Project Acronym:	GRASP
Project Type:	IP
Project Title:	Emergence of Cognitive Grasping through Introspection, Emulation and Surprise
Contract Number:	215821
Starting Date:	01-03-2008
Ending Date:	28-02-2012

Deliverable Number:	D16
Deliverable Title :	Development of robot hands models and sensor emulation in the simulator
Type:	PU/PP
Authors	B. León, G. Puche, A. Morales, S. Ulbrich, T. Asfour R. Dillmann, S. Moisio, and P. Korkealaakso,
Contributing Partners	UJI, UniKarl, LUT

Contractual Date of Delivery to the EC:	28-02-2010
Actual Date of Delivery to the EC:	28-02-2010

Chapter 1

Executive Summary

Deliverable 16 describes the second release of the simulator developed in WP6 "Introspection and Prediction through Simulation". The first release was described Deliverable 9. Both Deliverables have presented the activities in Tasks 6.1 and 6.2 according to the Technical Annex:

- [**Task 6.1**]: Implementation of the engine core architecture and the representation standards.
- [**Task 6.2**]: Development of the basic modules.

The work in this Deliverable is related to **Milestone 5:** "Implementation of high-level controllers including a global uncertainty model, integration and evaluation in the simulator and experimental platforms, grounding grasping primitives."

Improvements in the GRASP simulator

Deliverable 9 introduced the first release of the simulator. Since then, several improvements and extensions have been included in the software package:

- The GRASP simulator is a combination of different existing software packages with some important extensions developed by the GRASP project. The name of the whole toolkit is **OpenGRASP**. The whole software is now accessible at the new url:

  http://opengrasp.sourceforge.net

  From this URL it is possible to download all the components and have access also to new plug-ins, robot models, documentation, and demonstration movies.

  In addition to this we append Attachment 1 which fully describes the whole toolkit, its architecture and its features.

- The second release, described in this Deliverable, includes models of popular robots hands in the robotics community. They have been developed using the Robot Editor from OpenGRASP and can be downloaded from the project site:

  http://opengrasp.sourceforge.net/Downloads.html

- A computational contact model has been developed with the purpose of providing realistic contact simulation. Attachment 2 describes the foundations and implementation of this model. It also describes its use to implement the simulation of a tactile sensor. This report also describes the procedure for the validation of the contact model and the sensors simulation.

Appendix A

Attached papers

1. **OpenGRASP: A Toolkit for Robot Grasping Simulation**, Beatriz León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, Antonio Morales, Tamim Asfour, James Kuffner and Rüdiger Dillmann. IEEE/RSJ International Conference on Intelligent Robots and Systems, (submitted), IROS 2010.

2. **Model of tactile sensors using soft contacts and its application to simulated robot grasping**, Sami Moisio, Beatriz León, Pasi Korkealaakso, Antonio Morales, Technical Report, 2010.

OpenGRASP: A Toolkit for Robot Grasping Simulation

Beatriz León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski,
Antonio Morales, Tamim Asfour, James Kuffner and Rüdiger Dillmann

***Abstract*— Simulation is essential for different robotic research fields such as mobile robotics, motion planning and grasp planning. Especially for grasping, there are no software simulation packages, which provide a holistic environment that can deal with the variety of aspects associated with this problem. These aspects include development and testing of new algorithms, modeling of the environment and robots, as well as modeling of actuators, sensors and contacts.**

In this paper, we present a new simulation toolkit for grasping and dexterous manipulation –called *OpenGRASP*– with special focus the aspects mentioned as well as extensibility, interoperability and public availability. OpenGRASP is based on a modular architecture, that supports the creation and addition of new functionality and the integration of existing and widely used technologies and standards. For instance, OpenGRASP uses an abstraction of physics engines and widely accepted open industrial standard for representation of robot models. In addition, a designated editor has been created for the generation and migration of such models.

We demonstrate the current state of the OpenGRASP development and its application in a grasp evaluation environment.

