

# Linux Robotics Framework Software Architecture Specification

The LRF Team

July 8, 2008

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Background . . . . .	3
1.2	Purpose and Scope . . . . .	3
1.3	Audience . . . . .	3
1.4	References . . . . .	3
<b>2</b>	<b>Design Constraints</b>	<b>3</b>
2.1	Architecture principles . . . . .	3
2.1.1	Component based, object oriented approach . . . . .	3
2.1.2	Robotic Hardware Abstraction . . . . .	3
2.1.3	System Hardware abstraction . . . . .	3
2.2	Design Assumptions / Deficiencies . . . . .	3
2.2.1	Adequate similarity among a class of hardware . . . . .	3
<b>3</b>	<b>High Level Architecture</b>	<b>4</b>
3.1	Core . . . . .	4
3.2	Device packages . . . . .	4
3.2.1	Actuators . . . . .	4
3.2.2	Sensors . . . . .	4
3.3	Logic packages . . . . .	4
3.3.1	Motion . . . . .	5
3.3.2	Position . . . . .	5
3.3.3	Kinematics . . . . .	5
3.4	Testing Component . . . . .	5
<b>4</b>	<b>Interface Descriptions</b>	<b>5</b>
4.1	Actuators . . . . .	5
4.1.1	Servo Controller . . . . .	6
4.2	Sensors . . . . .	6
4.3	Motion . . . . .	6
4.4	Position . . . . .	6
<b>5</b>	<b>Use Case Realisation</b>	<b>6</b>
5.1	Car . . . . .	6
5.2	Arm . . . . .	7
<b>6</b>	<b>Architecturally significant aspects</b>	<b>9</b>
6.1	Deployment platforms . . . . .	9
6.2	Coding Conventions . . . . .	9
6.3	Packaging . . . . .	10

# 1 Introduction

## 1.1 Background

## 1.2 Purpose and Scope

This document describes the architecture for the Linux Robotics Framework (LRF). The system layout and component relationships are defined here, however individual components of the architecture are not specified in detail. Individual components will be specified in separate documents during later phases of the project.

## 1.3 Audience

This document is intended for those who wish to undertake development using the Linux Robotics Framework, or for those who are continuing development using the framework.

## 1.4 References

# 2 Design Constraints

## 2.1 Architecture principles

### 2.1.1 Component based, object oriented approach

The system is divided into separate components, allowing a user of the framework to choose a subset of the framework applicable to their robot.

### 2.1.2 Robotic Hardware Abstraction

Each class of hardware used by the framework has abstract interfaces to decouple users from any specific device. This allows the user of the framework to swap out one implementation of hardware (eg. pololu servo controller) for another with minimal change to their own code.

### 2.1.3 System Hardware abstraction

Assumptions about the underlying processor architecture are abstracted to a single core component, easing porting to new architectures.

## 2.2 Design Assumptions / Deficiencies

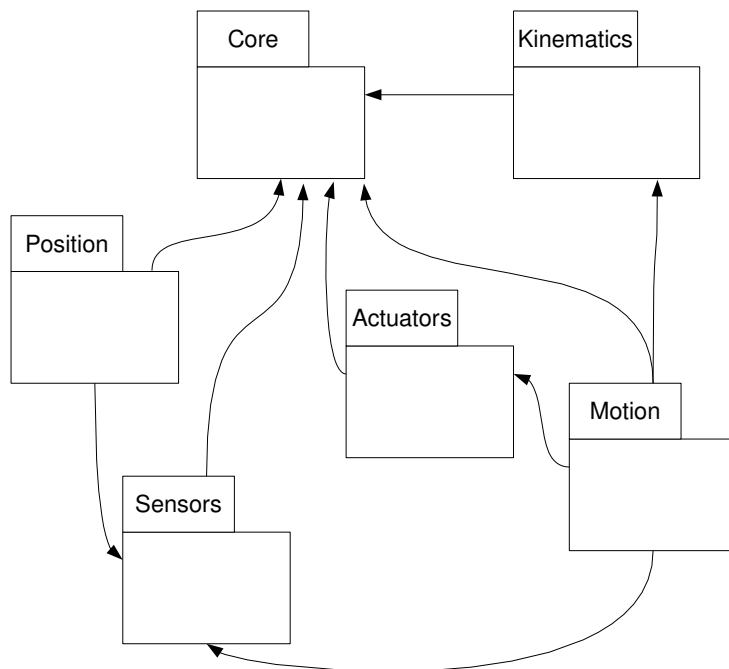
The following assumptions have been made in the design of the LRF:

### 2.2.1 Adequate similarity among a class of hardware

Abstracting a class of hardware can impose performance and functional consequences. It is assumed that the division of hardware classes is sufficient, providing similar enough interfaces, that both issues are minimised. As performance on low end hardware is critical to the project, performance inefficiencies are particularly important. (This paragraph needs cleanup)

### 3 High Level Architecture

The packages which make up the LRF are detailed below.



