

Linux Robotics Framework Software Requirements Specification

The LRF Team

June 28, 2008

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	References	4
1.4	Overview	4
2	Glossary: Definitions, Acronyms and Abbreviations	5
3	Overall description	7
3.1	Product perspective	7
3.2	System interfaces	7
3.2.1	User interfaces	8
3.2.2	Hardware interfaces	8
3.2.3	Software interfaces	8
3.3	Communication interfaces	8
3.4	Memory	8
3.5	Operations	9
3.6	Site adaptation requirements	9
3.7	Product functions	9
3.8	User characteristics	9
3.9	Constraints	9
4	Specific requirements	10
4.1	Research and develop reusability standards	10
4.1.1	10
4.2	Documentation standards	10
4.2.1	10
4.2.2	10
4.2.3	10
4.3	Comprehensive interfaces	10
4.3.1	10
4.4	Library organisation	11
4.4.1	11
4.4.2	11
4.4.3	11
4.4.4	11
4.5	Cross-architectural portability	11
4.5.1	11
4.5.2	11
4.5.3	11
4.5.4	11
4.5.5	11
4.6	Linkage against common Linux libraries	11
4.6.1	11
4.7	Acceptable performance	12
4.7.1	12
4.8	Explicit memory footprint	12
4.8.1	12
4.9	Runnable on Linux	12

1 Introduction

1.1 Purpose

This document provides the complete requirements of the Linux Robotics Framework (LRF). All the requirements found here will be specific and verifiable. This document is intended for all stakeholders and the developers of the LRF project.

1.2 Scope

The Linux Robotics Framework will provide a software framework to simplify the process of developing robotics applications under a Linux based operating system.

The framework will be used to develop robotic systems of varying levels of professionalism. It will streamline the development process for hobbyists and commercial developers alike. The LRF will allow for the development of robotics applications on small low-cost system architectures with specific focus on support for the avr32 microprocessor.

It will do this by providing a collection of component modules comprising a reusable and extensible robotics framework. The framework will include interface definitions and the implementation of specific drivers and libraries. The framework will be extensible, providing a mechanism for adding new hardware and software drivers.

Given the massive (infinite) range of devices and operations available to a robotic system, we will limit our effort to only those required by the two demonstration systems identified. The hardware classes to be supported include GPS devices, accelerometers and motor control units (see Requirements - Components List).

1.3 References

Nias Digital Project Proposal
Linux Robotics Framework Operational Concept Description
Linux Robotics Framework Project Brief.

1.4 Overview

Section 1 of this document contains the documents purpose and its intended scope. It also contains a list of references to related documents and this this overview .

Section 2 contains a glossary of the definitions for domain specific terms.

Section 3 describes the product to give readers an idea of what it is and what it is for. This section should not be considered to contain any requirements of the products.

Section 4 contains all the requirements for the LRF. It will be broken up into functional categories, each containing SMART shall statements.

Section 5 contains the appendices.

Section 6 contains the index.

2 Glossary: Definitions, Acronyms and Abbreviations

Accelerometer

A hardware device used to detect acceleration and tilt.

API

Application Programming Interface. A library or framework whose modules can be integrated into software package to provide services via function calls.

AVR32

This refers to a member of the Atmel Corporation's AVR32 family of motherboards.

Common Robotics tasks

These describe typical utility tasks executed by a robot in the course of pursuing higher goals. These may range from basic tasks such as unimpeded movement from point A to point B, to advanced tasks such as route optimisation and obstacle avoidance.

Common Linux libraries

This describes the GNU standard C library (glibc) and the "micro" version of the standard c library (uclibc).

Component

A unit of software controlling a robotic hardware and/or other components.

Core library

This refers to the components of the LRF encompassed by this document, ie listed under "Specific requirements - Component list" and therefore individually specified as per the SRS listed in Appendix A.

Demo application

An demo application is an application created using the LRF whose purpose it to demonstrate some subset of its capabilities.

Distribution

A distribution is a complete version controlled set (ie edition) of a framework, library or software package.

Driver

A unit of utility software which controls, ie "drives" the operation of a unit of hardware.

GPS Module

A Global Positioning System hardware device.

Linux

Linux is the name of the free open source operating system family based on the Unix kernel and toolbox approach.

LRF

This refers to the complete package of source code libraries, documentation, tutorials etc that comprise the Linux Robotics Framework.

Interface

An interface is a software interface that allows applications to read or setup necessary hardware configurations . It aims to conceal differences in hardware between different models from the software.

Environment

An environment is a

Functionality

A measure of the degree to which an application can provide the required output.

Gyroscope

A hardware device for measuring or maintaining orientation, based on the principles of angular momentum.

Interchangibility

The interchangibility of a component is the extent to which it can be exchanged with another component which implements the same configuration interface (as defined in section 4.5).

Kinematics library

A library for processing perceptual inputs.

Library

A library is a collection of subroutines used to develop software. Libraries contain code and data that provide services to independent programs. This allows code and data to be shared and changed in a modular fashion.

Module

A module is a collection (sub library) of components used for a single purpose.

Portability

The portability of a software artifact is defined as the degree to which it can compile/run and runs with acceptable performance.