I. Introduction and Related Work

Robot simulators have accompanied robotics for a long time and have been an essential tool for the design and programming of industrial robots. Almost all industrial manipulators manufacturers offer simulations packages accompanying their robotics products. These tools allow the users to program and test their applications without using the real hardware or even building it since such tools allow to analyze behaviour and performance in advance. In robotics research, simulators have an important role for the development and demonstration of algorithms and techniques in areas like path planning, grasp planning, mobile robot navigation, and others. The reasons for the use of robot simulations are several. First, they allow exhaustive testing and tuning of mechanisms, algorithms on different environmental conditions. Second, they avoid the use and wearing of complex and expensive robot systems. And third, simulation software is cheaper than real hardware.

Often, simulation tools used to support research are specifically developed for particular experiments. However, there has been some successful attempts to develop general robot simulators specially focused on mobile robotics. Stage and Gazebo are respectively 2D and 3D simulators back-ends for Player (see [1], [2]), which is a widely used free software robot interface. In particular, Gazebo [3] implements a 3D multi-robot simulator including dynamics for outdoor environments. It implements several robot models, actuators and sensors. USARSim [4] has a similar functionality. It is a free mobile robot simulator based on a gaming physics. There is also Webots [5], a commercial product which has a wide success for educational purposes. Microsoft is commercially delivering its Robotics Studio [6], a framework for robot programming that includes a visual simulation environment. OpenHRP [7] is open software platform with various modules such as dynamics simulator, view simulator, motion controllers and motion planners for humanoid robot systems. OpenHRP is integrated with CORBA, and each module, including the dynamics simulator is implemented as a CORBA server.

The variety of simulation tools for robotic grasping is rather limited. The most renowned and prominent one is the grasping simulation environment *GraspIt!* [8]. *GraspIt!* includes models of several popular robot hands, implements the dynamics of contacting bodies, includes a grasp planner for a Barrett Hand and recently has included a simple model of the human hand. However, *GraspIt!* has several limitations. It's rather monolithic and less modular architecture makes its improvement and functionality extension by other as well as integration integration with other tools and control frameworks very difficult. In addition, it does not provide a convenient Application Programming Interface (API), which allows script programming. Further, it does not include sensor simulation.

Another existing and public available architecture and software framework is OpenRAVE [9], which has been designed to be an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It is designed in a modular way, which allows its extension and further development by other users. Regarding robot grasping simulation it provides a similar functionality to what GraspIt! and various path planning components. It provides the models of several robot arms and hands and allow the integration of new ones. In addition, it also allows the development of virtual controllers for such models.

The remaining of the paper is organized as follows. Sec. II defines the requirements imposed on the simulation environment. Sec. III describes the underlying components

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 215821, and by Fundació Caixa-Castelló (P1-1A2006-11).

B. León, G. Puche and A. Morales are with the Robotic Intelligence Laboratory at the Department of Computer Science and Engineering, Universitat Jaume I, 12006 Castellón, Spain. {len,puche,morales}@uji.es

S. Ulbrich, M. Przybylski, T. Asfour and R. Dillmman are with the Institute for Anthropomatics, University of Karlsruhe, Germany {ulbrich, przybyls, asfour, dillmann}@ira.uka.de

R. Diankov and J. Kuffner are with the Insitute for Robotics, Carnegie Mellon University, USA {rdiankov, kuffner}@cs.cmu.edu

in *OpenGRASP* including the OpenRAVE architecture, the physics simulation, the implemented file formats and a modeling tools. In Sec. IV we present current results regarding the generation of robot models and the physics engines abstraction. Sec. V concludes the paper and gives an outlook on future work.

II. Requirements for a Grasp Simulator

From a scientific point of view, a novel simulator for robot grasping should provide primarily a realistic simulation of dynamic properties of, at least, rigid objects and advanced contact models including soft contacts and deformable surfaces. From practical and user-level point of view, it should include the models of the most popular robot hands, and provide the possibility of creating and adding new ones. Further, it should provide realistic simulations of real actuators and sensors, which would enable the use of the same API for those simulated robots. Regarding sensors, a grasping simulator has to provides simulations of specific grasping sensors, like force/torque, contact, tactile and others. Finally, it should provide a rich and detailed visualization of simulations.

With respect to software engineering, a novel robot grasping simulator must be implemented in a modular way that allows on the one hand an easy extension by both developers and users and on the other hand the integration within commonly used software frameworks in the robotics. Therefore, such a simulator should be implemented using standard interfaces and protocols to allow communication with third-party applications and tools. Thus, the simulator architecture must clearly separate the different functionalities of the system and integrate all modules through a common representation of the world, objects and robot models.

