# Flexible Modular Robotic Simulation Environment for Research and Education

Dennis Krupke
Dennis.Krupke@informatik.uni-hamburg.de

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
**Technical Aspects of Multimodal Systems**

May 15, 2012

# Table of Content

# Modular Robots

# Simulation Tools

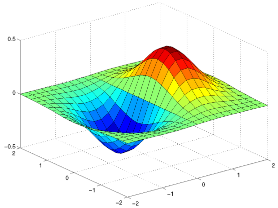
## Common systems for simulation of control algorithms

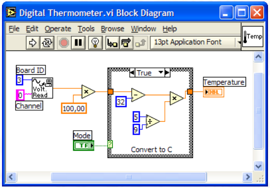
### Focus on mathematics

- Matlab
- Octave
- Scilab

### System flow centered

- LabVIEW



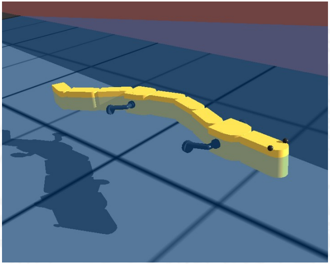



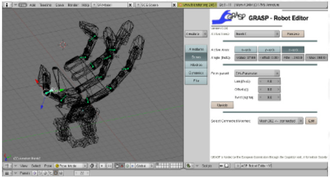
# Simulation Tools Cont.

## Systems for simulation and control of robots

### Interactive and integrated systems

- Player-Project
- Webots
- ROS
- OpenRAVE
  - OpenGRASP
  - GRASPit!



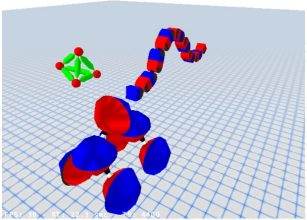


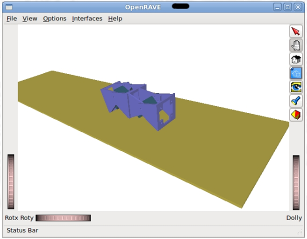
# Simulation Tools Cont.

## Systems for simulation and control of modular robots

### Simulating Modular Robots

- Unified Simulator for Self-Reconfigurable Robots (USSR)
- OpenMR





# Demands

## What is needed for efficient application?

- easy-to-use
- flexibility
- useful for beginners and experts
- reasonable results
- extendability

# Demands

## What is needed for efficient application?

- easy-to-use
- flexibility
- useful for beginners and experts
- reasonable results
- extendability

# Demands

## What is needed for efficient application?

- easy-to-use
- flexibility
- useful for beginners and experts
- reasonable results
- extendability

# Demands

## What is needed for efficient application?

- easy-to-use
- flexibility
- useful for beginners and experts
- reasonable results
- extendability

# Demands

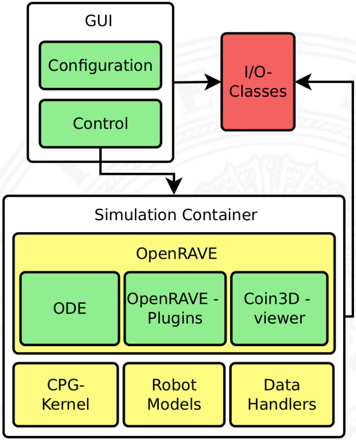
## What is needed for efficient application?

- easy-to-use
- flexibility
- useful for beginners and experts
- reasonable results
- extendability

# System Architecture

## Component Based View

- graphical user interfaces
  - configuration wizard
  - expert configuration dialog
  - control window
- simulation-/control-core
- data I-/O
  - calculated values
  - configuration files



# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Main Features

## Most important features of the proposed system

- two different kinds of graphical configuration interfaces
- reusability
- extendability
- interactivity
- data recording
- control of real robots
- support of OpenRAVE plugins

# Easy Configuration of the Simulation

## Configurability

To enable the user to set up a simulation very fast, **configuration file writers** have been created that can be accessed by the GUI:

- robot
- sensors
- actuation
- environment
- global properties

▸ Structure of the file format.