Reusability

Software reusability is a measure of the ease with which it can be used again to add new functionalities with minimal or no modification. In order to achieve reusability issues such as building, packaging, distribution, installation, configuration, deployment, maintenance and upgrade issues need to be addressed. Specific design features which improve reusability include:

- Modularity
- Genericity of use
- Parameterization
- Simplicity of understanding
- Localisation of volatile (changeable) design assumptions
- Stability under changing requirements

- Consistency
- Correctness
- Orthogonality

Robotic Application

A robotic application is a software application (typically designed to operate using a specific robotic architecture) created by the end user by combining their own code with modules of the LRF.

Robotic Architecture

A robotic architecture is a specific combination and configuration of robotic devices.

Robotic Devices

A robotic device is a hardware device which is used as a component of a robotic architecture

Servo

A hardware device comprising a motor (typically electric) that utilises some form of velocity and/or position feedback to effect mechanical motion for a specified distance.

Servo Controller

A hardware device capable of managing the operation of multiple servos.

Software Framework

See API.

Stakeholders

The parties which stand to benefit from the successful delivery of the LRF.

System Architecture

System architecture refers to the specific CPU and motherboard (computational hardware) family

Word Size

Word size refers to the byte size of the CPU.

3 Overall description

3.1 Product perspective

3.2 System interfaces

The Linux Robotics Framework will form a layer within a robotic application. It will act as an interface between the user supplied intelligence and the hardware supplied as 'the robot'. Above the LRF will be the user created application. This will interact with the framework by linking against the LRF libraries at compile time. The interface will differ among different classes of component. These should be well documented.

Below the LRF is the operating system and hardware components. Components of the LRF that represent hardware will interact with the specific hardware component via the OS. This

can involve calls for opening sockets and other file descriptors. Eventually the hardware layer will be reached and protocols for each device will be required.

3.2.1 User interfaces

There are two types of user interaction involved with the LRF. The first is the developer working on a robotic application and using the LRF in their implementation. The second is the user interacting with a complete robot which has the robotic application installed. For the developer, the interface is made up of the source code and documentation.

The user interface documentation will supply the user with all the information required to create a robotic application that makes use of the LRF, while the source code can be included in the users own build path. The user interface on the completed robot is really up to the developer. Some robots may not have and end user interface at all. However when they do, they could use any medium they like. The easiest way would be to implement the interface as a component that behaves as a sensor and actuator to the robot. This allows a developer to create their user interface over as a networked web interface or a hardware input device.

3.2.2 Hardware interfaces

The hardware interfaces are a very large portion of the LRF. This makes it very difficult to include every possible interface that may be required. For this reason, we will only attempt to specify the interfaces considered common. Some of these will be serial ports, usb devices and ethernet controllers. This list can be extended to any kind of hardware, like blue-tooth, sound cards, video cards, etc...

3.2.3 Software interfaces

The main software interface that needs to be specified is that with the operating system. The operating system used will be any common Linux distribution. The LRF will use the operating system to access all the hardware components as well as manage the execution of the application.

Some simulation tools have also been identified and would be very useful to be able to interact with. As usual this interface would be implemented as a component in the robotic application.

3.3 Communication interfaces

The LRF will be capable of communicating with pretty much anything. In fact many hardware components will technically be implemented via some communication link. This makes for a very small step to implementing communication channels with other robots or users.

3.4 Memory

The LRF will be required to run on embedded processors. A simple robotic application implemented with the LRF must be able to run with 32MB of primary memory and no swap space. This limitation is really up to the developer of the robot to meet, but our framework must not be limiting in this area.

3.5 Operations

The LRF will have no specific modes of operation, as this is a concern of the developer creating a robotic application. A system implemented with the LRF may operate for long periods of time, often unattended.

3.6 Site adaptation requirements

For each installation of the LRF, a developer will need to setup a configuration of framework components. This would represent the hardware configuration of the robotic application they were installing on. They would also need to provide the logic code that will define the robots behavior.

Some components included in the LRF will be locale specific, like a GPS device. As a complete list of components in the LRF is unknown, it is up to the developers of components to document locale/site specific configuration.

3.7 Product functions

The LRF is primarily a layer of abstraction between the hardware of a robotic application, and the users logic code. It will contain the drivers that interact with all the hardware components and some library functions for common tasks.

The LRF's main goal is to promote reusable code among different robotic projects. By agreeing with certain standards and having well thought out interfaces, hopefully new developers will add to the framework and create new components.

Portability is also an important function of the LRF. Being able to run on multiple system architectures and different Linux distributions will open the tool to many different projects and organisations.

3.8 User characteristics

The main users are expected to be technically capable and have a good understanding of the robotic applications they are trying to develop. They should be able to study documentation and be familiar with how to understand and use a third party API. They will not need to understand the lower level concerns abstracted by certain device drivers.

Users that wish to develop the LRF further will need to have a very good understanding of the robotic application they wish to use, as well as a good understanding of the design of the LRF and how components are to behave.

3.9 Constraints

The LRF is being developed under a number of constraints. These constraints are not always present around a particular instance of the framework, but need to be considered to make it portable and reusable.