In order to have the chance to be accepted and used in the scientific community, the simulator should be open source and make use of open standards for file formats and other representations. In addition, the simulator should have appropriate tools import/export of robot and object models from/to standard representations.

To our best knowledge, none of the existing simulation tools and software packages solutions fulfill all these requirements. Therefore, we are present a software toolkit for grasping simulation *OpenGRASP*, which build on top of OpenRAVE [9] to meet the requirements discussed above.

III. Toolkit Description

In order to develop a tool that meets the requirements listed in the previous section, we adopted a basic practical principle: *Do not reinvent the wheel*. This means, first to review the existing software paying special attention to those that already meet part of the requirements and second to make use of existing open and widely software packages and standards.

After a wide review of existing simulators, physics engines, 3D render engines, and CAD 3D modellers we conclude that OpenRAVE is the tool that most closely meets our requirements. So our efforts have consisted in improving and extending the OpenRAVE capabilities and features towards the realization of an advanced graping simulator. These enhancements have consisted in:

- We have improved OpenRAVE core itself by adding a new type of *plug-in* interface for including robot actuators *plugins* and also by developing new types of sensors.
- We have integrated Physics Abstraction Layer (PAL) [10] to allow the interchange of physics engines within the simulator.
- We have chosen COLLADA [11] as the file format for specifying object and robot models used by the simulator.
- We have developed a robot editor to create and modify new robot models.

In the following we describe these extensions in more details.

A. OpenRAVE Architecture

OpenRAVE, the Open Robotics and Animation Virtual Environment [9], is a planning architecture developed at the Carnegie Mellon University Robotics Institute. It is designed for autonomous robot applications and consists of three layers: a core, a plugins layer for interfacing to other libraries, and scripting interfaces for easier access to functions (see Fig. 1).

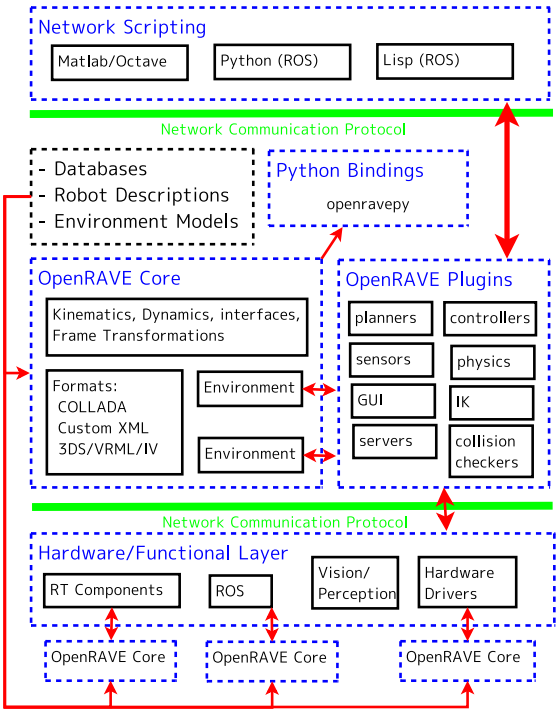


Fig. 1. OpenRAVE Architecture

The Scripting Layer provides network scripting environments like Octave, Matlab and Python to communicate with

the Core Layer in order to control the robot and the environment. It is possible to send commands to change any aspect of the environment, read any of its information, move real robots, or change physics/collision libraries. The scripts also allow the control of multiple OpenRAVE instances across the network, thus allowing different users to independently see and interact with the environment.

OpenRAVE is designed as a plugin-based architecture which allows to create new components to continuously improve its original specifications. Each plugin is an implementation of a standard interface that can be loaded dynamically without the need of recompiling the core. Following this design, different kind of plugins can be created such as sensors, planners, controllers or physics engines. The core layer communicates with the hardware through the plugins using more appropriate robotics packages such as Player and Robot Operating System (ROS).

A GUI can be optionally attached to provide a 3D visualization of the environment. It continuously queries the core to update the world's view andallows the user to change the position of the objects in the scene. Because viewers are provided through plugins, a single OpenRAVE instance can allow multiple viewers to communicate with multiple environments copies.

