Simox
A Robotics Toolbox for Simulation, Motion and Grasp Planning

N. Vahrenkamp[a,b], M. Kröhnert[a], S. Ulbrich[a], T. Asfour[a], G. Metta[b], R. Dillmann[a], and G. Sandini[b]

1 Introduction

Software development plays a major role besides hardware setup and mechanical design when it comes to building complex robots such as mobile manipulators or humanoids. Different requirements have to be addressed depending on the application. A low-level controller for example must be implemented for real-time use, whereas a task planning component will interact with the robot on a higher abstraction level. Hence, developing robotics software is subject to several constraints such as performance and robustness.

The selection of libraries or frameworks for software development is influenced by the application and the existing software environment. Therefore, several aspects have to be considered when building and/or choosing software components for developing robotics applications:

- Performance and Robustness: Although increasing CPU speed allows to run complex algorithms in reasonable amounts of time, performance is always an issue in the context of mobile manipulation and planning. Therefore often C++ is the preferred choice in robotics since it combines high level programming language features with efficient compilation techniques. Robustness can be achieved by relying on advanced coding techniques like smart pointers, exception handling and automated test cases.
- Extensibility: A library should offer easy extension capabilities. One possible way to achieve this is offering a plugin mechanism which uses common interfaces to provide uniform extension points to developers.
- Platform-independence and Dependencies: Since software development takes place on different platforms, a framework should at least offer sup-

[a] Institute for Anthropomatics, Karlsruhe Institute of Technology (KIT), Adenauerring 2, Karlsruhe, Germany,
[b] Robotics, Brain and Cognitive Sciences Department, Istituto Italiano di Tecnologia (IIT), Via Morego 30, Genova, Italy

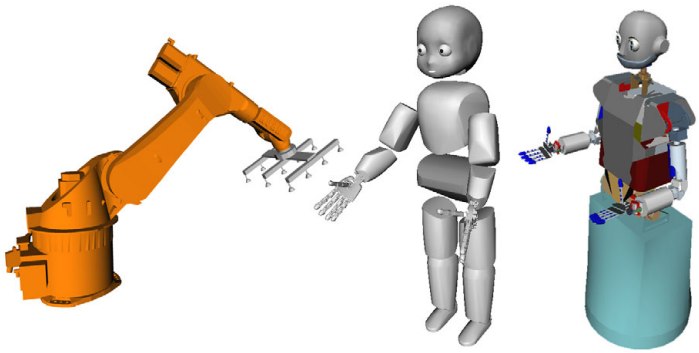


Fig. 1 Robot models realized with *Simox*: Kuka KR60-3, iCub and ARMAR-III.

port for the main operating systems Windows™, Unix/Linux, and Mac OS X™. Additionally, a low number of necessary compile-time dependencies ease the setup and compilation process of a library.

Furthermore, the license model might play an important role in selecting software libraries. In the scientific context, open-source software is preferred since it allows both reusing results from others as well as sharing own solutions with the community. The widely used GNU General Public License (GPL) is one possible open-source license, but it enforces projects that use GPL libraries to be published as GPL as well. Therefore, many libraries apply the GNU Lesser General Public License (LGPL), since it does not require derived products to be released under LGPL terms, thus allowing the usage of the library in commercial products [1].

Several open source frameworks exist in the context of robot simulation, differing in complexity, licensing, and objectives. Physics simulation libraries such as *ODE* [2], *IBDS* [3] or *Bullet* [4] usually offer possibilities to define constrained objects which can be used to setup joints of a robot. Based on these joints the kinematic structure of a robot can be defined. Motion planning libraries (e.g. *OMPL* [5], *MSL* [6], *MPK* [7]) and grasp planning frameworks such as *GraspIt* [8] or *OpenGRASP* [9] either rely on external frameworks for defining robot kinematics or a custom kinematic representation is used. With *OpenRAVE* [10] a comprehensive simulator for motion and manipulation planning is available under an open-source license. Although a lot of features are offered with *OpenRAVE*, a large number of external libraries have to be installed in order to take advantage of the full functionality. Users can write plugins in order to interact with the simulator and therefore the whole system needs to be running. In contrast Simox explicitly offers several lightweight libraries that can be used within a custom project without handing over the control to a simulator. The *Robotics Toolbox* offers a set of useful MATLAB™ scripts covering a wide range of algorithms related to robotics [11].

