FLEXIBLE MODULAR ROBOTIC SIMULATION ENVIRONMENT FOR RESEARCH AND EDUCATION

Dennis Krupke*, Guoyuan Li and Jianwei Zhang
Department of Computer Science
University of Hamburg
Email: {3krupke, li, zhang}@informatik.uni-hamburg.de

Houxiang Zhang and Hans Petter Hildre
Faculty of Maritime Technology and Operations
Aalesund University College
Email: {hozh, hh}@hials.no

*corresponding author

KEYWORDS

Modular robots control, educational software, OpenRAVE, interactive simulation

ABSTRACT

In this paper a novel GUI for a modular robots simulation environment is introduced. The GUI is intended to be used by unexperienced users that take part in an educational workshop as well as by experienced researchers who want to work on the topic of control algorithms of modular robots with the help of a framework. It offers two modes for the two kinds of users. Each mode makes it possible to configure everything needed with a graphical interface and stores configurations in XML files. Furthermore, the GUI not only supports importing the user's control algorithms, but also provides online modulation for these algorithms. Some learning techniques such as genetic algorithms and reinforcement learning are also integrated into the GUI for locomotion optimization. Thus, its easy to use, and its scalability makes it suitable for research and education.

INTRODUCTION

Robotics is one of the best ways to motivate young people to work with new technologies. As explained in (Daidié et al., 2007) especially modular robots are very popular among students. Simulative robotic systems are usually developed to help them to get familiar with robotics. On the other hand, these simulative systems are also good testbeds for research. Experienced people can save time by using the functions of the proposed framework and focus on the development of new algorithms.
A lot of robot simulation software has been developed. There are general purpose simulators for mobile robots in 2D, 2.5D and 3D, like Player/Stage (2D), Gazebo (2.5D) from the *player-project*, described in (Player-Project, 2008), and Webots (3D) that can be found in (Michel, 1998). All of them offer so much freedom to the users that it is hard for them to start immediately with a simulation according to current needs. There are also special purpose simulators for grasping like GraspIt! (Miller et al., 2004) and OpenGRASP (León et al., 2010) that are based on OpenRAVE. But to the best of our knowledge there is no special purpose simulation software for modular robots that allows for fast and easy creation of a simulation setup while being easy to use and easy to understand.
A modular robot GUI has been developed that enables the user to focus on robotics while most of the programming part is hidden. This idea is also described in (Zhang et al., 2006). In contrast to other powerful systems only few rules have to be learned for proper use of our system. Motivation is the most important aspect for people who have just begun with something new to proceed and succeed. The GUI enables the user to get results very quickly because only some basic knowledge about the application space of modular robotics is needed. This makes the simulation system suitable for educational purposes with hard time restrictions, typical for a workshop or project. The research-related user also benefits from these properties. In research an expert configuration interface can be used to create several simulations. Some parts in the hierarchy of a complete configuration, for example the robot or the environment, can be reused to save time and to make different control algorithms comparable. To compare different control algorithms in a reliable way, an abstract base class has been developed that is used to implement several locomotion algorithms published by different institutions. It has been tested with arbitrary algorithms like the MTRAN-CPG (Murata et al., 2002), different networks of CPGs based on spiking/bursting-neurons, described by (Herrero-Carrón et al., 2010), and many more. The extension of the library with control algorithms can be performed with the help of software development patterns described in (Gamma et al., 2005) and (Beveridge, 1998). Methods to analyze the kinematics, described in (Hirose et al., 1990) and (Hirose, 1993) can be added and the results passed to special data handlers. These handlers are needed to visualize the data and to write it to the disk.
In the following a short introduction to the modular robots GUI and its components that has been developed is given. It is followed by a detailed description of several parts of the software and an example that shows how the system can be used, before the final section presents the paper's conclusion and raises possible future work.

