new robotic application from scratch, and reuse mainly exists from component level down to snippets of code.

Adding a unified architecture to FroboMind increases the reusability of the components, but at the same time it limits the flexibility that ROS provides. In this section an architecture is proposed with the aim of balancing the tradeoff between reusability and flexibility.

Designing a conceptual architecture that appears intuitive to researchers, engineers and technicians is a challenging task due to their different backgrounds and experience from different projects. A simple basis for mobile robot software has been formulated by the questions: *Where am I?*, *Where am I going?* and *How should I get there?* [29]. The robot localizes itself based on percepts and prior knowledge, it then plans a route which leads towards the mission goal and navigates the route using motion control (Figure 2). However for a field robot in precision agriculture the task is the primary objective and navigation is merely a means to fulfil the task. Performing the task typically involves interaction with an attached or trailed implement, and the conceptual architecture in Figure 2 is therefore insufficient.

Instead a more generic approach based on Intelligent Agents [30] is used for modelling the architecture. An agent is an autonomous entity which perceives its environment through sensors and acts upon that environment through actuators. The action taken by the agent in response to any percept sequence is defined by an agent function. The FroboMind principal architecture consists of perception, decision making and action layers (Figure 3a). The layers have been decomposed to define abstraction levels (Figure 3b).

Figure 2. A conceptual basis for the architecture of mobile robot navigation software described in the literature: *Where am I? Where am I going? How do I get there?*

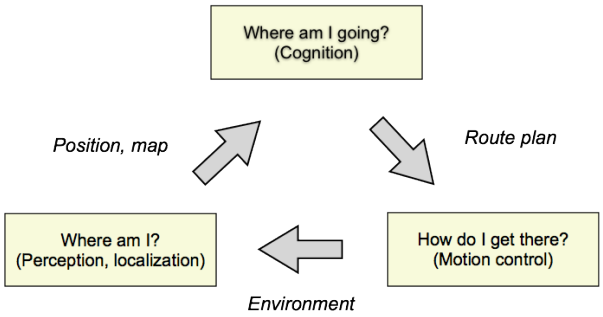
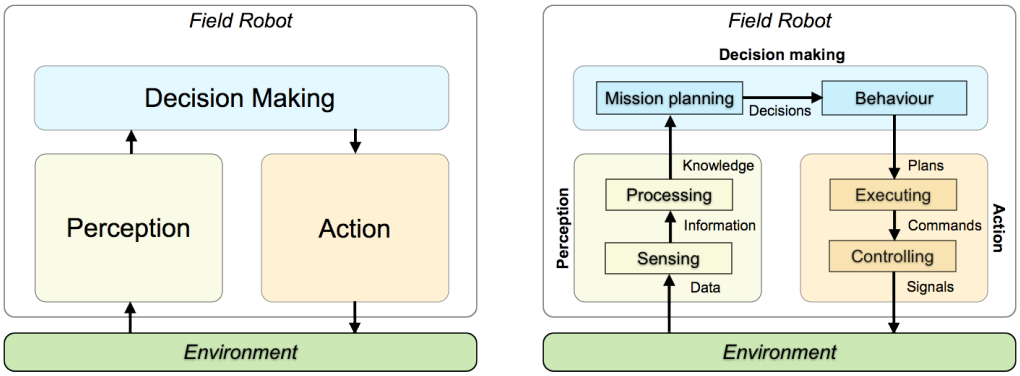


Figure 3. Decomposition of an Artificial Intelligence agent. (**a**) The agent perceives its environment through sensors and acts through actuators. The action taken by the agent in response to any percept sequence is defined by an agent function. (**b**) Decomposition to define data- and hardware abstraction levels.



(**a**) (**b**)

Perception represents the organization, identification and interpretation of sensory information in order to represent and understand the environment [31]. The perception layer has been decomposed into:

- **Sensing:** The robot perceives the surrounding partially observable environment (external percepts) through its sensors and assess the system interior state (internal percepts) through feedback from the platform and implement systems. Together these percepts constitute the available *information*. In addition this layer constitutes the abstraction between sensor hardware and the other parts of the architecture.

- **Processing:** By combining this *information* with previous, shared and a priori knowledge, the field robot maintains a model of the world and system state which constitute the accumulated *knowledge*. Since observables are governed by a certain amount of uncertainty, probabilistic methods are typically utilized to optimize the model.

Decision Making constitutes the cognitive layer and represents an AI agent function which determines the action taken in response to the accumulated *knowledge*. The implementation of the decision making layer may vary depending on application and users may choose to use other methodologies than the current support for model-based, utility-based AI agents and hierarchically organized finite-state machines to describe robot behaviour.

- **Mission Planning:** The robot mission planner continuously monitors the accumulated knowledge as well as any user interaction, and makes on this basis a *decision* about the optimal behaviour which leads towards the fulfillment of the mission. This corresponds to an AI agent utility function described in [30].
- **Behaviour:** The optimal behaviour decided by the mission planner origins from a library of possible behaviours available to the robot and implement. The active behaviour continuously monitors the accumulated knowledge and user interaction and updates action *plans* for the robot and implement action accordingly.

Action The action layer carries out the action defined by the action plans and has been decomposed into:

- **Executing:** The *plans* produced by the active behaviour are executed with respect to time and state. Based on this *commands* are sent to the controlling layer.
- **Controlling:** Commands issued by the executing layer are transmitted to low level controllers within the architecture or to external controller interfaces. This layer constitutes the abstraction between the field robot actuator hardware and other parts of the architecture.

The FroboMind architecture (Figure 4) represents an expansion of the decomposition in Figure 3. This is indicated by the grouping of modules containing software components into the *Perception*, *Decision making*, *Action* and *Safety* layers as well as the data interface between layers and modules indicated by the blue dashed lines. Internal fault diagnosis and incident handling are organized in a separate *Safety* module to minimize potential software errors and thus ensuring a high level of reliability. In order not to clutter the overview it is assumed that any component has access to data accessible by its predecessor. Multiple connections to successors are shown only where relevant to the understanding, and data available globally has not been included.

2.6. Components

The FroboMind components are implemented as ROS packages. Most of the current lower level components at the Perception and Action layers are written in C++ while many of the higher layer components are written in Python.

The components are located in a directory structure where each layer in the architecture (Figure 4) is represented by a directory (Table 3), each module as a subdirectory herein and the components

are located in the module subdirectories. The FroboMind component directory structure and related documentation are located in a repository available through the FroboMind website [32].

Figure 4. The FroboMind Architecture. The modules containing software components are grouped into the *Perception*, *Decision making*, *Action* and *Safety* layers. The blue dashed lines indicate the data interfaces between layers and modules.