In contrast to existing frameworks for robot simulation, *Simox* tries to achieve both: Covering a wide range of simulation features while not relying on a large number of dependencies to other libraries. Having lots of dependencies may cause difficulties in installing and using the framework when the execution environment is limited, e.g. when the framework should be running on a robot.

2 Simox

With *Simox* we tried to cover many of the above mentioned requirements. The library is completely implemented in C++ while using modern programming paradigms to achieve both efficiency and robustness. Reliability being a critical aspect, especially in robotic applications, the library comes with numerous unit test cases and relies on smart pointer usage to avoid memory leaks. *Simox* is released under the LGPL license and can therefore be used both in open-source and commercial products. The low number of dependencies required to compile and use the library ensures a convenient setup and a straightforward integration into any type of project.

Simox itself is divided into the three main libraries: *VirtualRobot* provides functionality for describing robot models including kinematic chains, end effectors and visualizations. These models can be read from XML files and afterwards be used for collision detection, inverse and direct kinematics calculation as well as reachability analysis. *Saba* provides sampling-based motion planning algorithms. The library holds several configuration space representations that are used by the planning algorithms. Additionally, path processing approaches and visualization routines are included. *GraspStudio* is a library offering grasp planning capabilities. Methods for generating grasping hypotheses, evaluating grasp qualities, and planning grasps are available. Visualization methods for all generated results are included additionally.

2.1 3D Visualization

Providing meaningful and easy to use visualization methods is crucial for a good framework dealing with geometric data. In *Simox* we provide abstract visualization interfaces which are independent of the underlying 3D libraries. To support a new 3D library it is only necessary to provide the matching models and to implement the visualization interfaces using the API of the specific library. Up to now, support for the two visualization frameworks *OpenSceneGraph* [12] and *Coin3D* [13] is realized with *Simox*.

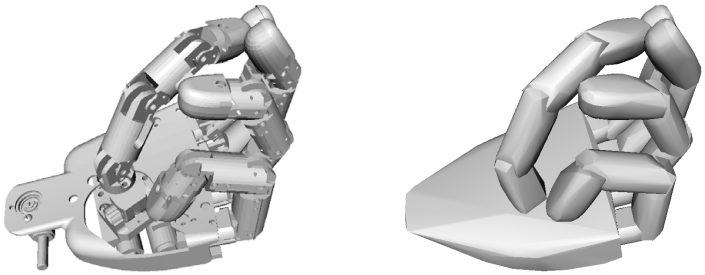


Fig. 2 The visualization model of iCub's hand (left) and its reduced model (right) that is used for efficient collision detection.

2.2 Collision Detection

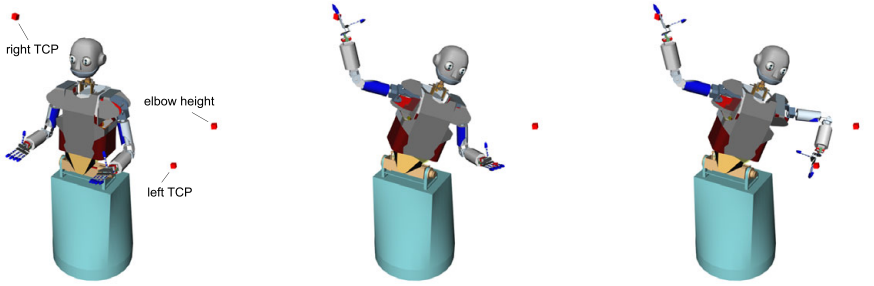
Fast collision detection is important for efficient motion and grasp planning, since sampling-based approaches need to perform numerous calls to the collision detection engine to validate sampled configurations. Hence, an efficient approach is needed, preferably not limiting the shape of the models (e.g. only convex shapes). The PQP library [14] offers efficient collision detection for arbitrary 3D models and an extended version, that is able to handle multi-threaded collision queries, is fully encapsulated by *Simox*. For collision detection we provide the same functionality as for visualization. Therefore, it is possible to use other collision detection engines by implementing the interface classes for collision detection.