Proceedings 26th European Conference on Modelling and Simulation ©ECMS Klaus G. Troitzsch, Michael Möhring, Ulf Lotzmann (Editors)
ISBN: 978-0-9564944-4-3 / ISBN: 978-0-9564944-5-0 (CD)

INTRODUCTION TO THE MODULAR ROBOTS GUI

For achieving realistic and reasonable simulations, the simulated world should look and behave as close to the real world as possible; therefore a system using a physics-engine and a 3D-viewer is needed. OpenRAVE (Diankov, 2010) offers calculation of the physics and three-dimensional visualization of the running simulation. It is a controlling and planning software that is developed as a general robotics simulation and control system. It can be extended easily to fulfill the demands of the user. In our simulation system many components of OpenRAVE have been applied:

- 3D-viewer interface and implementation of Coin3D
- Physics engine interface and implementation of ODE
- XML- and COLLADA readers
- Abstract sensor interface and implementations of several sensors

ODE (Smith, 2006) works as a physics engine to calculate realistic behaviour of the rigid bodies. After the robot has been assembled, sensors can be attached to the robot and the configuration can be stored in an XML-file. With another XML-file the user is able to describe a world. Concepts like *cloning* allow us to run a copy of a simulation. This is ideal to run copies of simulation in parallel or to start an identic simulation again with only a small change in the configuration of the control algorithms. OpenRAVE has become very popular because of its modules that calculate the inverse kinematics for several industrial robot arms. It is under intense development and the support using the user-list is very fast. Many institutes from different countries are currently working with OpenRAVE. That makes it an ideal base for our ideas. By now, OpenRAVE offers a package for the Ubuntu Linux operating system that enables the automatic installation of all needed dependencies including Coin3D, SoQt, ODE and many more. In this way, it is very comfortable to install OpenRAVE on a standard Ubuntu system.

A plugin for OpenRAVE (Gonzalez-Gomez, 2010) the *OpenMR-plugin* has been written by Juan Gonzalez-Gomez. It implements a servo controller as shown in Fig.1. In modular robotics these are used to drive robots. The controllers have been applied to our system as well as the 3D-model of a modular robot prototype included by the plugin. The robot model that has been used in our experiments is described in (Gonzalez-Gomez et al., 2006).

To access the core simulation system including the physics-engine and the 3D-viewer and the robot models, *graphical user interfaces* for comfortable handling have been implemented. In addition *I/O-classes* and a *kernel for control algorithms* used in our *simulation container* have been added.

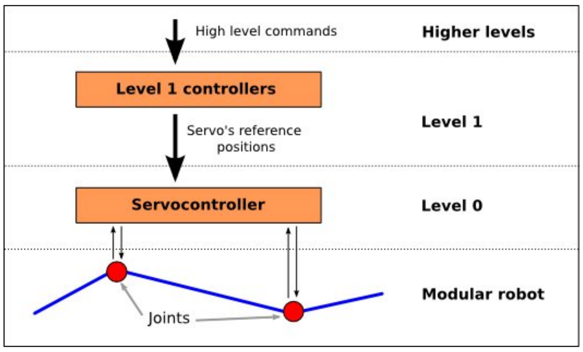


Figure 1: Concept of the used servo controller (Figure is taken from (Gonzalez-Gomez, 2010)). As level 1 controllers several algorithms has been implemented using the abstract interface for actuation modules.

The GUI elements and many classes, extending the standard library of C++, are taken from *Qt*, *Boost* and *QWT*. One of the main features is that the user can easily create his own algorithm for joint control. This can be a locomotion algorithm or an adaptive algorithm to adjust the shape of the robot. Another feature is that all parameters of this new algorithm can be accessed and changed while the simulation is already running. This makes it possible to experiment with different values to gain a better understanding of the algorithm. It also allows us to implement online-optimization methods to find good sets of values for the parameters for different purposes. The GUI is divided into two main parts, the *configuration-* and the *control-GUI*. The configuration part permits the setting up of everything that is neccessary to run a certain simulation while the control GUI offers online control over the control algorithms and supervision of the robot within its world. Additionally it is possible to see certain live plots of different data.