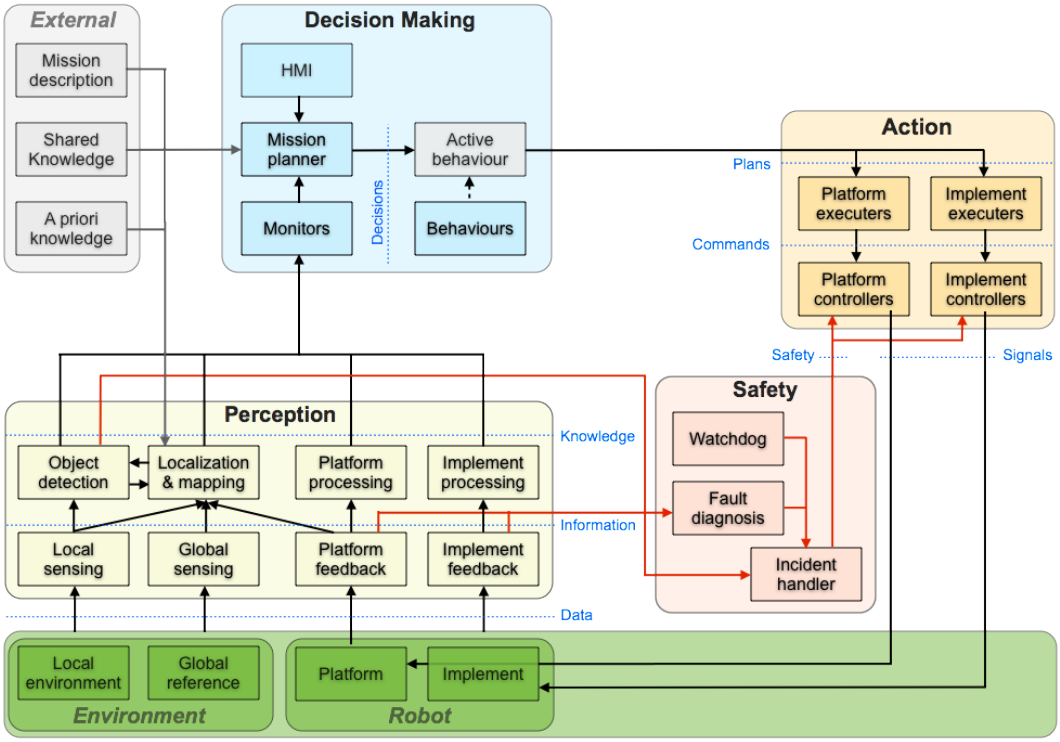


Table 3. The component (top level) directory structure at the FroboMind repository.

Directory	**Content**
/fmSensors	Sensor interfaces and information extraction (hardware abstraction layer)
/fmProcessors	Processing sensor information to obtain knowledge about the robot state.
/fmDecisionMakers	Robot mission, behaviour and HMI components.
/fmExecutors	Executing the current behaviour, e.g., navigation and implement control.
/fmControllers	Low level controllers and actuator interfaces (hardware abstraction layer).
/fmSafety	Watchdog, fault diagnosis and incident handling.
/fmApp	Directories containing components, scripts, launch files *etc*. related to the application of FroboMind to a particular project.
/fmLib	Interface drivers, software libraries *etc*. not represented in the architecture.
/fmTools	Various non-ROS tools and utilities.

2.7. Versions

FroboMind has been in active development since 2011 where the first prototype architecture and components were implemented in ROS Electric on Ubuntu 10.04. Since then FroboMind versions have been created each year with an updated architecture, improved components and compatible with updated versions of Ubuntu and ROS. The version numbers corresponds to the farming season. Version 2014 is the current FroboMind software platform described in this work. It is based on ROS Hydro and Ubuntu 12.04 LTS.

3. Experimental Section

In this work three experiments have been carried out to evaluate the performance of FroboMind when applied to various precision agriculture tasks and field robots. The first experiment evaluates the performance of FroboMind in robot systems with near real-time requirements including requirements with respect to computing power. The second experiment evaluates whether FroboMind significantly decreases the development time and resources required to perform field experiments. The third experiment uses available data from existing projects and robotic platforms that use FroboMind, to evaluate software reuse across these projects.

3.1. Evaluating the Performance of FroboMind in Robot Systems with Near Real-Time Requirements

An experiment was defined to evaluate the performance of FroboMind including ROS and Ubuntu in robot systems with near real-time requirements including requirements with respect to computing power. The experiment is based on a typical application for a field robot in precision agriculture: Autonomous navigation of a predefined route while controlling an attached or trailed implement. Figure 5 shows an example of the FroboMind architecture listing the components required for the autonomous navigation. Each of the components are described in Table 4. All the library functions listed in Figure 5 are implemented in C++.

Figure 5. FroboMind example architecture for autonomous navigation of a route plan.

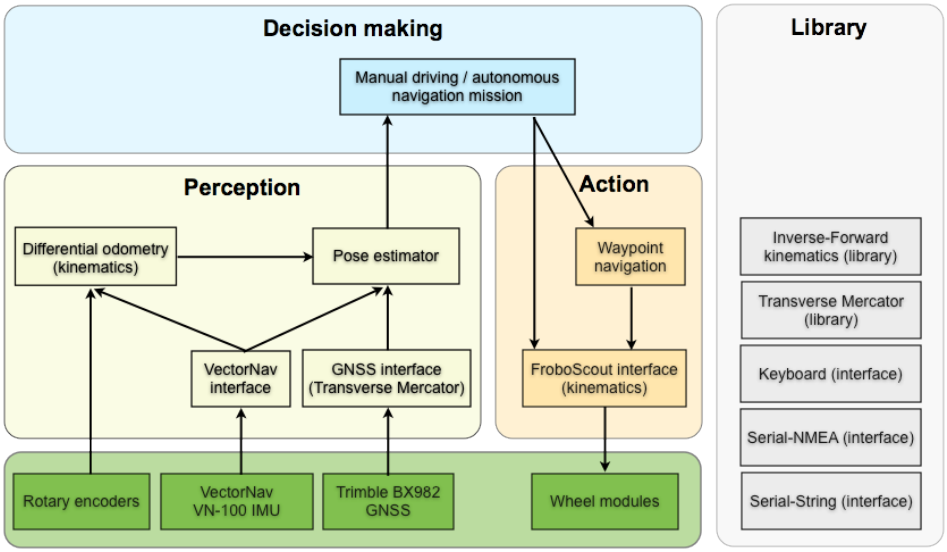


Table 4. FroboMind example components used for autonomous navigation of a route plan. Rate describes the update function rate. Source Lines Of Code (SLOC) include comments and blank lines but not library functions. c refer to C++ and p to Python.

Layer	Component	Rate	SLOC
/fmSensors	**VectorNav interface** reads and publishes the yaw axis gyro rate from a VectorNav VN-100 IMU.	40 Hz	261c
	GNSS interface reads and publishes the GNSS *GPGGA* information including conversion of the position to Transverse Mercator coordinates.	10 Hz	271c
/fmProcessors	**Differential odometry** estimates the current position and orientation based on information from the wheel encoders and the IMU yaw angle gyro.	50 Hz	527c
	Pose estimator uses pre-processing and an Extended Kalman Filter to estimate the absolute position and orientation based on the differential odometry and GNSS information.	50 Hz	592p
/fmDecisionMakers	**Manual driving/autonomous navigation mission** In the *manual driving* behaviour velocity commands from the keyboard are sent directly to the FroboScout interface. In the *autonomous navigation* behaviour the waypoint navigation component is activated.	20 Hz	199p
/fmExecutors	**Waypoint navigation** navigates the waypoint list by publishing linear and angular velocity commands.	20 Hz	1077p
/fmControllers	**FroboScout interface** converts linear and angular velocity commands to wheel velocities, publishes encoder feedback.	50 Hz	428p

In the experiment a differentially steered FroboScout robot (Table 5) was set to autonomously navigate a static route plan containing 5 waypoints using the architecture in Figure 5. The route length is 47 m. Two auxiliary components were added to monitor the system real time performance:

- A computer load monitor component which averages the CPU and memory load every 0.2 s and publishes the results through ROS.
- A real time performance test component consisting of a Python script and an external Atmel ATmega series microcontroller connected to a serial port. The Python script running at at rate of 100 Hz sends one byte to the serial device at each update. The ATmega firmware continuously counts the number of 100 μs ticks between each received byte and returns the latest count upon receiving a byte. This count is in turn received by the Python script and published through ROS.