3 VirtualRobot

The *VirtualRobot* library offers methods to define robots and environments and numerous simulation tools. Furthermore, advanced algorithms such as collision detection, Jacobian calculations or reachability analysis are covered.

Robot Modeling
A robot is defined via its kinematic structure resulting in a tree-like structure of connected *Robot Nodes*, whereas parameters describing the dynamics can be optionally specified. In Fig. 1, three exemplary models are shown.

Visualization and Collision Detection
Based on the kinematic definition, 3D models for visualization and for data processing can be attached to the *Robot Nodes* resulting in two models: one in high resolution for appealing visualizations and a reduced one that can be used for efficient collision detection and distance calculations (see Fig. 2).



(a) The target locations for the body parts.

(b) Solution without elbow target height.

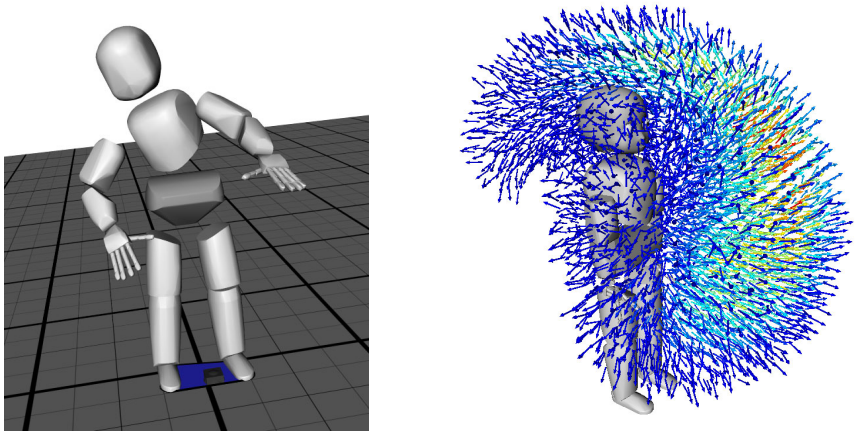
(c) Solution considering the elbow.

Fig. 3 Results of differential inverse kinematics queries.

Kinematic Chains and Robot Node Sets
Usually when a robot is used, several kinematic chains, as logical sets of joints, are needed. Therefore, so-called *Robot Node Sets* can be defined in *VirtualRobot*, allowing to specify a collection of *Robot Nodes* or a strictly defined kinematic chain.

End Effectors
End effectors play an important role in grasp and manipulation planning in the context of humanoid robots. Hence, a convenient end effector definition is provided by *VirtualRobot*, allowing to easily open and close hands while considering self-collisions and collisions with the environment.

Jacobian Calculations and Differential IK
The Jacobian of a robot manipulator is required for many complex calculations in a robotic application and, hence, has to be calculated very efficiently. The most prominent example is the numerical solution of the inverse kinematics (IK) but it also plays a crucial role in calculating the dynamics of a robot. In short, the Jacobian matrix holds the partial derivatives of an end effector's Cartesian position in its columns and can be used to transform joint angle velocities into Cartesian movement. In Simox, special attention has been paid to the construction of the Jacobian and its convenient usage. It is well integrated into its differential inverse kinematics algorithm – a numerical IK solver – but it can also be used easily in other applications. For any node of a node set, the Jacobian as well as its pseudo inverse can be obtained conveniently in its reference frame. In addition, it is possible to exclude either the position or orientation and even individual coordinates from the matrix. This can be useful, for instance, if one is only interested in the vertical acceleration of the end effector while balancing.



(a) The support polygon together with the projected center of mass.