#### 3.1 Core

The core contains code common to other components. It includes maths operations (vectors), the framework for object orientation and abstractions for processor assumptions (such as word size, endianness).

#### 3.2 Device packages

Each of these packages contain generic interfaces for device classes. Components implementing these interfaces will also be included for specific devices.

##### 3.2.1 Actuators

Interfaces included: Light, Servo, Servo Controller, Motor controller, Sound, Linear Actuator

##### 3.2.2 Sensors

Interfaces included: Accelerometer, Gyroscope, GPS, Range Finder, Button, Switch, Camera

#### 3.3 Logic packages

Logic packages contain higher level components. Implementations in these packages use interfaces from the Actuator and Sensor packages rather than directly accessing hardware.

### 3.3.1 Motion

The motion package provides an interface to abstract the motion of a Robot. It includes concrete components for several types of motion, such as wheels, legs or wings.

(Discussion: Should we really be stating what each of these components are composed of? There may be more advanced implementations here which compose of other things, such as a wheels system which implements traction control. This may need accelerometers, etc.) - **We could make these use cases, to give a clearer example of how the interface will be used.**

- **Wheels** coordinates several motor controllers and steering servos.
- **Legs** coordinates several servo controllers using the data from an accelerometer.

### 3.3.2 Position

Position provides an interface and implementations to determine the current position and/or orientation. Different implementations can be written for the available hardware configurations.

The LRF will provide one implementation which uses a combination of GPS, accelerometer and gyroscope data to calculate an estimate of current position and orientation.

Other optional implementations, developed if time permits, may estimate position with more constrained hardware, such as when only accelerometer data and motion commands are available.

### 3.3.3 Kinematics

The kinematics package provides all common kinematics functionality, such as position, acceleration and relative velocity calculations. The package may make use of other Device Packages.

## 3.4 Testing Component

The testing component will contain automated unit tests for other components in the framework.

## 4 Interface Descriptions

These will likely change as each component is designed at each phase, once their related hardware is studied more thoroughly.

(Are we describing all functionality of each interface here, or just what the interfaces do as a whole?) - **I intended it to be the same detail you see on a java interface - ie. just the method signatures for the interface.** (Okay then. I'm just going to guess and we can discuss later on. We should probably decide on these formally as a group though. Bring on the next meeting.)

### 4.1 Actuators

Most actuators can only perform one action.

```
int doAction(generic[] data);
```

### 4.1.1 Servo Controller

(I have moved this from under Sensors to here, under Actuators). Actuator functionality as well as:

```
float getPosition();  
int setPosition();
```

## 4.2 Sensors

To get the data (most sensors can only grab data in one way)

```
generic getData();
```

All other **get** functions will be specific to each implementation.

## 4.3 Motion

To set target co-ords individually:

```
int toPosition(float x, float y);  
int toPosition(float x, float y, float z);
```

Or to send a vector instead:

```
int toPosition(Vector);
```

## 4.4 Position

To get x, y or z co-ords individually:

```
int getX();  
int getY();  
int getZ();
```

Or to get it all in one vector:

```
Vector getPosition();
```

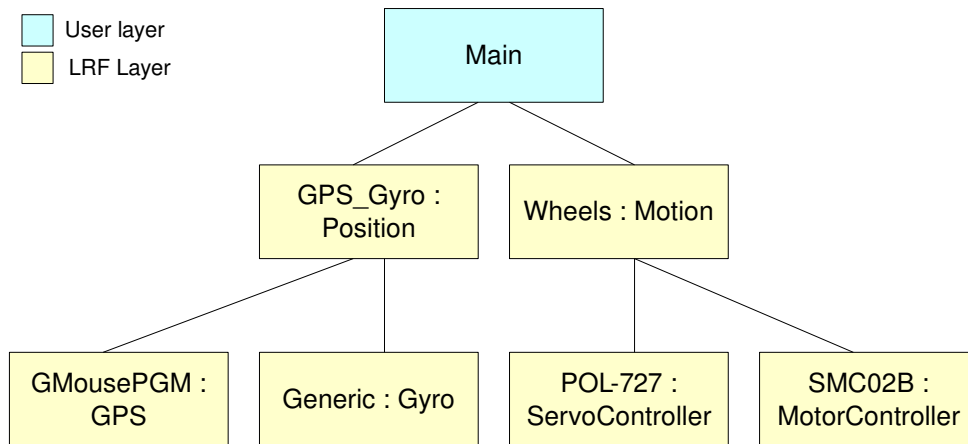
# 5 Use Case Realisation

This section shows how the various components work together to realise real world use cases.

This section is not exhaustive and may be revisited in later phases of the project.

## 5.1 Car

This object diagram shows the basic structure of how a remote control car could be automated with the LRF.