To enable the user to configure simulations, a **configuration GUI** with two different modes has been implemented. One interface for a user at the *beginner level* and one for an *intermediate or experienced user*. A level of detail has been found that is complex enough for reasonable usage. Beginners get in touch with a *configuration-wizard* that guides them step by step through the process of configuration. Clear instructions explain the current task and the meaning of the information that needs to be filled in. The user is not asked for filenames as everything gets stored automatically. The filenames are generated automatically at predefined places within the project folder. The *expert-configuration-dialog* makes very flexible configurations possible. Both configuration interfaces allow the user to create and *extend the user-library* of control algorithms for robot-joint-actuation during the runtime of the program. OpenRAVE utilizes XML-format to define robots with its sensors, physical properties and the environments the simulation is set in. This property has been extended with actuation-configuration files to assign control algorithms to the robot very easily.

In this way, as many control algorithms as needed can be assigned to many different groups of joints. Each group must consist of at least one joint. Overlapping groups are allowed. Complex architectures can be built from basic building blocks.
The function of the **control GUI** is to supervise the simulation with the help of several visualizations and to adjust everything just in time while the simulation is running. This can be used to do experiments in an explorative way, manually by a user or automatically by an optimization-process. To supervise the control algorithms and the robot, applied to the current simulation, a monitoring system has been included. It allows us to select and combine several data plots to show them on the screen. These are *live-plots*, used to see the current state of the simulation. This combined with the possibility to adjust the parameters of the locomotion algorithms on-line makes the GUI a powerful system for interactive experiments and automated optimization.
To record all of the calculated data of interest, a *data file writer* has been developed that stores everything to an XML-file. Special readers for this file format are able to extract a single series of data. Data series can be exported into separate files for later usage.

DETAILED SYSTEM DESCRIPTION

Our software consists of the following parts.

- Configuration GUI
- Control GUI
- Simulation container
  - CPG kernel
  - Data handlers
  - OpenRAVE as core simulation system
- Several I/O-classes

The structure of the simulator is also shown in Fig.2.

Configuration GUI

The configuration GUI can be started in two different modes. One is a *wizard* for simple usage that guides the user through all steps that are neccessary to set up and start a simulation. The *expert mode* allows us to configure only the neccessary steps seperately. It can be used to generate configuration files for only a single step of the whole configuration process and to reuse configured parts that were already defined.
The *configuration wizard* is made for people who are not familiar with this software or are just starting with modular robots. Without knowledge about the simulator and some basics about C++ it should be possible to create a robot, to define the behaviour of its joints and to create a world, the workspace of the robot. Only a few ideas about locomotion principles of modular robots in chain configuration will be needed to fullfill this task. This also

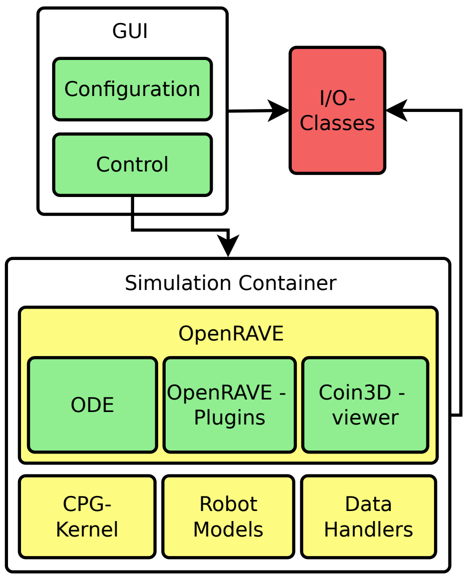


Figure 2: The architecture of the simulator.

means that everybody dealing with computer science is able to implement a simple locomotion algorithm after a short introduction to the locomotion principles of modular robots.
The *expert mode* is intended for the experienced user who wants to create different simulation set-ups to compare the results for qualitative analysis. The expert mode is useful because the configuration of a whole simulation consists of many parts and it is possible to reuse some of them for another simulation set-up to save time and avoid monotonic work. A simulation configuration file containes three parts:

- the *robot configuration* file with all of the physical parts of the robot and the sensors
- a configuration for the *joint actuation* of a specific robot
- the definition file of the *environment*

Complex environments, suitable for tasks of a robot described in (Granosik et al., 2005), can be built by using the integrated environment editor. Additionally, there are some global options that can be adjusted to determine the simulation mode and some other properties.