(b) The 6D reachability considering 10 DoF (torso and left arm).

Fig. 4 Stability and reachability computation.

In Fig. 3, a use-case for the algorithm is shown. The differential IK is used to calculate the bimanual robot configuration covering 17 degrees of freedom (DoF) of ARMAR-III (see Fig. 3(a)). Two IK requests have been performed, where both TCPs should reach a given position. Firstly, the left elbow may move freely and has no assigned target location (see Fig. 3(b)). For the second IK-query, the desired height for the robot's elbow is considered additionally to the TCP positions (see Fig. 3(c)). For both queries the differential IK approach is able to serve a solution after few iterations.

An application of Jacobian calculations for stability analysis can be seen in Fig. 4(a). Here, the center of mass (CoM) Jacobain is used to move the robot until the surface projection of its CoM lies within the 2D support polygon to ensure static stability.

Reachability Analysis

Having a representation of the reachability of a kinematic chain, e.g. a robot arm, helps to fulfill several tasks in the context of grasp planning, inverse kinematics (IK) solving and mobile manipulation. The reachability is defined as a 3D or 6D volume being reachable by the end effector. In 6D, the orientation is explicitly considered while it is ignored in the 3D case. Several approaches exist to build a representation of the reachability [15, 10, 16], most of which use a voxelized approximation of the 6D workspace. Reachability structures in *VirtualRobot* can be created for arbitrary kinematic chains considering joint limits and self collisions. A visualization of iCub's reachability for the left end effector is shown in Fig. 4(b). The kinematic chain used for building the reachability data covers three torso and seven arm joints.

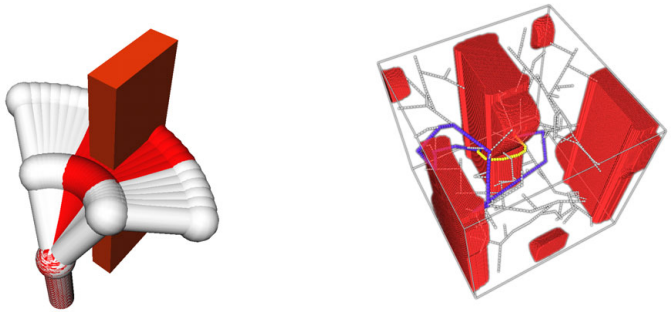


Fig. 5 Extended visualization features allow to analyze planned motions as well as low dimensional configuration spaces.

4 Motion Planning

Saba, a sampling-based motion planning library, is part of *Simox*.

Generic interfaces are provided for collision detection setups, configuration spaces and planners in order to offer a comprehensive and extensible implementation.

4.1 Setup of Collision Detection

Static and moving environmental objects as well as parts of the robot are handled uniformly. All objects of these types which are considered for collision detection are thus grouped into collision sets. Mutual collision detection is achieved by defining pairs of collision sets.

4.2 Representation of Configuration Spaces

The configuration space C (C-space) is defined based on a collision detection setup and a set of *Robot Nodes*, which in most cases form a kinematic chain of the robot. A configuration $c \in C$ can either be valid or invalid, i.e. resulting in a collision in the workspace. In derived implementations, additional constraints or quality measures can be incorporated. When considering paths from $c_0 \in C$ to $c_1 \in C$, several implementations for collision detection are offered. In most cases, a sampling-based approach, where intermediate samples on the path are checked for collisions will be sufficient (see Fig. 5 left). Nevertheless, different implementations, e.g. relying on continuous collision detection, are explicitly supported, as we showed in [17], where continuous collision detection was realized by Quinlan's *Free-Bubble* approach [18].