In total 14 ROS nodes were launched. Together they published ROS messages under 28 different ROS topics, hereof 2 at 100 Hz, 9 at 50 Hz, 1 at 40 Hz, 4 at 20 Hz, 3 at 10 Hz, 6 at 5 Hz and 3 at a lower rate. During the experiment trials all messages published by ROS were recorded using the rosbag tool. The computers used in the trials were connected to internet during the trials but FroboMind did not initiate external network connections. All computers had a standard Ubuntu installation, no attempts to optimize for performance by shutting down default services were made.

Table 5. A list of robots using FroboMind. Column 3 lists the FroboMind version number followed by the reused components detailed in Table 6.

	Latest Application	**FroboMind**
ASuBot 1	Navigating orchards using lidar.	2013 ASU RDT RNV
Armadillo I 2	Navigating maize fields using lidar.	2012 ARD RDT RNV
Armadillo Scout 3	Crop monitoring.	2012 ARD
Armadillo III 4	Driving and navigation in dunes.	2012 ARD ODO POS WPT
AMS 5 Figure 6a	Navigating orchards using lidar [33].	2012 AMS RDT RNV
Armadillo IV 6	Large scale precision spraying trails [34].	2013 ARD ODO POS WPT PMP
Pichi 7	Detection and mapping of anti-tank mines and large unexploded objects.	2013 PIC ODO POS WPT COV MDI HMP
Robotti 8	Mechanical row weeding [2].	2013 ARD ODO POS WPT RWD
FroboMower 9	Mowing using low cost sensors.	2013 FBT ODO POS WPT
Frobit 10	Education [35] & rapid prototyping.	2013 FBT ODO WPT
SMR 11	Education.	2013 MBW ODO WPT
GrassBots 12	Grass mowing on low lands.	2013 GBT ODO POS WPT COV
FroboScout 13	Highway surveying.	2014 FST ODO POS WPT SUR

Table 6. List of components referred from Table 5. Physical Source Lines Of Code (SLOC) for the Grassbot robot interface is unknown as it is closed source. c refer to C++ and p to Python.

Component	**Description**	**SLOC**
AMS	Interface to the AMS robot	410c
ARD	Interface to the Armadillo robot	845c
ASU	Interface to the ASuBot robot	435c
COV	Area coverage algorithm	170p
FBT	Interface to the Frobit robot	285c
FST	Interface to the FroboScout robot	428p
GBT	Interface to the Grassbot robot	?
HMP	Hazard mapping algorithm	850c
MBW	Interface to Mobotware [22]	367c
MDI	Mine Detection Implement interface	125c
ODO	Differential odometry algorithm	527p
PIC	Interface to the Pichi robot	890c
PMP	Parcel map localization algorithm	330p
POS	Pose estimator algorithm	592p
RDT	Detecting crop rows using LIDAR	266c
RNV	Navigation in row crops	255c
RWD	Row weeding implement interface	90c
SUR	Highway surveying application	304p
WPT	Waypoint navigation algorithm	775p

Figure 6. Autonomous Mechanisation System (AMS) and orchard used for the test. (**a**) AMS; (**b**) Apple three orchard.

(**a**) (**b**)

3.2. Evaluating if FroboMind Significantly Decreases the Resources Required to Perform Field Experiments

Evaluating whether the FroboMind software platform significantly decreases the development time and resources required to perform field experiments is difficult because no reference data seem to exist in the literature. An experiment was therefore conducted with the purpose of trying to quantify the portability of FroboMind to a new robot platform by analysing the associated work. The experiment was conducted in collaboration with the University of Hohenheim (UH) and is presented in [33]. A brief description and the results is included here for completeness.

The experiment was designed so that as many parameters as possible were controllable. The task was defined as field robot navigation through an apple tree orchard using local sensors only which is another typical precision agriculture application [36,37]. The Autonomous Mechanisation System (AMS) robot (Figure 6a) owned by UH was used for the experiment. The AMS has been utilized in several previous precision agriculture research projects [38,39] using Mobotware [22], and both the hardware and low level software is well tested and documented. A lidar and an Inertial Measurement Unit (IMU) were used as navigation sensors. An RTK-GNSS system was included for the purpose of recording reference data.

The experiment was conducted during a 5 day workshop where 4 developers worked full time porting FroboMind to the AMS, implementing the orchard navigation task and documenting the process. Preparations before the workshop included preparing the AMS and related hardware, reading documentation and planning the workshop. The task at the workshop was therefore defined as interfacing FroboMind to the AMS and building an application supporting autonomous navigation along the tree rows of the orchard and perform headland turns at the end.

In the orchard used for the navigation tests the tree rows were approx. 100 m long and interspaced by 4 m (Figure 6b). In the first trial the AMS mission was to navigate autonomously between two rows of trees. At the end of the row the AMS would make a 90 degree left turn, drive straight, and make a 90 degree left turn which would position the robot in an adjacent row. It was decided to repeat this process continuously 12 times corresponding to 6 full rounds driving the same track. In the second trial the AMS mission was to navigate autonomously through the same orchard alternating between left

and right turns. Estimation of position and orientation (pose) relative to the tree rows is handled by FroboMind components from a previous project: A Ransac based algorithm uses the lidar data to detect the tree rows and outputs the angle and offset relative to the rows. The angle is fused with the IMU yaw angle data using an Extended Kalman Filter, and the resulting pose is used to control the AMS steering wheels using a PID controller. When the lidar detects the ends of the tree rows, it switches to a turn state and performs the turn based on the IMU data. When the lidar detects the ajacent row, it switches back to in-row navigation.

3.3. Evaluating Software Reuse Across Existing Projects Using FroboMind

During the past three years FroboMind has been used in various projects in precision agriculture as well as in some projects in similar domains. Examples of tasks are navigation of route plans using GNSS, navigation in row crops and orchards using local sensors, control of passive and active implements and interfacing to autonomous implements. Examples of sensor interfaces implemented in FroboMind are encoders, GNSS, IMU, lidar, 3d lidar, 3d vision row camera, localization using stereo vision, total station and metal detector. Examples of actuator interfaces controlled by FroboMind are relays, brushed and brushless motors, servos, linear actuators, hydraulics propulsion and diesel engines. Together these tasks and interfaces support many of the operations required in precision agriculture and the potential for efficient software reuse is thus high.

The third experiment is based on available data from existing projects and field robots that use FroboMind. The purpose is to evaluate the extent of software reuse by looking at the approximate number of physical Source Lines Of Code (SLOC) shared across these projects. As the projects have been carried out during a period where the FroboMind architecture and core components were in very active development it is difficult to perform an accurate comparison. But it provides an indication of the degree of reusability.

4. Results

4.1. Evaluating the Performance of FroboMind in Robot Systems with Near Real-Time Requirements

Three different trials were conducted, each using one of the computers listed in Table 7. The Graphical User Interface (GUI) *Ubuntu desktop* was running on PC2 and PC3, but on PC1 it was shut down prior to the trial because of high CPU load.