Although a many plugins are already implemented to provide basic functionality, the current grasp simulation functionality offered has several shortcomings. In order to make OpenRAVE suitable for our purposes, we require:

- Implementation of plugins for specific sensors used to improve the grasping capabilities of the robot.
- Implementation of more physics engines and collision checkers that helps to compare and improve the simulation performance.
- A standard plugin interface of a basic actuator should be added and implementations for these type of motors should be developed. This would allow us to accurately simulate the motors of the arm and hands joints.

We have taken these considerations into account in our toolkit. First of all, we have developed two new sensor plugins to be used mainly on anthropomorphic robot hands. One is a tactile sensor, commonly used in fingers tips such as in the Barrett hand, which detects and calculates the forces on the predetermined sensory area and return them as an array. The other is a force sensor, placed for example in the wrist, to measure the forces applied while grasping.

Additionally, as models for actuators were not included in OpenRAVE, we have developed a new plugin interface called ActuatorBase. Using this interface, we implemented a new plugin to simulate the motor of the arm and hands joints which can be controlled using angles, velocities or voltages.

In order to use different physics engines, we have also implemented a plugin of this kind which make use of a physics abstraction layer called PAL, which is addressed with more detail in the next section.

Visualisation is an important part of the simulation. At the moment OpenRAVE uses Coin3D/Qt to render the environment, but we are extending it to communicate with Blender given that our RobotEditor (see Section III-D) is developed on top of it. Because both Blender and OpenRAVE provide a Python API, it is possible to use Blender as a front-end for not just visualization, but also for calling planners, controlling robots, and editing robot geometry.

We have created the environment setup that we have in the Universitat Jaume I, consisting of a Barrett hand attached to a PA10 Arm, which is fixed to a mobile base (An screenshot can be seen in Fig. 2).

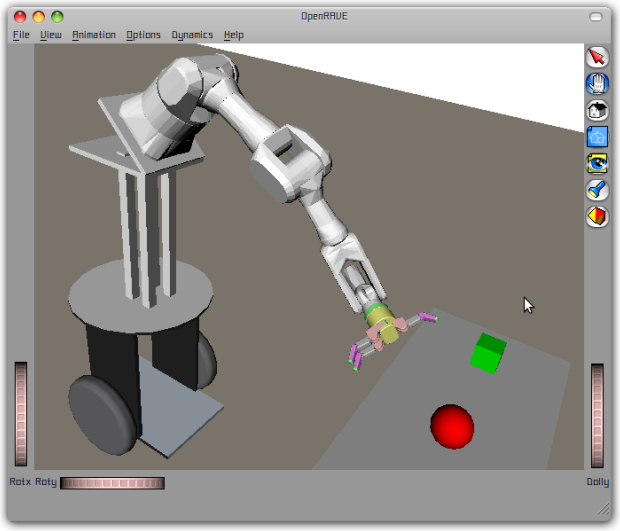


Fig. 2. Screenshot of OpenRAVE simulating the environmet setup located at the Universitat Jaume I (UJI).

B. Physics Simulation

Simulation of the real world consist not only in creating a realistic graphic representation of the environment but also to simulate the forces applied to the bodies and the interactions between them. Physics engines have been created to simulate these interactions and are able to approximately predict what happens in real life.

Nowadays there are many available physics engines, both commercial and open-source. Some of them are high-precision engines that require elevated computational power and others sacrifice this accuracy to work in real time. The methods they use to simulate physics are also different. Some of them use penalty methods, specially useful for deformable objects, some rely on physical laws using constraint equations; and others, use methods based on impulses [12].

Non of these engines are perfect, they all have advantages and disadvantages which make it very difficult to decide which one to use for a simulator. It basically depends on what we want to simulate and also what the application of the simulator will be.

The Physics Abstraction Layer (PAL) [10] is a software created by Adrian Boing which save us from having to decide, since the beginning, which engine to use for our simulator. This layer provides an interface to a number of different physics engines allowing us to dynamically

interchange between them. These functionality adds an incredible flexibility to our simulator offering us the possibility to, depending of our specific environment and use, decide which engine give us the best performance [13]. Using their interface, it is also possible to create our own engines, test and compare them with the existing ones.