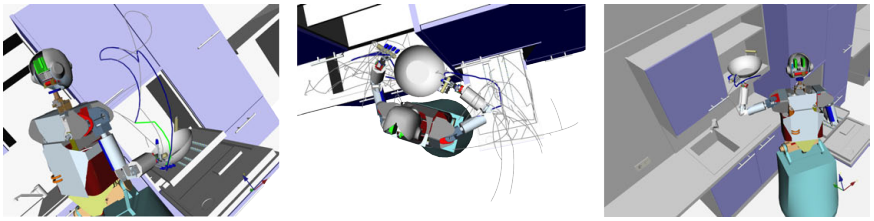


Fig. 6 The bimanual IK-RRT approach is used to plan hand-over motions.

4.3 Motion Planners

Motion planners in *Saba* basically rely on the definition of a C-space, which implicitly defines the incorporated joints and the collision setup. Several ready-to-use implementations, related to Rapidly-exploring Random Trees (RRT), are offered as well as advanced planners, such as the *IK-RRT* [19] approach for the planning of reaching and grasping motions (see Fig. 6). Further, several visualization features allow analyzing the results in the workspace and in low-dimensional C-spaces (see Fig. 5).

4.4 Path Processors

A well-known issue in motion planning with sampling-based approaches is that the resulting trajectories are not optimal. In order to create appealing movements, the trajectories have to be smoothed. This has been done by classes implementing the path processor interface of *Saba*. For example, one processor named *ShortcutProcessor* searches collision-free shortcuts in C-space in order to smooth the result. A resulting path in C-space can be seen in Fig. 5 on the right. Here the planned path is shown in blue and the smoothed one in yellow.

5 Grasp Planning

Grasp planning can be performed with the *GraspStudio* library which is part of *Simox*. Based on the robot definitions, any end effector can be decoupled from the model and considered for grasp planning. The Grasp Center Point (GCP) of an end effector defines the favorite grasping position and an approach direction. A generic grasp planner consists of a module for creating approaching motions and a second module for evaluating the grasp quality. Both components are exchangeable by custom implementations of the provided interfaces. Further, custom grasp planners, which do not rely on the presented planning loop, are explicitly supported. When using the generic

planning approach, the following steps are executed until the number of requested grasps or a timeout is reached:

- A collision-free grasping hypothesis is generated by an implementation of the *Approach Movement Generator* interface.
- The fingers are closed and all contacts are stored.
- The contact information is passed to the *Grasp Evaluation* module in order to compute the grasp quality and/or force closure information.
- Depending on the quality, the grasp is discarded or added to the set of valid grasps.

5.1 Grasping Hypotheses: Approach Movement Generator

GraspStudio provides an already built-in implementation for generating grasping hypotheses. This *Approach Movement Generator* randomly creates grasping positions based on the object's triangle model:

- The normal information of the object's surface is used to sample potential approach directions, which are aligned with the favorite approach movement of the end effector as given with the GCP definition. Additionally, the remaining DoF (rotation around the approach direction) is randomly sampled.
- The end effector is moved towards the object until the GCP is reached by the object's geometry or a collision is detected.
- In case a collision was detected, the end effector is moved backwards until it is collision-free again.

This approach is suitable for most applications, although the interface definition for *Approach Movement Generators* allows convenient realizations of custom approaches.

5.2 Grasp Evaluation: Grasp Wrench Space Computation

A common approach in grasp quality measurement is the grasp wrench space computation, where 6D wrenches are constructed from the contact information [20, 21, 8]. The 6D wrenches represent the contact force and torque and a quality measure is given by analyzing the convex hull of all contact wrenches. Checking whether the origin is inside the convex hull gives an initial indication if the resulting grasp will be a force closure or not, i.e. the object will be fixed in the hand. Further, the minimum distance from the

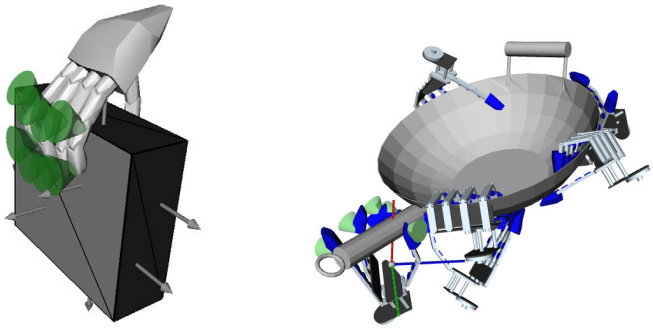


Fig. 7 Several planned grasps for the hand models of iCub and ARMAR-III.

surface of the convex hull to the wrench space origin gives a quality evaluation since it describes the ability of compensating external disturbances. On the left of Fig. 7, a grasping configuration with corresponding friction cones is shown. The right picture shows several grasps that have been planned with *GraspStudio*.