In all trials did the robot complete the waypoint navigation task successfully in about 100 s. Figure 7 shows the CPU and memory load averaged at 0.2 s intervals. Figure 8 shows the scheduling delays experienced by the 100 Hz FroboMind component during each of the 3 trials, Table 8 shows statistical data for the delays. Due to lack of an absolute, accurate time source in the experiment setup the delay measurements have been calibrated for offset and skew errors based on statistical calculations and, the data may therefore slightly inaccurate. The skew calibration constants were verified using external frequency measurement.

Table 7. Specifications for the computers used for the 3 trials in the FroboMind real-time performance experiment.

	PC1	**PC2**	**PC3**
Laptop:	Acer Aspire One ZG5	IBM ThinkPad X61s	Acer Travelmate B113
RAM:	1 Gb	2 Gb	4 Gb
CPU (Intel):	Atom N270	Core 2 Duo L7300	Core i3-2377M
Clock:	1.60 GHz	1.40 GHz	1.50 GHz
Cores/siblings:	1/2	2/2	2/4
Cache:	512 kb	4096 kb	3072 kb

Figure 7. CPU & memory load during autonomous navigation of a route plan for each of the 3 trials. The red graphs shows % of max CPU load. The black graphs show usage % of the total memory.

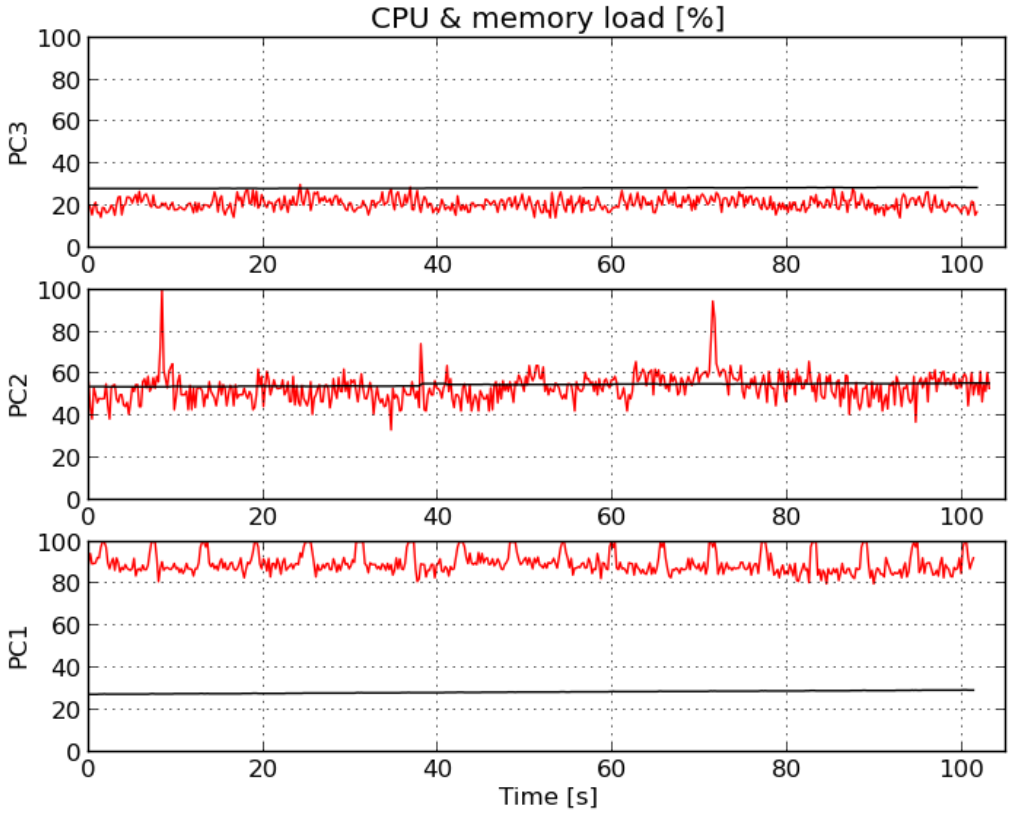


Figure 8. 100 Hz scheduler delays during autonomous navigation of a route plan for each of the 3 trials. The y axis represents the delay of each 10 ms schedule.

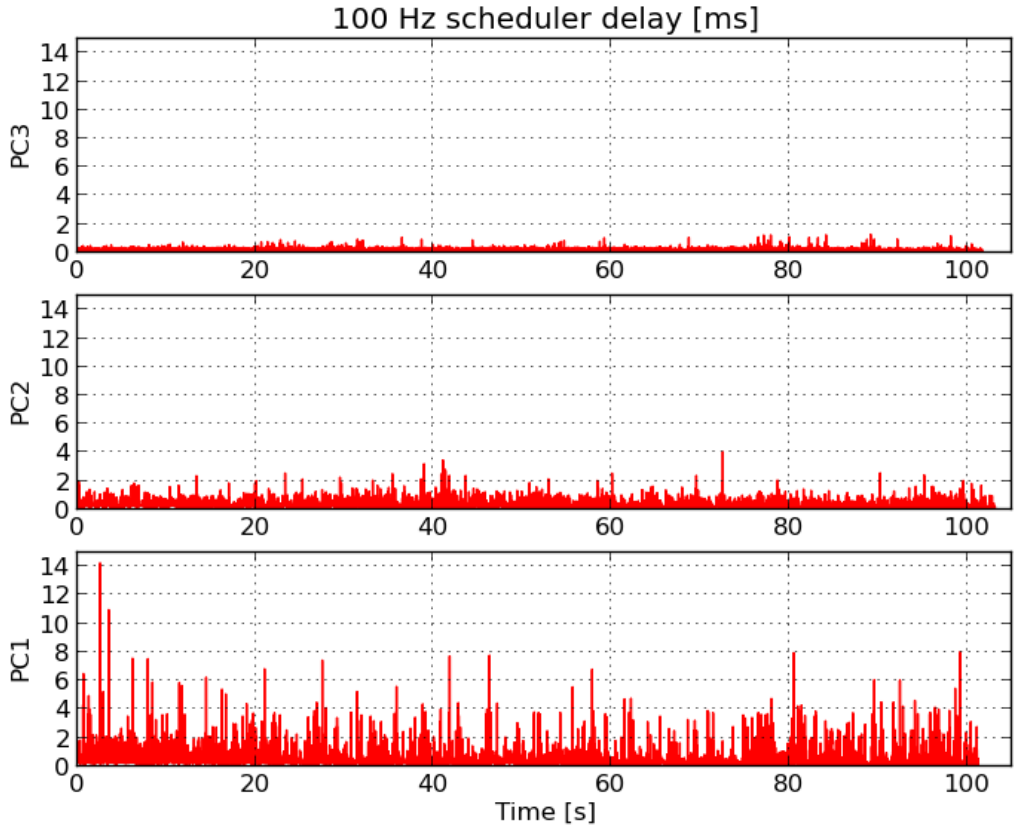


Table 8. Statistics for the schedule delays experienced by the 100 Hz component in the three trials of the FroboMind performance experiment.

	Mean	**Variance**	**95'th Percentile**	**Maximum**
PC1	0.58 ms	0.41 ms^2	1.52 ms	14.1 ms
PC2	0.26 ms	0.055 ms^2	0.72 ms	4.0 ms
PC3	0.24 ms	0.0074 ms^2	0.36 ms	1.3 ms

4.2. Evaluating if FroboMind Significantly Decreases the Resources Required to Perform Field Experiments

During the workshop a journal was continually updated with information about completed tasks *etc*. A summary of the estimated time consumption on different sub tasks is listed in Table 9.