# Extending the Library of Control Algorithms

## Extendability

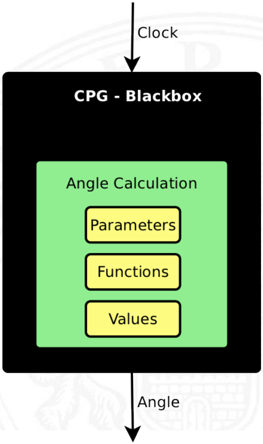
- New control algorithms can be added by the user with the graphical configuration interface.
- Combination of **self-registering types** and **dynamic class loading** allows to extend the library of user-defined control algorithms during the runtime of the program.

# CPG Blackbox

## User's view to the control algorithms

The user only needs to take care of how to calculate the next joint positions using:

- parameters
- functions
- interim results



# Implementing New Control Algorithms

## How can this be done, easily?

Implementing a new control module needs just to add a small code snippet:

```
current_time = old_time + Stepsize;
for (int jointNr=0; jointNr<_numOfJoints; jointNr++)
{
  current_angle(jointNr) = Amplitude
            * sin(2*PI * Frequency * current_time
                      + jointNr * PhaseDifference);

  SetAngle(jointNr, current_angle(jointNr));
}
```

# Data Handling

All data of interest can be stored to XML-formatted files.

- control algorithms
- sensor information
- robot information

## Exporting

Data series of single types can be exported with the GUI for later usage with GNU-Plot, Matlab or other tools.

# Configuration GUI

The configuration GUI allows to write down all information to a configuration file that is neccessary to run a proper simulation.

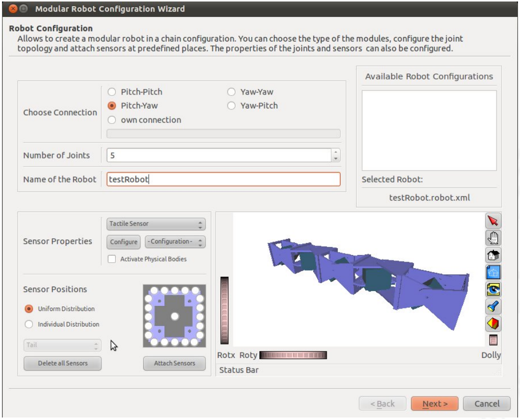
- robot
  - modules
  - topology of joints
  - sensor positions
- sensors
- control algorithms and their assignment to the joints
- environment
- simulation parameters

# Beginner's Configuration Wizard

- every neccessary adjustment will be done until the wizard is finished successfully
- explanations of the current page are presented to the user
- mandatory fields assert valid configurations