The most pressing constraint is that of the hardware. The LRF is intended to run on small cheap processors that do not have much processing power. It is also intended to run on any common architecture that supports the Linux kernel.

It will need to interface easily with the operating system and the hardware beneath it. Other software interfaces should be considered but are not required, as many components can be created to interact with other software systems.

Considerations should be made regarding the safety of some operations of the framework, as making a reliable control system for automated devices, makes it much more attractive to the market. Security should also be considered for similar reasons.

The LRF as detailed in this document must be completed by October 2008

4 Specific requirements

4.1 Research and develop reusability standards

Consistency and appropriateness of coding and documentation are key factors in improving reusability, therefore:

4.1.1

The LRF shall include a report on contemporary standards of reusability and the projects incorporation of those standards.

4.2 Documentation standards

The quality and quantity of documentation is a key factor in improving reusability, therefore:

4.2.1

There shall be two categories of documentation, namely:

- developer (internal comments inline with source code and external (instructions and advice extraneous to, but associated with, a particular module)
- user (library compilation, assembly and usage tutorials, general manuals and documentation).

4.2.2

The project shall utilise documentation conventions which conform with the accepted Linux coding standards and documentation doxygen convention

4.2.3

Each component shall contain documentation parsable by doxygen for each function, parameter and dependency.

4.3 Comprehensive interfaces

Each component will have well defined interface specs

4.3.1

Each component's interface shall be documented externally.

4.4 Library organisation

Ease of use is a key factor in improving reusability, therefore:

4.4.1

The LRF library shall be organised into components.

4.4.2

Each component shall have a clearly defined intended functionality.

4.4.3

The documentation for each component shall have that component's dependencies clearly identified.

4.4.4

Code documentation for each method shall include details of the functionality, pre-conditions and post-conditions.

4.5 Cross-architectural portability

The ability to deploy non-hardware specific implementation is a key factor in improving reusability, therefore:

4.5.1

The components of the core library shall must run on the target processors (the AVR32 and x86)

4.5.2

The library shall provide identical functionality on the target processors.

4.5.3

All processor-dependent code shall confined to one module for each architecture.

4.5.4

The LRF components must be written in C.

4.5.5

All LRF source code produced shall compile and run successfully using gcc 4.4.3 (check with client)

4.6 Linkage against common Linux libraries

4.6.1

The LRF shall only link against the common Linux libraries

4.7 Acceptable performance

The LRF must be able to meet client standards of performance, therefore:

4.7.1

Specific individual components and grouped modules shall deliver specific performances as agreed between the team and the client in the Performance Deliverables Schedule (schedule subject to variation).

4.8 Explicit memory footprint

The memory footprint of LRF components must be identifiable to users, therefore:

4.8.1

Module memory footprints shall be reported in the Component Data and Dependencies Schedule.

4.9 Runnable on Linux

The ability to run on Linux systems is essential, therefore:

4.9.1

The LRF shall be loaded and executed by the Linux kernel

4.9.2

Kernel specific functionality shall be Linux compatible

4.10 Component list

The LRF components that will comprise the 'core library' of the LRF framework will comprise one component for either a specific (model and type) of each of the following hardware devices, or for the indicated purpose:

4.10.1

Accelerometer.

4.10.2

GPS device.

4.10.3

Gyroscope device.

4.10.4

Kinematics functionality.

4.10.5

Motor controller.

4.10.6

Servo device.

4.10.7

Servo controller device.

4.11 Supply of example applications

The LRF must be demonstrably useful in constructing robotic applications, therefore:

4.11.1

The LRF shall provide demo applications

4.11.2

Demo applications shall use the supplied libraries, interfaces and drivers to produce functionality.

4.11.3

Demo applications shall be documented.

4.12 Supply of demo application documentation

The LRF must include examples of usage, therefore:

4.12.1

Detailed documentation for the demo applications shall be provided in the LRF

4.13 Supply of example drivers

The LRF must include aids to learning the use of basic components, therefore:

4.13.1

The LRF shall include drivers for robotic devices used by the demo applications

4.13.2

Robotic devices shall only communicate to a demo application using supplied drivers.

4.14 Supply of example libraries

The LRF must be demonstrably useful in constructing robotic applications, therefore:

4.14.1

The LRF shall provide libraries.

4.14.2

LRF libraries shall provide functions to achieve common robotics tasks. Libraries included:

- Move according to directions
- Manipulate arm according to directions

4.15 Interfaces for classes of robotic hardware

The LRF must be portable across different robotic architectures

4.15.1

A collection of interfaces shall be provided.

4.15.2

Each component shall implement an interface.

4.15.3

The LRF shall define a common interface for each class of component

4.15.4

Each class of components shall implement the class interface.

5 Appendixes

Appendix A will comprise of a list of the component level SRS documents, which will be added as they are drafted. Appendix B will comprise the Performance Deliverables Schedule (Specific Requirements - Performance). Appendix C will comprise the Component Data and Dependencies Schedule (Specific Requirements - Memory footprint).

6 Index

For a specific term, consult section Definitions, acronyms, and abbreviations. For section headings check the table of contents.