While porting FroboMind to the AMS it was discovered that the AMS exhibited errors when controlling the front wheels. When requesting the AMS to drive straight it would drift slowly to the left. When requesting the AMS to turn left the wheels were positioned correctly but when requesting the AMS to turn right the steering wheels were positioned at an angle significantly lower than the requested angle. Based on the performed tests it is likely that the problem is with the steering hardware, however

due to the time constraints it was decided to mitigate the problems by modifying the relevant FroboMind components. The process to identify, understand and mitigate these errors added substantially to the consumed time.

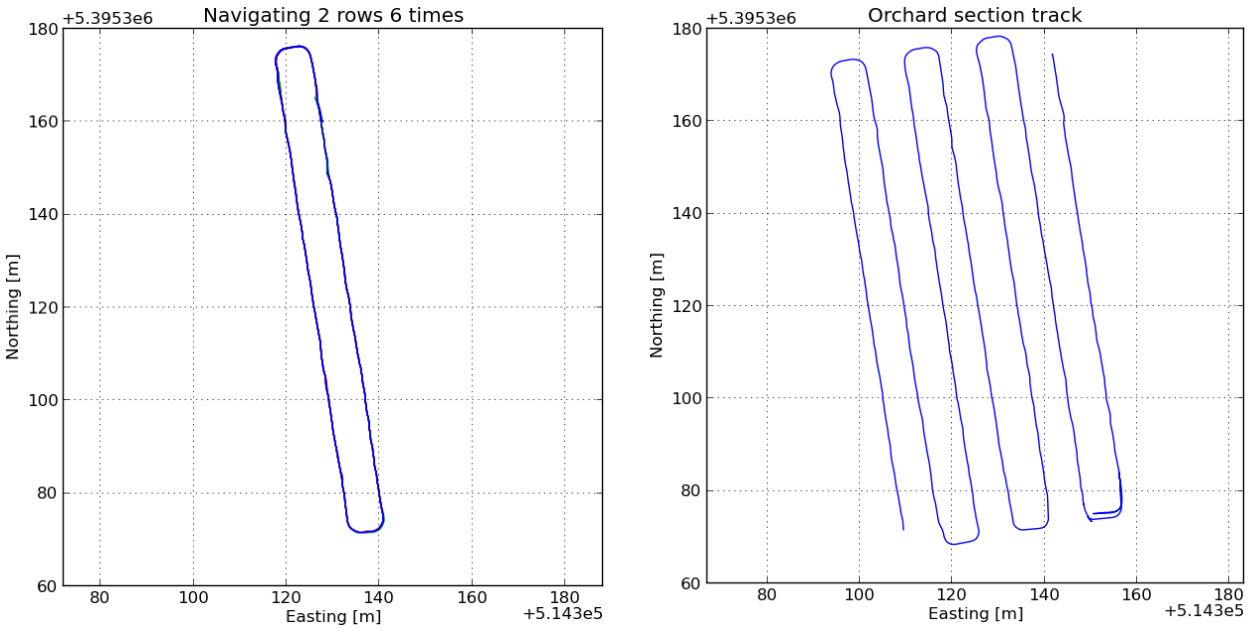
Table 9. Summary of estimated time consumption during the workshop.

Activity	**Duration**
Adding new components to FroboMind that directly support the AMS platform. This includes obtaining information about the AMS interface from documentation and available source code.	60 h
Simulating and testing remote controlled and autonomous driving with the AMS positioned in a test stand.	30 h
Adding and improving the documentation www.frobomind.org when the project work revealed issues not properly documented.	10 h
Updated documentation for the AMS when the project work revealed issues not properly documented.	10 h
Testing the implemented behaviours with the AMS on the ground.	20 h
Collecting data from a manual run in the orchard.	6 h
Analysing data from a manual run in the orchard to prepare autonomous operation.	20 h
Creating a FroboMind mission controller and behaviors for the AMS navigating autonomously in an orchard using local sensors.	40 h
Testing autonomous behaviour in the orchard.	30 h
Updating project documentation and web pages.	12 h

The first trial navigating 6 full rounds driving the same track in two adjacent rows was completed successfully. Three rounds were completed without human intervention. During two of the rounds the navigation failed one time at the same location where the apple trees at one of the rows were replanted by much younger and hence smaller trees. During one round the navigation failed at a location where trees were missing for more than 6 meters at one of the rows. Figure 9a shows the overlay GNSS track from all 6 rounds.

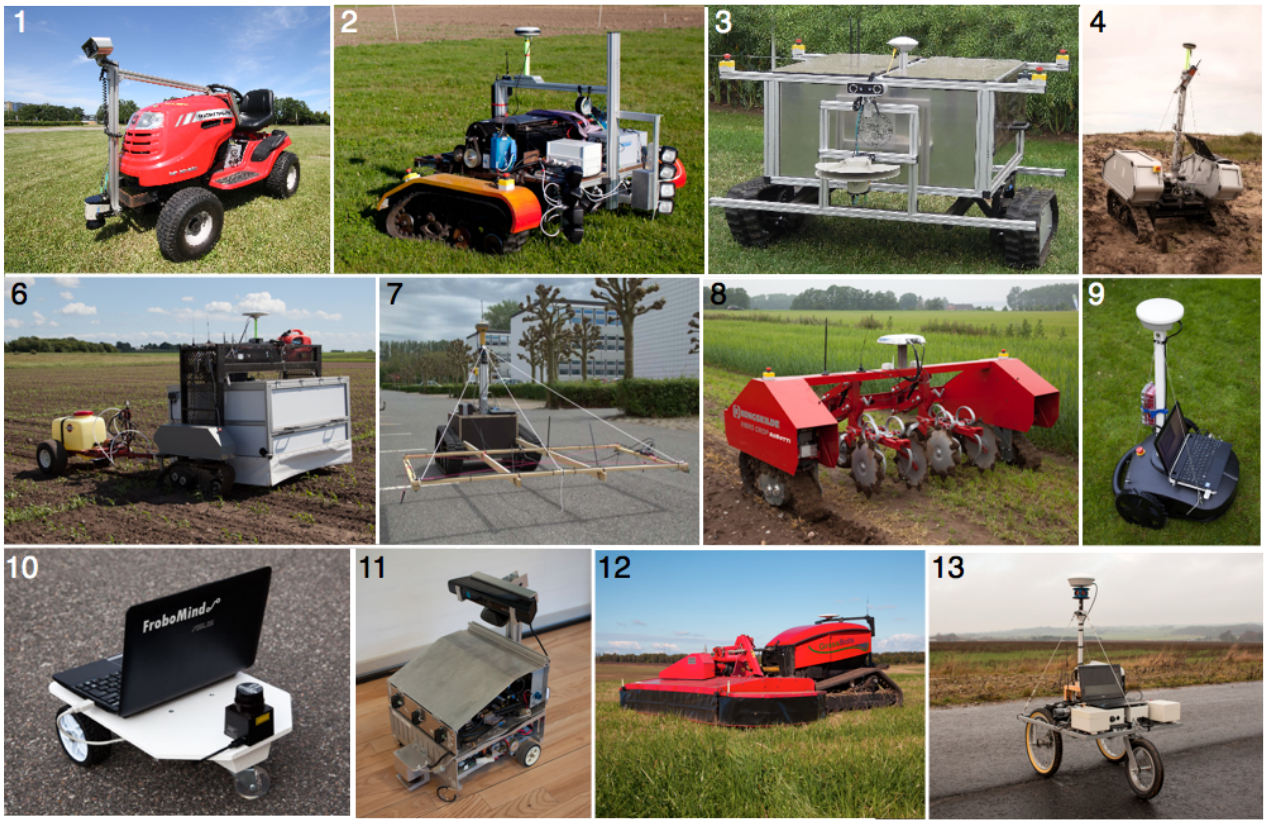
In the second trial navigating a section of rows in the orchard it was decided that focus would not be on the sensor/controller performance under difficult circumstances, so a person stepped in as “tree” whenever one of the rows had a large hole between the trees. The trial was concluded after completing 7 rows. The trial was completed without human intervention except two times where the AMS failed a turn to the right at a row end due to the described steering problems. Figure 9b shows the recorded GNSS track from this trial.