# Beginner’s Configuration Wizard Cont.

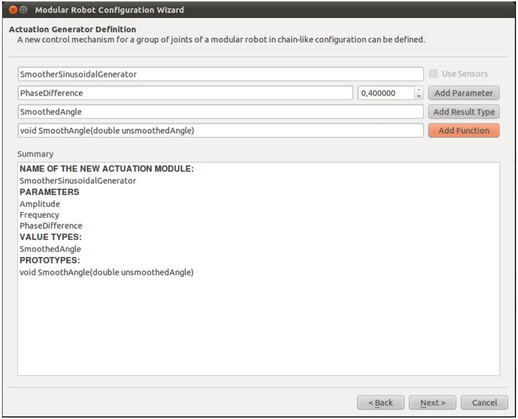
## Robot configuration




- number of modules
- topology of joints
- sensors

# Beginner's Configuration Wizard Cont.

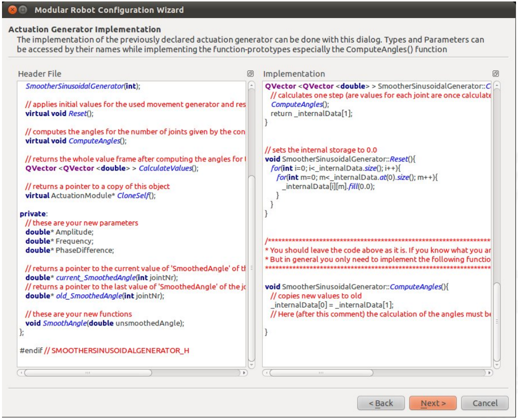
## Actuation generator definition




- parameters
- interim results
- prototypes of functions

# Beginner's Configuration Wizard Cont.

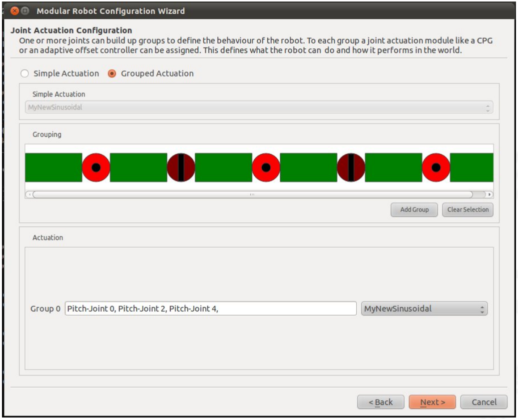
## Actuation generator implementation




- C++ header access
- C++ implementation
- full access to the new algorithm

# Beginner's Configuration Wizard Cont.

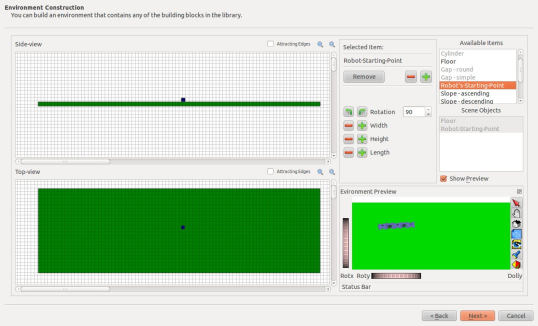
## Joint actuation configuration




- building groups of joints
- assignment of algorithms to groups
- groups are allowed to overlap

# Beginner's Configuration Wizard Cont.

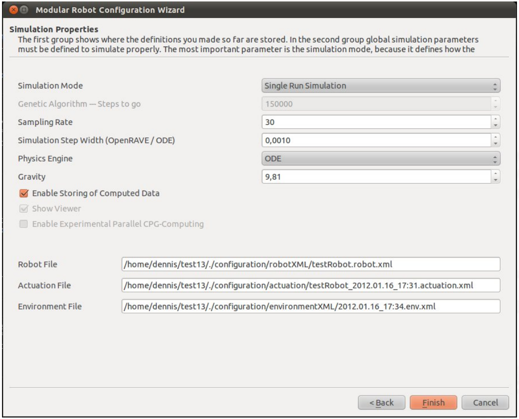
## Environment construction




- creation of several objects
- terrain can be created according to the needs
- previewing in a 3D viewer
- manipulation of the scene with two 2D projections

# Beginner's Configuration Wizard Cont.

## Simulation properties




- simulation mode
- physics properties
- sampling and accuracy
- storing data
- summary of included configuration files

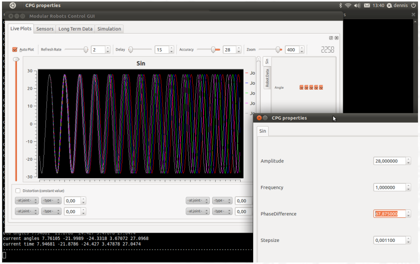
# Expert's Configuration Dialog

- users can decide what to configure
- seperated creation of configuration files
- useful for adding new components
- time-saving *reusage* of configuration files
- recombinations are possible