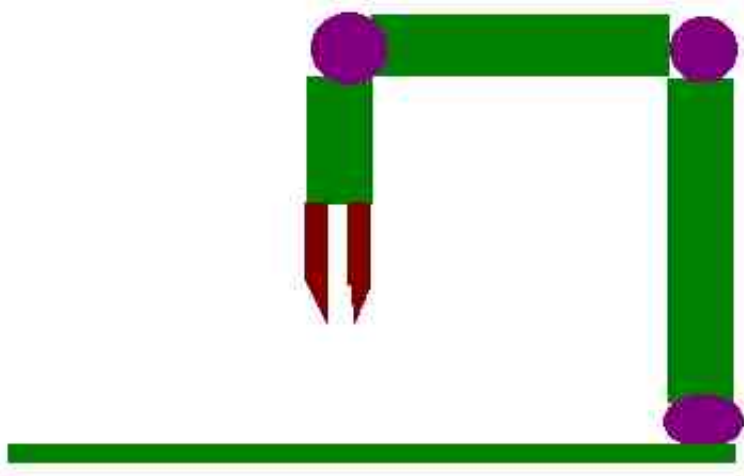


The *Main* class is written by the developer of the car. It takes input from a user and utilises the LRF interfaces to control the hardware. In this example, the car uses a POL-727 servo controller for steering, so the related component is chosen to implement the ServoController interface required by the Wheels component. The code in *Main* is independant of the underlying chosen implementations, so if hardware is swapped out it will ideally run with little modification.

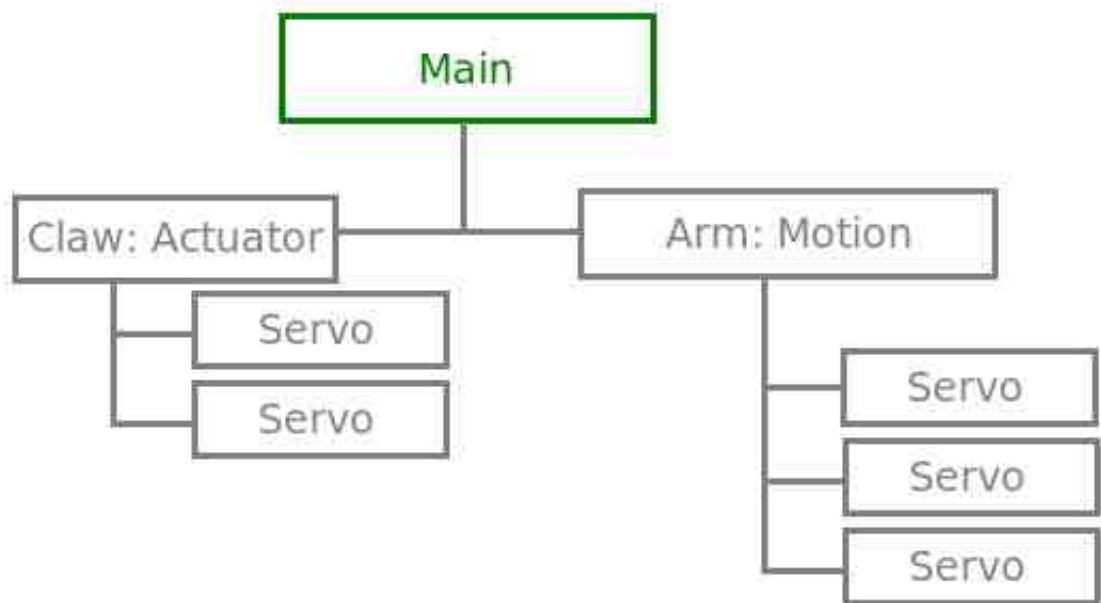
(Is this paragraph too informal/off-topic? Should it just stick to the implementation points?) I don't think that having the model numbers for the servo controllers is necessary, but the point that any servo controller may be substituted is important.

## 5.2 Arm

The object diagram below shows a possible structure for how a robotic arm could be implemented using the LRF. This example is made up of three arm pieces and a claw, as shown below. This hardware is controlled by five servos: one to rotate the arm itself, two to act as 'elbows' and two to work the gripper.



The *Main* class is written by the developer of the arm. It takes input from a user and utilises the LRF interface to control the arm behaviour. The position of the arm is controlled by a Motion component - the component is given a position for the claw and the servos are adjusted to obtain this position (this sentence is pure shite. Someone please fix it. Or delete it). The claw is controlled by an Actuator component.



**Note:** these images need to be cleaned up. I figure that all images should be in the same style.

## 6 Architecturally significant aspects

### 6.1 Deployment platforms

The framework is targeted specifically at systems running the Linux kernel, and no extra effort is expected to be undertaken by the development team in support of other platforms.

The framework will be targetting the GCC compiler, at least being fully functional on version 4.1 (as is installed on partch and the project server).

### 6.2 Coding Conventions

- Detail of the object oriented system used for C, linux kernel coding style document, etc.

### **6.3 Packaging**

- Detail the method of packaging the framework for distribution to users.