Figure 9. Recorded GNSS track from the trials. (**a**) shows the overlay GNSS track from all 6 rounds. (**a**) Trial 1; (**b**) Trial 2.



(**a**) (**b**)

Figure 10. Images of robots using FroboMind (excluding the AMS depicted in Figure 6a). The numbers show the historical order of FroboMind integration and refer to the list of robots in Table 5.



4.3. Evaluating Software Reuse Across Existing Projects Using FroboMind

Table 5 lists robots that are known to have been interfaced to FroboMind. The table lists the latest known application where FroboMind has been used as well as the latest version of FroboMind known to work on the robot. Based on information obtained from each application, abbreviations for software components shared with other robots are listed next to the version number. The abbreviations refer to Table 6 which lists the component function along with the physical SLOC including comments and blank lines. Due to different component versions the SLOC may differ between different robot installations and should thus be considered to be approximate. Excluded from the list of components are sensor interfaces and libraries *etc*. Figure 10 shows images of each robot, the image number refer to the list of robots in Table 5.

5. Discussion

5.1. Evaluating the Performance of FroboMind in Robot Systems with Near Real-Time Requirements

Ubuntu was chosen as operating system and ROS as middleware because they complied best with the design goals *extensibility*, *scalability*, *software reuse* and *open source*. This was, however, at the cost of hard real-time capabilities that could have been obtained using e.g., RTAI [26] and Orocos [14]. Although hard real-time capabilities are rarely required in field robotics except for low-level control which is typically handled by external embedded systems, it is important to know the system timing uncertainty.

The computing power in the three computers used in the performance evaluation in the first experiment (Table 7) are at the low end of what is available today, however still above the currently popular ARM based embedded boards such as Raspberry Pi and BeagleBone Black.

The trials show a stable memory usage (Figure 7) which is consistent with that none of the active FroboMind components perform any significant dynamic memory allocation. The memory usage was approximately 1/3 Gb for PC1 where the GUI was disabled. For PC2 and PC3 the memory use was approximately 1 Gb.

The CPU load varies in the different trials which is to be expected considering the available computing power. The CPU load for PC1 exhibits a certain periodic pattern of peaks. A subsequent analysis revealed that the peaks were caused by an installed fan control script which sleeps 5 seconds between each execution, so this is not related to FroboMind.

Comparing the CPU load with the scheduler delays shown in Figure 8 and Table 8 it is clear that PC1 is pushed to the limits at this load and has difficulties running a 100 Hz scheduler without overstepping every now and then. PC2 and PC3 manages fine at the current load if the observed delays are acceptable to the application.

In other precision agriculture applications there may be components causing the CPU load to vary significantly with time which may be more or less predictable. Examples are dynamical mapping, route planning, image processing *etc*. In these cases there are different options: (1) Perform a similar experiment to validate that enough computing power is available to the application; (2) Distribute the

components with varying load to another networked platform; (3) Integrate ROS with the Orocos Real Time Toolkit which is documented at the ROS website.

The results from this experiment gives rise to the recommendation that in any application where safety is a concern such as the use of large or heavy machinery, implements with moving mechanical parts, driving at high speed *etc*., the CPU load and the soft real-time performance should be monitored continuously, and detected anomalies should be used as input to the safety system.

5.2. Evaluating if FroboMind Significantly Decreases the Resources Required to Perform Field Experiments

During the 5 day workshop the 4 man development team managed to interface FroboMind to the AMS and build the application for autonomous navigation of an orchard as well as testing the application in two trials. The trials were completed, but navigation was not completely reliable. Upon reviewing the observations at the trials and subsequent analysis of logged ROS messages it was concluded that the reliability of the in-row navigation can be increased by a few modifications to the row detecting algorithm and the navigation controller.

The workshop revealed some issues with FroboMind that need to be addressed. (1) Some of the basic components needs further development with respect to reliability, in particular interfacing to a robot platform through low level interface drivers needs to work seamlessly; (2) Interfaces between layers in the architecture need to be reviewed and properly documented; (3) The usability of FroboMind must be enhanced by a better integration with a simulation environment like Stage. These issues have not yet been solved completely, but a number of improvements have been made since the workshop took place.

As shown in Table 9 the developers worked approximately 12 hours/day each, about half the time was spent on interfacing and the other half on the orchard application. Factors like the described problems with the AMS, transport from the university to the orchard test site and changing weather conditions prolonged the work. Similarly the developer team had collectively an extensive experience in all part of the FroboMind software platform, and the AMS actuators and sensor interfaces were already installed and thoroughly tested using Mobotware, which eased the work. This makes it very difficult to compare with similar experiments, and thus to conclude if FroboMind significantly decreases the resources required to perform field experiments, but experiences from previous projects show that this is a very short time moving from a completely new software implementation on a robot to trials in the field navigating autonomously.

5.3. Evaluating Software Reuse Across Existing Projects Using FroboMind

Table 5 lists a total of 13 robots that have been interfaced to FroboMind during the past three years. With the exception of the AMS and SMR robots that are based on Mobotware they have all been constructed within the same timeframe. The use of FroboMind is not widespread though, the listed robots are spread across only 4 universities and 3 companies, most of them working together developing the field robots. In addition to the list a smaller number of robots developed by researchers, companies and students are known to either use FroboMind or have used FroboMind as a starting point for the software development.

The statistical website ohloh.net reports that the FroboMind directory structure contains 104,617 SLOC (April 2014) and the core part of the ROS platform contains 418,977 SLOC (December 2013). In this context the third column of Table 5 and the associated Table 6 listing SLOC for the components that are shared between some but not all platforms could be considered an indication of the differences in the robot software rather than the similarities.

It seems reasonable to conclude that the level of software reuse between the listed robots is high which may in part be attributed to the modular structure of the FroboMind architecture and to the flexibility that ROS provides. It is, however, also caused by the fact that the robot platforms and their current applications are fairly similar with respect to the robot software.

5.4. Lessons Learned

Choosing Ubuntu as operating system for the software platform has proven to be a great advantage. The limitations caused by not having hard real-time capabilities are counterbalanced by a number of advantages that all relates to simplicity in installation, use and maintenance and hence decreasing the time consumption from development to field trials. The ability to use the same operating system on the robot and the developers laptops speeds up the development process significantly.