# Virtual Robot

## Controlling a Virtual Robot

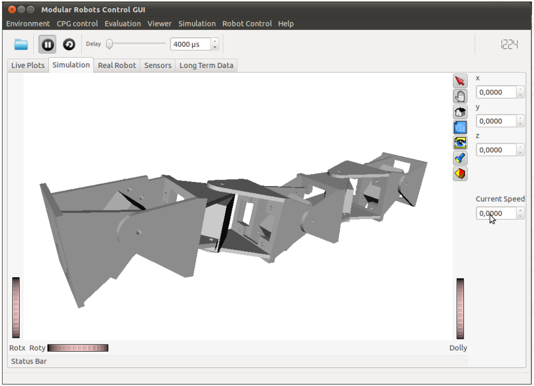
- manipulation of the scene with the 3D viewer
- interactive modulation of the control algorithms
- supervision of control algorithms → live-plots



# Real Robot

## Controlling a Real Prototype

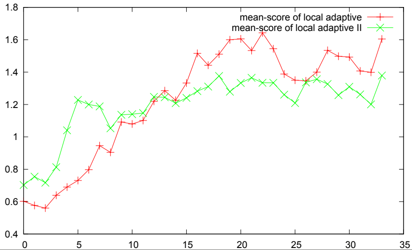
Same implementation of control algorithms can be used for real robots:



# Future Work

## Next Steps

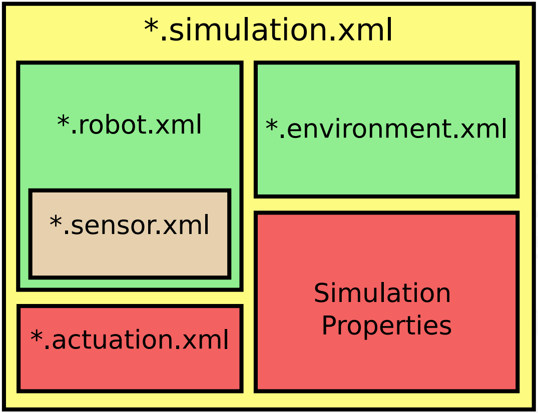
- extending the configuration interface
  - integration of *module creation* into the configuration interface
  - adding a page to set simulation runs for automated optimization
- evaluation by people at different knowledge level



# The End!

Thank you for your attention!

# Configuration File Format




- simulation
- robot
- sensor
- actuation
- environment

Back to feature description.
Back to configuration interface description.

# Self Registering Types

Each class gets a factory-proxy that registers the *name* of the current class and a *pointer to its maker-function*:

```
class FactoryProxy {
public:
  FactoryProxy(){
    // registers the maker-function
    Factory["ReflexControl"] = maker;
  }
};
```

- allows very flexible software
- users can extend the software at runtime

# Factory

## How to Use a Factory

### Construction

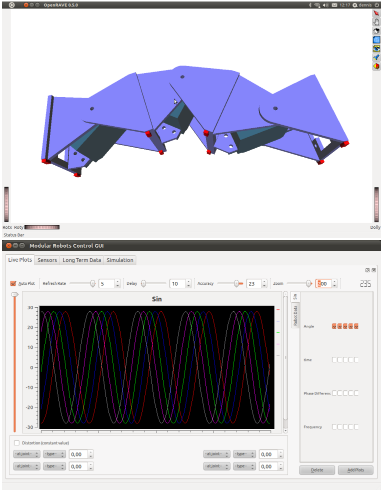
The right side of the assignment calls a *maker-function* which invokes the constructor of the current class and returns a pointer to the created object.

```
ActuationModule* controlModule = cpgFactory["MTRAN"](5);
```

⇒ Objects can be created, even if the name of the specialized class is not known directly.

Back to feature description.

# Visualization




- Coin3D / OpenRAVE-Viewer
- QWT – live-plots

# Other libraries

- Qt
- ODE
- GAlib
- Boost
- OpenMR