The OpenRAVE Physics Engine interface allows the simulator to run using different engines. It has also an interface to implement different collision checkers. Each one of them has to be created as a plugin, extending either the PhysicsEngineBase or the CollisionCheckerBase class. The actual version of OpenRAVE only offers the implementation of ODE (Open Dynamics Engine) within the oderave plugin. We have created a new plugin to use PAL, called palrave. This plugin is able to initialize PAL with the specific engine we want to use, without the need of creating different plugins for each one of them.

C. COLLADA File Format for Robot Models

During the development of the simulator, a new file format for robot models has been chosen. In contrast to the original XML-based file format of *OpenRave*, a format was looked for that

- is widely accepted,
- supports the definition of both kinematics and dynamics,
- is extensible,
- and is public domain.

This way, it becomes possible to easily exchange robot models between all supporting applications and gain more flexibility in the selection of appropriate tools. Another important aspect is the conversion from and to other formats that now becomes possible. However, among the large variety of file formats for 3D models, there are only a few that are open and are not limited to store only visual information. The simulator environment does not rely only on geometrical structures, but also –for instance– on information on dynamics, kinematics, sensors and actuators of the robot.

The acceptance as an industry standard and the wide distribution, in addition to a clear and extensible design led to the choice of COLLADA[1] as the preferred file format for robot models accepted by the simulator. It is already supported as an interchange format by many modelling tools such as 3D Studio, Blender, OSG, OGRE, Sony, etc. In addition to that, there are open source frameworks available that facilitate the integration into new applications. COLLADA stands for COLLAborative Design Activity. The development of the standard is currently managed by the Khronos Group[2]. In August 2008, version 1.5 of the standard was introduced. It now contains useful constructs dedicated to describe kinematic chains and dynamics that can be used directly for the descriptions of robot models. COLLADA is an XML-based file format and enables and encourages developers to extend the specifications to their needs without having to violate the underlying schema definition. Every file entry can hereby be augmented by additional data that is associated to a specified application. The only restriction to this data is to be valid XML code by itself. Constructs embedded in this way can be validated according to an separated schema definition together with the complete document. In order to support specific robot features like sensors and actuators, we have used this mechanism to extended COLLADA. The additions include several parts of the original OpenRAVE file definition. All entries specific to the simulator are be hidden to all other applications and the compatibility remains guaranteed. So far, basic support for the COLLADA import and export has been included in the Simulator.

D. Robot Editor

With the creation of a simulator for grasping also arises the need for a large data base of geometrical, kinematic and dynamic models of robot arms and manipulators. To fill this gap, the development of a modelling tool –the *Robot Editor*– has been started. Its main goal is to allow the convenient creation and integration respectively of the all popular robots available. The development is driven by the following key aspects:

- **Geometric modeling:** The modelling of new robots requires a tool that excels in modeling of the geometrical components (i.e. meshes).
- **Semantic modeling:** Even more important is the ability to allow the description of semantic properties of the robot manipulator, such as the definition of kinematic chains, sensors, actuators, etc –or even specify algorithms.
- **Dynamics modeling:** Another important aspect is the ability to define physical attributes of the robot's elements. At the moment, the focus lies on the dynamics of rigid bodies.
- **Conversion:** Robot models usually come in a variety of different file formats. The modelling tool needs to be capable to process these formats and convert them into the COLLADA standard (see section III-C). Especially GraspIt! files –being an already widely used standard with many conform models available– should be usable by the simulator.

To our knowledge, there is no existing solution openly available that could meet all of these requirements. Therefor, we decided to develop a new modelling tool which is based on available open source software. The conceptual design of the Robot Editor hence relies on two techniques: on the one hand the open data format COLLADA (see section III-C) and on the other hand on the open source project Blender[3]. Blender is a very versatile, powerful and extensible 3D editor that has been chosen as basis because of its

- built-in support for many CAD formats,
- convenient 3D modeling,
- support of rigid body kinematics,
- capability of ray-tracing and the production of high quality rendered images,

[1] https://collada.org
[2] http://www.khronos.org/
[3] http://Blender.org

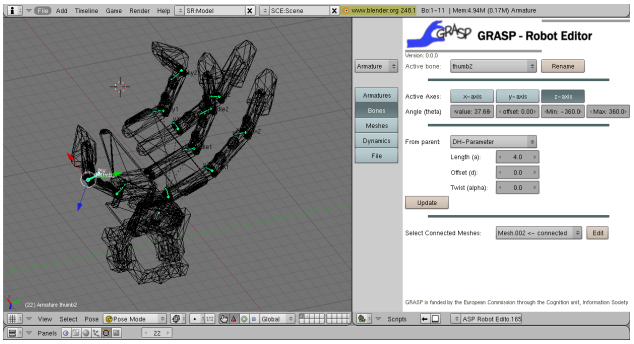


Fig. 3. The Robot Editor with its user interface. The Robot Editor allows the definition of new robot kinematics in a convenient way.