The choice of ROS as middleware has had its advantages as well as drawbacks. ROS has a very steep learning curve in the beginning, but after some struggling with understanding the concepts it becomes a valuable tool. The user feedback indicates that FroboMind decreases the learning curve because it provides full working examples for simulation and execution on small robots rather than the user having to build the first test ROS setup from scratch. The ROS messaging system working across networked platforms, the ROS launch tool that ease launching of multiple ROS nodes locally and on networked computers, and the setting of parameters, as well as the many existing libraries and drivers etc. are highly valuable. The ROS tool for recording and playback of messages saves a tremendous amount of time when analyzing data and debugging. But ROS is still in a very active state of development. Experience shows that at new version releases, not all core components are fully working with the new version, and some of the new versions include substantial changes that induce unforeseen maintenance tasks. Examples are the introduction of the catkin build system and removal of the Stack concept in the ROS Groovy version. Seen in retrospect ROS was a better choice than Orocos when considering the purpose of FroboMind and the stated design goals. But it comes with the price of lacking hard real-time capabilities which is not a major problem but needs to be addressed in most applications, and the ROS code base appears to be less mature and stable than Orocos.

The FroboMind architecture design builds upon the results of the reviewed publications, especially Stanley [25], CARMEN [8] and [17] provided a significant input to the design. The architecture has undergone several revisions since the first prototype was developed in 2011. The original draft was very detailed but has been simplified significantly because experiences from using FroboMind in research projects have revealed a relationship between complexity and the willingness to accept and utilize the architecture. If the architecture is too complex, users tends to short circuit it by merging all the remaining functionality into a single or a couple of components, which makes them less suitable for reuse. It is debatable if the architecture is still too complex, this should be investigated in the future work.

An important part of the architecture is the specification of data exchanged between the components. For navigation sensors, localization and low level propulsion control and actuation this is reasonably well defined and where applicable aligned with current ROS standards. Interfacing the decision making layer and implement modules still need to be properly specified. This work should where applicable be based on existing standards within agriculture and related field work such as OpenGIS, agroXML, GPX, Isobus *etc*.

Compared to the architectures listed in Table 2 FroboMind obtains a *Yes* in all parameters which indicates that the aim of this work has been achieved. An overall performance comparison of the FroboMind software platform to solutions described in the related work is impossible though, as data relevant for comparison of the architectural and software performance has not been published.

5.5. The future of FroboMind

The primary focus in the development of FroboMind has until now been to establish a common open software platform that promotes software reuse and thus decreases the development time and resources required to perform field experiments. This objective has to a large extent been achieved and the work will continue though the application of FroboMind to new research projects, development of new components as well as improving the existing components in terms of reliability.

In addition to this a project focusing on safe behaviour of agricultural machines has been launched recently. FroboMind will be used in parts of the project, and it is expected that this will contribute significantly to the development of: (1) The safety layer which is currently limited to a deadman signal which enables the actuator controllers; (2) The decision making architecture, which currently is limited to fairly simple behaviours defined in finite state machines; (3) source code integrity through the implementation of model driven auto-generation of component interfaces and structures.

6. Conclusions

In this paper we have presented FroboMind, a software platform tailored to field robots in precision agriculture. FroboMind optimizes field robot software development in terms of code reuse between different research projects. At the current stage FroboMind has been ported to 2 different 4-wheeled tractors, 7 different configurations of tracked field robots and 4 differentially steered robots. Examples of current FroboMind applications are autonomous crop scouting, precision spraying, mechanical weeding, grass cutting, humanitarian demining and land surveying.

In an experiment evaluating the performance of FroboMind in robot systems with near real-time requirements it was concluded that FroboMind's soft real-time execution is sufficient for typical precision agriculture tasks such as autonomous navigation of a predefined route. In any application where safety is a concern, the CPU load and the soft real-time performance should be monitored continuously, and detected anomalies should be used as input to the safety system.

To test whether FroboMind decreases the development time and resources required to perform precision agriculture field experiments, an experiment was performed porting FroboMind to a new field robot and applying this robot to autonomous navigation in an orchard using local sensors. This was

achieved by 4 developers during a 5 day workshop, which is a very short time moving from a completely new software implementation on a robot to trials in the field navigating autonomously.

The software reuse across projects has been assessed using available data from existing projects and robot platforms. It was concluded that the level of software reuse between the listed robots is high which may in part be attributed to the modular structure of the FroboMind architecture, the flexibility that ROS provides and the fact that the robot platforms and their current applications are fairly similar with respect to the robot software.

It is concluded that FroboMind is usable in practical field robot applications by research groups and other stakeholders. Further development of FroboMind continue through new research projects where the current activities focus on the decision making structure and autonomous safe behaviour. The FroboMind software has been released as open-source at the FroboMind website [32] for others to build upon.

Acknowledgements

This work is linked to and partially funded by the ERA-Net ICT-AGRI project: Grassbots, the Danish Ministry for Food, Agriculture and Fisheries project: FruitGrowth, and the Danish National Advanced Technology Foundation project: The Intelligent Sprayer Boom. The authors wish to thank Claes D. Jaeger, Hans W. Griepentrog, Ulrik P. Schultz and Mikkel K. Larsen for supporting this work.

Authors Contributions

Kjeld Jensen: Main author, main architect of FroboMind, developed FroboMind components, designed the experiments and used FroboMind in research projects; Morten Larsen, Søren H. Nielsen, Leon B. Larsen and Kent S. Olsen: Contributed to the FroboMind design and experiments, developed FroboMind components, used FroboMind in research projects and gave important feedback on drafts; Rasmus N. Jørgensen: Used FroboMind in research projects and gave important feedback on drafts.

Conflict of Interest

The authors declare no conflict of interest.

References

1. Pedersen, S.M.; Fountas, S.; Have, H.; Blackmore, B.S. Agricultural robots—System analysis and economic feasibility. *Precis. Agric.* **2006**, *7*, 295–308.
2. Green, O.; Schmidt, T.; Pietrzkowski, R.P.; Jensen, K.; Larsen, M.; Jørgensen, R.N. *Commercial Autonomous Agricultural Platform*; Kongskilde Robotti: Soroe, Denmark, 2014.
3. Bakker, T.; van Asselt, K.; Bontsema, J.; Muller, J.; van Straten, G. Systematic design of an autonomous platform for robotic weeding. *J. Terramechanics* **2010**, *47*, 63–73.
4. Ruckelshausen, A.; Biber, P.; Dorna, M.; Gremmes, H.; Klose, R.; Linz, A.; Rahe, R.; Resch, R.; Thiel, M.; Trautz, D.; Weiss, U. BoniRob: An Autonomous Field Robot Platform for Individual Plant Phenotyping. Avaliable online: https://my.hs-osnabrueck.de/ecs/fileadmin/groups/156/Veroeffentlichungen/2009-JIAC-BoniRob.pdf (accessed on 12 May 2014).