Control GUI

The control GUI offers different tabs to give structured access to the different components. It communicates with the core of the simulator with the help of the *Signals and Slots principle* used by Qt. In this way the delay between two simulation steps of the discrete simulation can be changed. When the simulation is started and the user has configured the simulator to store the calculated data, all of the data gets stored to a temporary file after a

certain period. The user can store this data to a file any time.
One tab is for the *data live-plots* and enables the addition of several plot widgets that can contain any combination of the available data type. To select the desired combination of plots a GUI element with checkboxes, one for each avilable data series, has been created. It is possible to change the sampling rate, the refreshing rate, as well as the size of the buffer containing the data. The GUI elements used for the plotting are taken from the *QWT* library. QWT supports a lot of features that can help to display data on the screen in a clean and clear way. As a data buffer for the plots the *circular buffer* from the *Boost-library* has been used. It is easy to handle and very efficient because the single entries in the circular buffer are contiguous in the memory. Resizing of this circular buffer has been proved to be very stable in our experiments.
Another tab offers access to the used *sensors* in the current robot. It is possible to change the state of the single sensors, e.g. sensors can be switched on and off.
With the help of the *data*-tab, recorded simulation data of the currently running or already past simulations can be loaded. It is possible to select one or more graphs to plot it on the screen. These plots can be exported to portable network graphics (*.png) or into a gnuplot file for later, more differentiated usage. After loading a data file, all contained types can be selected. In case of a specific value type the number of the joints is an additional criterium that must be chosen.
The last tab contains the *Coin3D-viewer* taken from OpenRAVE to visualize the simulation running in the core module with OpenGL. It supports all features from the original OpenRAVE software and can be used as described in the OpenRAVE documentation. The viewer uses very simple shading mechanisms to achieve high performance even on slower systems. Tests with an EeePC 1000H netbook have revealed that our software works on such slow systems. With the viewer the pose of the camera can be changed according to the needs of the user and videos of the simulation can be recorded. It is possible to manipulate displayed objects. Objects can be moved and joint-angles can be adjusted.

Simulation Container
The container, where the core of the simulator is running, is a class inherited from *QThread*. The thread calls all neccessary step-functions of the OpenRAVE environment and the CPG-Kernel and emits certain signals. For example, one signal is emitted after one calculation step has been completed. The speed of the simulation depends on the machine where the program is running. To achieve a short time of response in the system, an adjustable delay between the calculation steps has been added. In this way the speed of the simulation and the CPU load can be controlled.
The container can run without the graphical components. This is useful for longtime- or remote-runs of the program, for example when a parameter or an algorithm for locomotion needs to be optimized. In these cases graphical output is not needed and computational effort can be saved.

CPG-Kernel
The most important part of the work was to create an abstract interface that fulfills the needs of any general central pattern generator or other locomotion algorithms. Our interface facilitates the creation of new control-algorithms. It has been tested for arbitrary algorithms from scientific publications which can be evaluated, optimized or combined in several ways. Even distributed control algorithms that have been described in (Conradt et al., 2003) can be implemented. To integrate as much functionality as possible in the Control-GUI and to allow for further extension of the program, it has been realized as an abstract base interface that holds only the most common features of a control algorithm. To provide enough flexibility for future implementations of several locomotion or actuation algorithms it has been kept simple. But it offers some basic functionality like parameter-modulation, data-handling and supervising of the current state.

I/O-Classes And Data Handlers
There are many XML-readers and -writers to store and read all of the needed information for specifying a simulation in detail. To overcome this complexity, arbitrary classes for each category have been created. Access to these classes using a graphical interface hides the syntax needed to build valid configurations and makes the process of configuring parts of a simulation very fast, easy and robust.