- and easy extensibility via a Python scripting interface.

Blender itself cannot be used directly to generate appropriate input for the simulator. It lacks the functionality and the interface for the definition of robot kinematics and dynamics as well as information on sensors, actuators and algorithms. In addition to that, conversions between certain file formats need to be improved or newly implemented, namely the import of GraspIt! robot models and the COLLADA format.

The scripting interface mechanism mentioned above allows to build the modelling tool on top of Blender. On the scripting level, one gains access to all relevant data structures in Blender. The robot model can be augmented by the required data structures and conserved within the native file format. The scripting mechanism also allows for the creation of an user-interface that is specialized for the use in robotics (see Fig. 3), e.g. you can define a kinematics via Denavit-Hartenberg parameters. In the long run, the Robot Editor will contain and provide interfaces for essential robotics algorithms, e.g. the computation of dynamics characteristics from the geometric meshes and conversions between kinematics representations. Adjacency information of joints and the impact of joint movements to the robot are additional computational information, which is useful for developers planning algorithms. In Fig. 4, a functional model of the anthropomorphic hand of Karlsruhe [14] loaded into the simulator that has been created with the Robot Editor.

The COLLADA support is currently (Blender version 2.49) available in form of import and export scripts that ship with the main Blender distribution. They are published as open source software and have been developed by Illusoft[4]. However, they were designed with compatibility only to documents in version 1.4, and they do not allow other scripts to include or modify the document either. Hence, neither the kinematics and dynamics introduced in version 1.5 nor additional descriptions needed by the simulator can be included in the resulting output . This led to the further development of this COLLADA compatibility which now enables the Robot Editor to create valid COLLADA documents that can be used with the simulator (see Fig. 4). At the time of writing, there is also a concurrent effort in the

[4]http://colladablender.illusoft.com/cms/

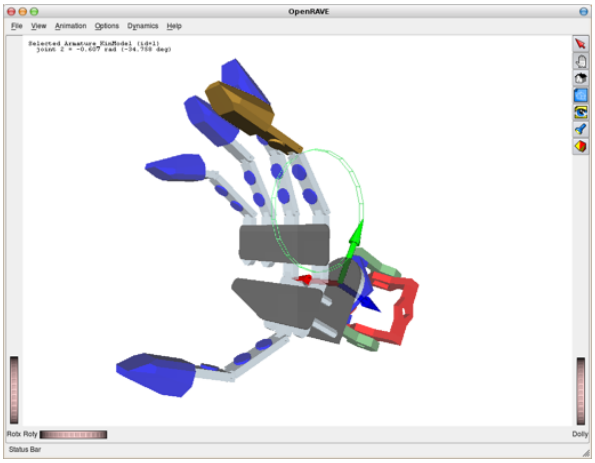
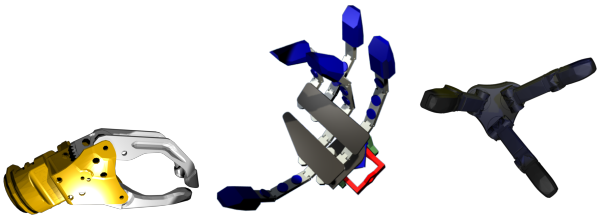


Fig. 4. Screenshot of the complete model of the Karlsruhe anthropomorphic robot hand in the simulator.



a. b. c.

Fig. 5. Different robot hand models generated with the Robot Editor: a) the Otto-Bock hand, b) the Karlsruhe five-fingered hand , and c) the Schunk hand.

community to integrate version 1.5 support in Blender[5] based on the OpenCOLLADA framework[6] which is also currently adapting to the new standard.