5. Jørgensen, R.; Sørensen, C.; Pedersen, J.; Havn, I.; Jensen, K.and Søgaard, H.; L.B., S. Hortibot: A system design of a robotic tool carrier for high-tech plant nursing. *CIGR E J.* **2007**, *9*, ATOE 07 006.
6. Blackmore, B.S.; Griepentrog, H.W.; Nielsen, H.; Nørremark, M.; Resting-Jeppersen, J. Development of a deterministic autonomous tractor. In Proceedings of the 2004 CIGR Olympics of Agricultural Engineering, Beijing, China, 11–14 October 2004.
7. Neisser, U. *Cognitive Psychology*; Meredith Publishing: New York, NY, USA, 1967.
8. Montemerlo, M.; Roy, N.; Thrun, S. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA, 27–30 October 2003.
9. Nesnas, I.A.D.; Wright, A.; Bajracharya, M.; Simmons, R.; Estlin, T. CLARAty and challenges of developing interoperable robotic software. InProceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA, 27–30 October 2003; Volumes 1–4, pp. 2428–2435.
10. Pivtoraiko, M.; Nesnas, I.A.; Nayar, H.D. A Reusable Software Framework for Rover Motion Controls. Available online http://robotics.estec.esa.int/i-SAIRAS/isairas2008/Proceedings/SESSION%2011/m086-Pivtoraiko.pdf (accessed on 12 May 2014).
11. Cepeda, J.; Chaimowicz, L.; Soto, R. Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics. In Proceedings of the Robotics Symposium and Intelligent Robotic Meeting (LARS), Latin American, Sao Bernardo do Campo, Brazil, 23–28 October 2010; pp. 7–12.
12. Makarenko, A.; Brooks, A.; Kaupp, T. Orca: Components for Robotics. In Proceedings of the Workshop on Robotic Standardization, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006), Beijing, China, 9–15 October 2006.
13. Henning, M. A New Approach to Object-Oriented Middleware. *IEEE Internet Comput.* **2004**, *8*, 66–75.
14. Bruyninckx, H. Open robot control software: The OROCOS project. In Proceedings of the IEEE International Conference on Robotics and Automation, Seoul, Korea, 21–26 May 2001.
15. Gerkey, B.P.; Vaughan, R.T.; Sukhatme, G.S.; Stoy, K.; Mataric, M.J.; Howard, A. Most valuable player: A robot device server for distributed control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, HI, USA, 29 October–3 November 2001.
16. Quigley, M.; Conley, K.; Gerkey, B.P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009.
17. Vázquez-martín, R.; Martinez, J.; Toro, J.C.; Núñez, P. A Software Control Architecture based on Active Perception for Mobile Robotics 1. Available online: http://www.researchgate.net/publication/228638002ASoftwareControlArchitecturebasedonActive PerceptionforMobileRobotics/file/5046351f943eef19bb.pdf (accessed on 12 May 2014).
18. Brooks, R.A. A Robust Layered Control-system For A Mobile Robot. *IEEE J. Robot. Autom.* **1986**, *2*, 14–23.

19. Patricio Nebot, J.T.S.; Martinez, R.J. A New HLA-Based Distributed Control Architecture for Agricultural Teams of Robots in Hybrid Applications with Real and Simulated Devices or Environments. *Sensors* **2011**, *11*, 4385–4400.
20. Garcia-Perez, L.; Garcia-Alegre, M.C.; Ribeiro, A.; Guinea, D. An agent of behaviour architecture for unmanned control of a farming vehicle. *Comput. Electron. Agric.* **2008**, *60*, 39–48.
21. Kuhnert, K.D. Software Architecture of the Autonomous Mobile Outdoor Robot AMOR. In Proceedings of the 2008 IEEE Intelligent Vehicles Symposium, Eindhoven, The Netherlands, 4–6 June 2008.
22. Beck, A.B.; Andersen, N.A.; Andersen, J.C.; Ravn, O. MobotWare—A plug-in based framework for mobile robots. In Proceedings of the 7th IFAC Symposium on Intelligent Autonomous Vehicles, Lecce, Italy, 6–8 September 2010; Volume 7, pp. 127–132.
23. Simon Blackmore, S.F.; Have, H. Proposed System Architecture to Enable Behavioral Control of an Autonomous Tractor. Automation Technology for Off-Road Equipment. In Proceedings of the American Society of Agricultural and Biological Engineers (ASABE), Chicago, IL, USA, 26–27 July 2002. pp. 13–23.
24. Fountas, S.; Blackmore, B.S.; Vougioukas, S.; Tang, L.; Sørensen, C.G.; Jørgensen, R. Decomposition of Agricultural tasks into Robotic Behaviours. *Agric. Eng. Int. CIGR Ejournal* **2007**, *IX*, Manuscript PM 07 006.
25. Thrun, S.; Montemerlo, M.; Dahlkamp, H.; Stavens, D.; Aron, A.; Diebel, J.; Fong, P.; Gale, J.; Halpenny, M.; Hoffmann, G.; *et al*. Stanley: The robot that won the DARPA Grand Challenge. *J. Field Robot.* **2006**, *23*, 661–692.
26. Dozio, L.; Mantegazza, P. Linux Real Time Application Interface (RTAI) in low cost high performance motion control. In Proceedings of the Motion Control 2003, a Conference of ANIPLA, Associazione Nazionale Italiana per l'Automazione (National Italian Association for Automation), Milano, Italy, 27–28 March 2003.
27. Carsten Emde, T.G. *Quality Assessment of Real-Time Linux*; Technical Report, Open Source Automation Development Lab: Stuttgart, Germany, 2011.
28. McKenney, P.E. *'Real Time' vs. 'Real Fast': How to Choose*? Technical Report, IBM Linux Technology Center: New York, NY, USA, 2008.
29. Leonard, J.; Durrant-Whyte, H. Mobile robot localization by tracking geometric beacons. *IEEE Trans. Robot. Autom.* **1991**, *7*, 376–382.
30. Stuart Russell, P.N. *Artificial Intelligence: A Modern Approach*, 3rd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2010.
31. Schacter, D. *Psychology*; Worth Publishers: London, UK, 2011.
32. FroboMind Website. Available online: http://www.frobomind.org accessed on (5 May 2014).
33. Jaeger-Hansen, C.; Jensen, K.; Larsen, M.; Nielsen, S.; Griepentrog, H.; Jørgensen, R. Evaluating the portability of the FroboMind architecture to new field robot platforms. In Proceedings of the 9th Europaean Confernce on Precision Agriculture, Lleida, Spain, 7–11 July 2013.
34. Jensen, K.; Laursen, M.S.; Midtiby, H.; Jørgensen, R.N. Autonomous Precision Spraying Trials Using a Novel Cell Spray Implement Mounted on an Armadillo Tool Carrier. In Proceedings of the XXXV CIOSTA & CIGR V Conference, Billund, Denmark, 3–5 July 2013.

35. Larsen, L.B.; Olsen, K.S.; Ahrenkiel, L.; Jensen, K. Extracurricular Activities Targeted towards Increasing the Number of Engineers Working in the Field of Precision Agriculture. In Proceedings of the XXXV CIOSTA & CIGR V Conference, Billund, Denmark, 3–5 July 2013.
36. Barawid, O.C.; Mizushima, A.; Ishii, K.; Noguchi, N. Development of an autonomous navigation system using a two-dimensional laser scanner in an orchard application. *Biosyst. Eng.* **2007**, *96*, 139–149.
37. Blas, R. Fault-Tolerant Vision for Vehicle Guidance in Agriculture. Ph.D. Thesis, Technical University of Denmark, Lyngby, Denmark, 2010.
38. Reske-Nielsen, A.; Mejnertsen, A.; Andersen, N.A.; Ravn, O.; Nørremark, M.; Griepentrog, H.W. Multilayer Controller for Outdoor Vehicle. In Proceedings of the Automation Technology for Off-Road Equipment (ATOE), Bonn, Germany, 1–2 September 2006; pp. 41–49.
39. Griepentrog, H.; Andersen, N.; Andersen, J.; Blanke, M.; Heinemann, O.; Madsen, T.; Nielsen, J.; Pedersen, S.; Ravn, O.; Wulfsohn, D. Safe and Reliable: Further Development of a Field Robot. Available online: www.staff.kvl.dk/ hwg/pdf/papers/Griepentrog2009-7ECPA.pdf (accessed on 12 Mary 2014).