- Robot-configuration-writer
- Sensor-configuration-writer
- Actuation-configuration-writer and -reader
- Environment-configuration-writer
- Simulation-configuration-writer and -reader
- Simulation-data-writer and -reader

There are three different kinds of *simulation-data* occuring in a running simulation. For each type a data handler has been implemented to manage the values.

- CPG-/locomotion-algorithm data
- Robot data
- Sensor data

For each type the values are computed after the global step-function is called. It is the same for the physics engine and the OpenRAVE-core as for our simulator- and CPG-core. A reader and a writer have been implemented to access and generate these files. The data format makes it possible to acquire specific data-series from the file even if different files contain different types of data.

EXAMPLE — HOW TO USE THE SYSTEM

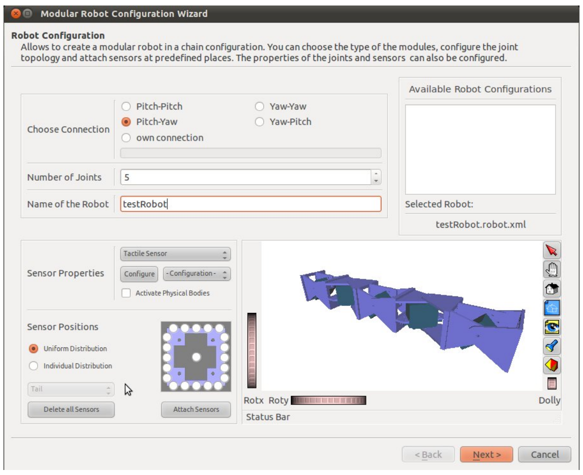
In the following the complete process to create a proper simulation is described. It includes how a new locomotion algorithm can be defined using the beginner's *configuration wizard* of the simulator and how the simulation can be started.

Starting The Simulator

After the program has been started from the command-line a dialog is shown. A button with *'Start Wizard'* has to be used to start the wizard.

Creating A Robot

The first page of the current version of the wizard, shown in Fig. 3, is intended to create a robot. It is neccessary that a reasonable name is given to the robot. It will be used for the filename-generation of the configuration file describing the robot. Because the interface is built to create robots in a chain-like configuration it is recommended to select the needed topology. To create a robot with three-dimensional locomotion capabilities the *'Pitch-Yaw'* connection type has to be selected. The size of the robot can be changed by selecting a reasonable number of modules, e.g. five. The preview in the lower right corner gets updated if anything has been changed. For simplicity no sensors have to be added this time. That is enough for a simple robot and the *'Next'* button can be clicked.


Modular Robot Configuration Wizard

Robot Configuration
Allows to create a modular robot in a chain configuration. You can choose the type of the modules, configure the joint topology and attach sensors at predefined places. The properties of the joints and sensors can also be configured.

Choose Connection: Pitch-Pitch [✓ Pitch-Yaw] own connection Yaw-Yaw Yaw-Pitch
Number of Joints: 5
Name of the Robot: testRobot
Available Robot Configurations
Selected Robot: testRobot.robot.xml
Sensor Properties: Tactile Sensor, -Configuration-, Activate Physical Bodies
Sensor Positions: [✓ Uniform Distribution] Individual Distribution

Figure 3: With the robot creation page robot description files can be created in a simple way without any knowledge about the OpenRAVE XML tags.