IV. Current Achievements

A. Robot Models

As stated in section II, it is of great importance to have models of the most popular robot hands included in the tool kit. The modelling capabilities of the Robot Editor already allow to quickly and comfortably create new models for inclusion. So far, a selection of robot hands have been transformed into COLLADA 1.5 for the use with the simulator:

- a myoelectric upper extremity prostheses of Otto Bock[7],
- the anthropomorphic hand of Karlsruhe [14] (Fig. 5-b),
- and the Schunk three finger hand (Fig.5-a).

All these models can applied in simulation as shown in Fig. 4. In addition to the new models, there are various models available in the old file format which is still supported. These are the barrett hand and the katana, pa10, puma, shadow and ARMAR-III robots.

[5]http://www.blendernation.com/blender-google-summer-of-code-2009

[6]http://www.opencollada.org/

[7]http://www.ottobock.de/cps/rde/xchg/ob_de_de/hs.xsl/384.html

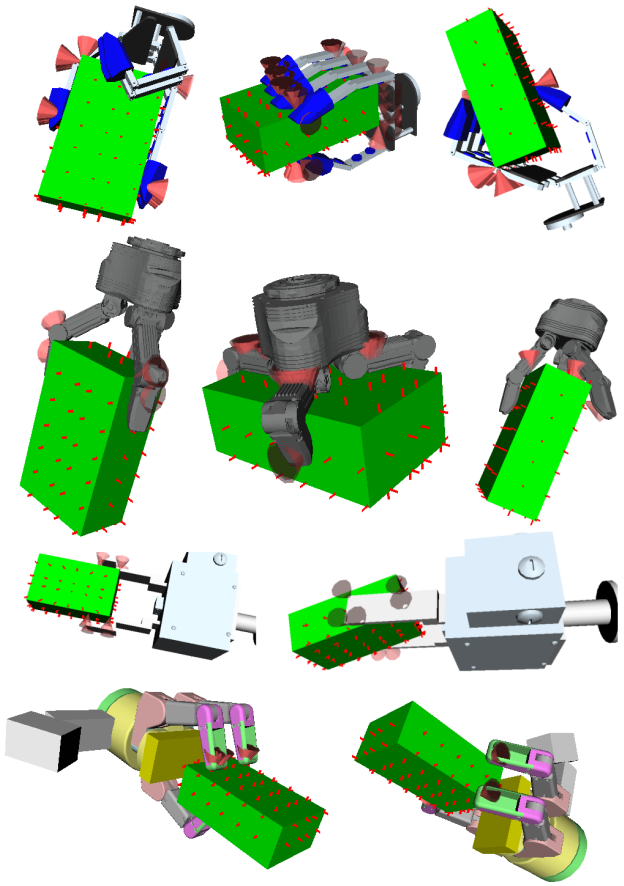


Fig. 6. Grasping simulated for several robot hands.

B. Physics Simulation

The PAL plugin has been implemented to be use with OpenRAVE. Currently PAL supports the most popular physics engines such as Bullet, Newton, Novodex (Ageia PhysX) or ODE (Open Dynamics Engine). In theory, with the development of this plugin, it should be possible to use any of these engines. The reality is that most of them are in current development and new versions are released very often, so their implementation in PAL also has to be constantly updated, which is not always the case. We have tested and fixed some problems with Bullet and ODE, so they can be certainly used with OpenRAVE. We are planning to test some of the others, so we have more options to compare and choose the one with better performance for each particular application.

C. Planning and Grasping

Using the functions provided by OpenRAVE, we can easily build a set of stable grasps and quickly get our robots to manipulate various objects in their environment. Figure 6 shows the grasp simulation process by analyzing the contact points between the robot and the target object. In order to get the robot to autonomously manipulate objects in the environment, we would need an inverse kinematics solver that can quickly map grasp locations into robot configuration joints. Recently OpenRAVE started providing analytical inverse kinematics equation solver called **ikfast**. With it we can generate C++ code that can return all possible IK solutions while simultaneously handling degenerate cases. By combining the automatically generated grasp sets, inverse kinematics solvers, and planners, we can get robots developed in our RobotEditor to manipulate everyday objects (Figure 7).

