Deliverable D5.1
Software and documentation of architecture for audio integration

Collaborative Project - EARS	
Editor(s)	G.Rump, R.Gelin
Responsible Partner	Aldebaran Robotics
Status-Version:	Revision B – final
Date	30/06/2014
EC Distribution:	Public

Document History

Rev.	Issue Date	Description of Change	Author
A	17/06/2014	Draft First Version	G.Rump
B	30/06/2014	Final Version	G. Rump
C	Date	Short description	

Revision	Summary of Changes	
	Reference	**Description**
A	All chapters	Draft – First issue
B	All chapters	Final Version

Grant Agreement Number:	609465
Project Full Title:	Embodied Audition for RobotS
Project Acronym:	EARS
Call (part) identifier	FP7-ICT-2013-10

Title of Deliverable:	Software and documentation of architecture for audio integration
Date of Delivery to the EC:	30/06/2014

Work package responsible for the Deliverable:	WP5 – System Integration and Validation
Editor(s):	G.Rump; R.Gelin
Contributor(s):	G.Rump, C. Le Molgat, R.Gelin
Reviewer(s):	R.Gelin
Approved by:	R.Gelin

Abstract:	
Keyword List:	Signal processing, audio acquisition, modularity, NAOqi

TABLE OF CONTENTS

1 INTRODUCTION

This document is a presentation of the deliverable 5.1. The deliverable itself is a software component that is made available to the partners on a FTP site: ftp://ftp3.aldebaran-robotics.com/

On the site, are made available:

- A library including the Modularity framework
- Full HTML documentation of the Modularity framework
- A SDK including Modularity allowing the development of applications
- A Virtual Machine to run the development environment under Windows, MacOs or Linux.
- A tutorial explaining how to use the environment.

In this document, we remind the purpose of the development and the main principles of the Modularity module.

2 PURPOSE OF THE DEVELOPMENT

Nao has been designed as a development platform on which a user can develop its own robotic applications based on the framework NAOqi that provides all the basic robotic functions (motion of the joints, walking, speech to text, automatic speech recognition, sound localization, face detection by vision…). A graphical development environment, Choregraphe, gives a simple way to assemble these basic functions, as boxes connected to each other, to build rapidly robotic applications. These functions can also be called from Python or C++.

The Software Development Toolkit originally provided with Nao did not really offer the possibility to improve the content of the provided boxes. These boxes were mainly “black” boxes. If their APIs are public, their content is not. As long as application development is concerned, it does not matter. But when researchers wanted to improve the feature of a box, the architecture was not open enough.

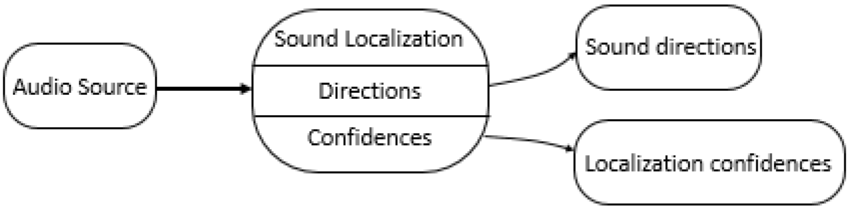
Within the EARS project, the academic partners of Aldebaran wants to have a lower access to the sensor signals to bring their own processing. They need to access the different levels of processing.

The purpose of the development made by Aldebaran within the task 5.1 is to represent the signal processing algorithm by diagrams of filters. Each proposed filter can be replaced by a filter developed by a partner and new filters can be created and integrated in a process. This approach will enforce a better reusability of most of the code/algorithm/image primitives and will avoid that each developer has to duplicate or redo the same things multiple time.

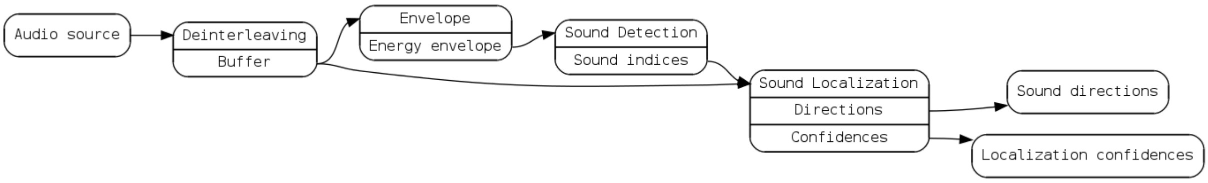
The access to this level of detail should probably not be made available to any developers. It is restricted to the Aldebaran's developers and to the partners of the EARS project and other collaborative projects.

Within the EARS project, the partners are mainly interested in audio processing but the Modularity approach concerns vision processing as well.

As a simplified example, we describe the function of the sound localization that will be improved by the partners of the EARS project. In the previous architecture, this feature was made in a very efficient way by an optimized function taking the sound signal as an input and giving, as output, the computed direction of the audio source and an estimation of the confidence of this direction.



This sound localization function is, internally, made of several filters, chained to each other. Each of these filter can be optimized and reused in another function like source sharing for instance. Modularity provides the tools for the generic description filters and the assembly of filters to create a new feature.



3 FEATURES

3.1 GLOSSARY

- A filter chain links to **source(s)** and **sink(s)** is a **process**.
- A filter chain computed is a **process** update.
- A **scheduler** can compute several different **processes** in parallel.

3.2 MAIN FEATURES

- Modularity
  - Allows to manage filters, sources, and process life.
  - Checks the data type between filters to insure that a "filter chain" is valid.
  - Provides data Type to optimize type casting (no dynamic cast)
  - Provides source signal for each video device and resolution (up to QVGA today).
  - Provides source signal for each audio device (experimental)
  - There is **no synchronization** between processes.
- Scheduler
  - Each filter can't be processed in parallel (i.e., avoid doing the same computation).
  - A scheduler evaluates periodically each process using a priority scheduling.
- Memory
  - Automatic memory management (use buffer pool + smart ptr).
- Processes
  - Connect inputs to sources and outputs to sinks of a filter at runtime.
  - A sink can emit an event to trigger the evaluation of another filter chain.
  - A sink can execute a callback each time it is updated.
  - A process can be updated once a sink from another process have been updated (i.e. reverse event)
  - Priority and frequency of a process can be updated on the fly.
  - Process dependency management is performed automatically.
- Filters
  - Apply an algorithm on inputs, provide results through outputs.
  - Manage filter's input and output ports.
  - Input ports can accept several types.
  - Allow to easily build a filter by inheriting from UserFilter class.
  - Allow to easily build a filter from basic filters. (i.e. having composite filter)
- Tool
  - a "small" script language to easily build complex processes.
  - a Modularity Tool to prototype/write/test processes.
  - Can generate a Dot Graph of a filter with specified depth level.