Creating A User-Defined Locomotion Module

A new dialog page will appear for creating a new control algorithm. With this interface all neccessary properties of the new algorithm can be declared and applied with initial values. To create a new control module that generates sinusoidal output, a name for the new module must be given first. This name will be used for the new class. To have access to the hidden functionality of the actuation modules, like plotting and storing of data, as well as live modulation of the output, some properties of the new module must be declared. All parameters must be registered one after the other and initial values must be given to initialize the new module. In this example, four parameters including amplitude, frequency, phase-difference and stepsize are typed as neccessary parameters. Using the *sine* function to calculate the output angles, an input will be needed. That is the reason why a value type called *'time'* must be added. It is related to the parameter *stepsize*. Both together will be used to generate a slowly increasing value as input for the oscillating sine function. By clicking the button *'Next'*, a header- and an implementation-file will be generated. For this simple actuation module only the *'CalculateAngles'*-function must be implemented in the implementation window of the next dialog page. A single loop that calculates the output for each joint has to be added for this purpose:

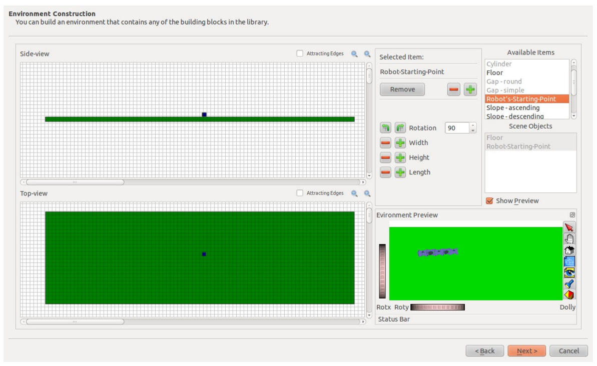
```
for (int m=0; m<_numOfConnectedJoints; m++)
{
  *current_time(m) = *old_time(m) + *Stepsize;

  *current_angle(m) = *Amplitude
              * sin(2*M_PI**Frequency**current_time(m)
              +m**PhaseDifference*M_PI/180);

  SetAngle(m, *current_angle(m));
}
```

A click on the *'Compile'* button causes the new module to get compiled to the user library.

Defining The Behaviour Of The Robot

Functionality can be added to the joints with a grouping-interface. Users can select desired joints as a group and perform them with a common control method. In this example, by clicking the button *'Add Group'*, only the pitching joints are added as a group. The newly developed control method is then applied to this group.

Creating An Environment


Environment Construction
You can build an environment that contains any of the building blocks in the library.

Selected Item: Robot-Starting-Point
Rotation: 90
Available Items: Cylinder, Floor, Gap - round, Gap - simple, Robot's-Starting-Point, Slope - ascending, Slope - descending
Scene Objects: Floor, Robot-Starting-Point
[✓ Show Preview]

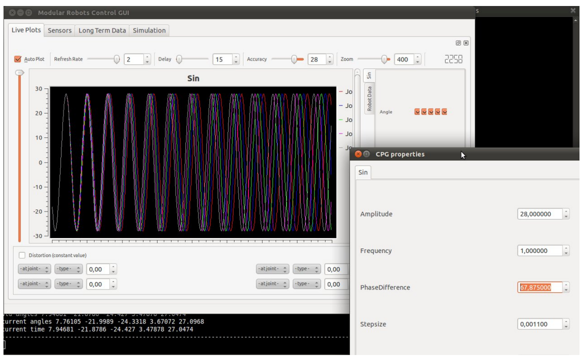
Figure 4: A very simple environment containing the robot and one floor has been constructed with the integrated environment-editor.

Arbitrary *'Available Items'* can be added to the currently empty environment with the help of *drag-and-drop* into one of the two editor views. The previously defined robot will be placed with the help of the representative object for its pose. In this way the pose of the robot can be changed. To change the position of any object it can be dragged in one of the views. To change the x- and the z-coordinate the *side-view* can be used, for x- and y-coordinate the *top-view*. The size and the orientation relative to the z-axis can be changed after the current object has been double-clicked. The selected item can be resized and rotated with the help of the buttons. To fit the created objects in the two views the zooming-buttons should be used. The preview, as shown in Fig. 4, can be eanbled by selecting the checkbox labeled with *'Show Preview'*. Each object, except the robot's position object, can be right-clicked to adjust the frictional coefficient that has a range between zero and one. In this way a slippery ground can be simulated as well as a very sticky floor.