Note that neither the contact model nor the grasp wrench space computation can perfectly represent the reality, but these approaches are meant to approximate the grasping process and to allow efficient planning or efficient first guesses to be refined through learning. Due to the expandable design, *GraspStudio* offers the possibility to also implement more sophisticated planning approaches.

6 Conclusion

With *Simox*, we provide a collection of efficient algorithms to be used in the context of robot simulation, motion and grasp planning. This open source toolbox supports a wide range of applications by offering both state-of-the-art implementations and extendable basic functionalities which can be used to realize more complex algorithms. Future work will address the support for industry standards, such as the COLLADA file format. Further, network transparency and support for shareable objects to be handled within a network of loosely coupled components will be emphasized.

7 Acknowledgments

The research leading to these results has received funding the German Research Foundation (DFG: Deutsche Forschungsgemeinschaft) under the SFB 588 and from the European Union Seventh Framework Programme under grant agreement 270273 (Xperience).

References

1. A.M.S. Laurent, *Understanding Open Source and Free Software Licensing* (O'Reilly Media, Inc., 2004)
2. Open Dynamics Engine (ODE) (2011). URL http://www.ode.org
3. J. Bender, Computer Animation and Virtual Worlds **18**(4–5), 225 (2007). DOI http://dx.doi.org/10.1002/cav.v18:4/5
4. Bullet Physics Engine (2011). URL http://bulletphysics.org
5. Open Motion Planning Library (OMPL) (2010). URL http://ompl.kavrakilab.org
6. Motion Strategy Library (MSL) (2003). URL http://msl.cs.uiuc.edu/msl
7. Motion Planning Kit (MPK) (2006). URL http://ai.stanford.edu/ mitul/mpk
8. A.T. Miller, Graspit!: a versatile simulator for robotic grasping. Ph.D. thesis, Department of Computer Science, Columbia University (2001)
9. S. Ulbrich, D. Kappler, T. Asfour, N. Vahrenkamp, A. Bierbaum, M. Przybylski, R. Dillmann, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2011), pp. 1761–1767
10. R. Diankov, Automated construction of robotic manipulation programs. Ph.D. thesis, Carnegie Mellon University, Robotics Institute (2010)
11. P.I. Corke, IEEE Robotics Automation Magazine **3**(1), 24 (1996)
12. Open Scene Graph (2011). URL http://www.openscenegraph.org
13. Coin3D (2011). URL http://www.coin3d.org
14. E. Larsen, S. Gottschalk, M.C. Lin, D. Manocha, Fast proximity queries with swept sphere volumes. Tech. rep., Department of Computer Science, University of North Carolina (2000)
15. F. Zacharias, C. Borst, G. Hirzinger, in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* (2007), pp. 3229–3236
16. N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour, R. Dillmann, in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on* (2008), pp. 406–412
17. N. Vahrenkamp, P. Kaiser, T. Asfour, R. Dillmann, in *International Conference on Robotics and Automation (ICRA 2011)* (Shanghai, China, 2011), pp. 715–722
18. S. Quinlan, Real-time modification of collision-free paths. Ph.D. thesis, Stanford University (1994)
19. N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, R. Dillmann, in *Intelligent Robots and Systems, IROS* (2009)
20. N. Pollard, Technical Report AI-TR 1464, MIT, Artificial Intelligence Laboratory (1994)
21. D. Kirkpatrick, B. Mishra, C. Yap, in *Proc.of the 20th ACM Symp. on Theory of Computing* (1990), pp. 341–351