Setting-Up The Global Simulation Properties

At the bottom of the last wizard-page the full paths of the recently created configuration files are summarized. With the upper part the *simulation mode* and some properties can be changed. *'Single Run Simulation'* must be selected to create a simple simulation that keeps running until it gets stopped by the user. With the *sampling rate* the computational effort of the simulation and its relative to the steps of the OpenRAVE-core is determined. A value of 30 means that every 30th physics-engine-step one step of our sensors and actuation modules will be computed. For the physics-engine-step size a standard value of 0.001 is recommended. Storing of the calculated data can be deactivated, if needed. After hitting *'Finish'* a master configuration file will be stored that contains the names of all other files and the newly added simulation properties. This file will be passed to the simulator that gets initialized with the information from this file.

Starting The Simulation And Using The Control GUI



Refresh Rate 2, Delay 15, Accuracy 28, Zoom 400

Amplitude 28,000000
Frequency 1,000000
PhaseDifference 67,875000
Stepsize 0,001100

```
urrent angles 7.76105 -21.9989 -24.3318 3.67072 27.0968
urrent time 7.94681 -21.8786 -24.427 3.47878 27.0474
```

Figure 5: Example of online-modulation of the phase difference between neighboring modules.

To start the simulation the *'Play'*-button, located at the toolbar of the control GUI, must be pressed. It can also be used to pause the simulation.With the *properties dialog* all control parameters of the currently applied actuation modules can be modulated online. As shown in Fig. 5 for example the phase can easily be changed using the properties dialog. To visualize the calculated data the checkboxes of the *plot-selector*-element of the control GUI can be used to add arbitrary data-lines to a new plot. The live-plot will be created after the button, labled with *'Add'*, has been pushed.

CONCLUSION

A new graphical user interface to create and control simulations for modular robots in an easy and flexible way has been introduced. The software focusses on modular robotics and their control algorithms. It tries to hide the programming part as well as possible behind the GUI. In this way even beginners with very limited time are able to use the software with only basic knowledge in writing C++ functions and little idea of modular robotics. It enables beginners to get in contact with modular robotics and intermediate users will benefit from a decrease in time for implementing their own ideas. The software fulfills two tasks. On the one hand it can be used as an educational framework to introduce somebody to modular robotics in an explorative way and on the other hand it can be used in research to get results from optimization methods. Studies and improvements of control algorithms can be done with reduced time effort. To the best of our knowlegde this is very new and offers a framework to achieve good results from the simulation of control algorithms that can be applied to real modular robots.

FUTURE WORK

Considerable works remains to be done, e.g. in extending existing parts. To reduce the restrictions in building modular robots another page to create new *modules* will be added to the configuration interface. Simple geometric objects or 3D-data of a single link will be used to create a new module available in the robot-construction page of the configuration-GUI. More *sensors* with adjustable properties will be implemented. The placing of the sensors needs to become more flexible. Creation of actuation modules using sensor information via the configuration-GUIs will be implemented. The library of objects for creation of the *environment* should also be extended to build a world suitable for more complicated tasks. More simulation modes that support the user in getting good results from the simulations will be implemented. These will include *optimization methods*, especially learning mechanisms, that improve a single parameter or a whole feature vector of a certain actuation module.

REFERENCES

Beveridge, J. (1998). *Self-Registering Objects in C++*. Dr. Dobb's Journal, Vol. 23, Issue 8, pp. 38-41, August 1998.

Conradt, J., Varshavskaya, P. (2003). *Distributed central pattern generator control for a serpentine robot*. ICANN 2003.

Daidié, D., Barbey, O., et al. (2007). *The DoF-Box Project: An Educational Kit for Configurable Robots*. Switzerland, ETH Zurich, Proceeding of AIM 2007, 4 - 7 Sept., 2007.

Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. Carnegie Mellon University, Robotics Institute, 2010.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., Booch, G. (2005). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 2005.

Gonzalez-Gomez, J., Zhang, H., Boemo, E., Zhang, J. (2006). *Locomotion Capabilities of a Modular Robot with Eight Pitch-Yaw-Connecting Modules*. Proceeding of CLAWAR 2006, Brussels, Belgium, September 12-14, 2006.

Gonzalez-Gomez, J. (2010). *OpenMR: Modular Robots plug-in for Openrave*. http://www.iearobotics.com/wiki/index.php?title=OpenMR:_Modular_Robots_plug-in_for_Openrave, 2010, (last access in March 2012).

Granosik, G., Hansen, M. G., Borenstein, J. (2005). *The Omnitread Serpentine Robot for Industrial Inspection and Surveillance*. Industrial Robot: An International Journal, Vol.32, No.2, pp.139-148, 2005.

Herrero-Carrón, F. and Rodríguez, F. B. and Varona, P. (2010). *Study and application of Central Pattern Generator circuits to the control of a modular robot*. Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2010.

Hirose, S., Morishima, A. (1990). *Design and control of a mobile robot with an articulated body*. The International Journal of Robotics Research, Vol. 9 No. 2, pp. 99-113, 1990.

Hirose, S. (1993). *Biologically inspired robots (snake-like locomotor and manipulator)*. Oxford University Press, 1993.

León, B., Ulbrich, S. Diankov, R.,Puche, G.,Przybylski, M.,Morales, A.,Asfour, T.,Moisio, S.,Bohg, J., Kuffner, J., Dillmann, R. (2010). *OpenGRASP: A Toolkit for Robot Grasping Simulation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

Michel, O. (1998). *Webots*. http://www.cyberbotics.com, 1998, (last access in March 2012).

Miller, A., Allen, P. K. (2004). *Graspit!: A Versatile Simulator for Robotic Grasping*. IEEE Robotics and Automation Magazine, Vol. 11 No. 4, Dec. 2004, pp. 110-122, 2004.

Murata, S., Yoshida, E., Kamimura, A. , Kurokawa, H., Tomita, K., Kokaji, S. (2002). *M-TRAN: self reconfigurable modular robotic system*. IEEE(ASME) Transactions on Mechatronics, December 2002.

Smith, R. (2006). *Open Dynamics Engine*. http://www.ode.org, 2006, (last access in March 2012).

The Player Project (2008). *The Player Project*. http://www.playerstage.sourceforge.net, 2008, (last access in March 2012).

Wall, M. (2007). *Matthew's Genetic Algorithm Library* http://lancet.mit.edu/ga/, 2007, (last access in March 2012).

Zhang, H., Baier, T., Zhang, J. , Wang, W., Liu, R., Li, D., Zong, G. (2006). *Building and Understanding Robotics — a Practical Course for Different Levels Education*. Proceedings of the 2006 IEEE, International Conference on Robotics and Biomimetics, Kunming, China, December 17-20, 2006.

AUTHOR BIOGRAPHIES

DENNIS KRUPKE is student assistant at TAMS, Department of Informatics, University of Hamburg, Germany. His research interests are simulative systems and control of modular robots.

GUOYUAN LI is PhD student at TAMS, Department of Informatics, University of Hamburg, Germany. His research interests lie in locomotion control and CPG model design.

JIANWEI ZHANG is professor and head of TAMS, Department of Informatics, University of Hamburg, Germany. His research interests are multimodal information systems, novel sensing devices, cognitive robotics and human-computer communication.

HOUXIANG ZHANG joined the Department of Technology and Nautical Sciences, Aalesund University College, Norway in April 2011 where he is a Professor on Robotics and Cybernetics. The focus of his research lies on mobile robotics, especially on climbing robots and urban search and rescue robots, modular robotics, and nonlinear control algorithms.

HANS PETTER HILDRE is professor at the Faculty of Maritime Technology and Operations, Aalesund University College, Norway.

ACKNOWLEDGEMENT

This work has been supported by the BICCA project, which is a DFG project concerning the development of a biologically inspired climbing caterpillar.