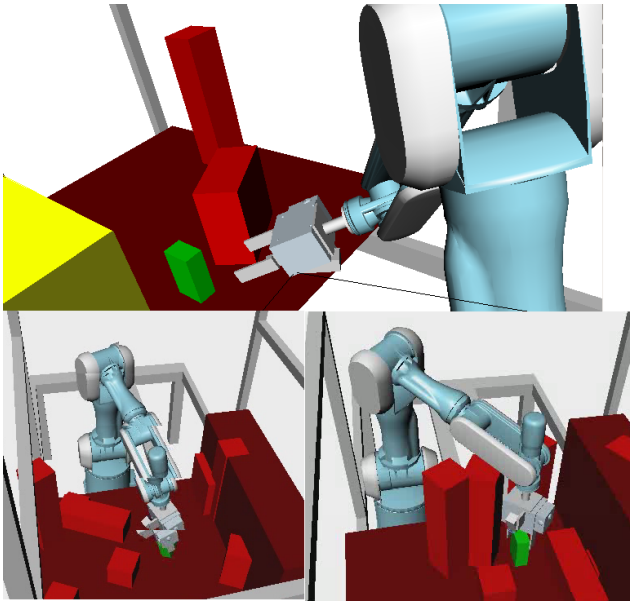


Fig. 7. Several grasping experiments in a cluttered environment with the Mitsubishi PA10 arm and a Schunk gripper.

V. Conclusion and Future Work

The paper presents our ongoing work on the development of an open source software toolkit for grasping simulation. It motivates and describes the features and internals of the toolkit. The toolkit can be downloaded from `http://wikis.itec.uka.de/grasp/wiki/GRASP_Simulator`.

We are currently working on several important features that will be released in the near future. First, and more important we are working in the development and integration on soft contact models within the physics engines, that provides a more realistic behaviour of the interaction between objects and robots hands. Such a contact model will permit also the development of tactile sensor models.

We are also working on the modelling of popular and commercial robot hands, including its actuators and sensor capabilities. This will help for the establishment of the toolkit as helpful tool for researcher teams.

The toolkit has been developed within the GRASP [15] project funded by the European Commission.

References

[1] B. Gerkey, R. T. Vaughan, and A. Howard, “”the player/stage project: Tools for multi-robot and distributed sensor systems”,” in *11th Inter-*

national Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugal, June 2003, pp. 317–323.
[2] "The player project," "http://playerstage.sourceforge.net/wiki/Main_Page".
[3] "Gazebo," "http://playerstage.sourceforge.net/index.php?src=gazebo".
[4] "Usarsim," "http://sourceforge.net/projects/usarsim/".
[5] Cyberbotics, "Webots," "http://www.cyberbotics.com/".
[6] "Microsoft robotics studio," http://msdn.microsoft.com/robotics.
[7] "Openhrp: Open architecture humanoid robotics platform," http://www.is.aist.go.jp/humanoid/openhrp/Englishq/indexE.html.
[8] A. Miller and P. Allen, "Graspit!: A versatile simulator for robotic grasping," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, Dec. 2004.
[9] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34, July 2008.
[10] "Pal (physics abstraction layer)," http://www.adrianboeing.com/pal.
[11] "Collada," http://collada.org.
[12] Wikipedia, "Physics engine - wikipedia, the free encyclopedia," 2009, "http://en.wikipedia.org/wiki/Physics_engine".
[13] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, New York, NY, USA, 2007, pp. 281–288.
[14] I. Gaiser, S. Schulz, A. Kargov, H. Klosek, A. Bierbaum, C. Pylatiuk, R. O. T. Werner, T. Asfour, G. Bretthauer, and R. Dillmann, "A new anthropomorphic robotic hand," in *Proc. IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2008, pp. 418–422.
[15] "GRASP. emergence of cognitive grasping through introspection, emulation and surprise," http://www.grasp-project.eu.

Model of tactile sensors using soft contacts and its application to simulated robot grasping

Sami Moisio, Beatriz León, Pasi Korkealaakso, Antonio Morales

Abstract— **This work uses soft contacts to model a tactile sensor. It is essentially a contact patch with multiple contacts based on a geometry. The element calculates the collisions to triangularized target geometries and then determines the contact forces. The contact itself is a soft contact with a full friction description including stick-slip phenomena. Due to the discrete nature of simulations soft contacts are difficult to model. Especially in grasping a complete friction description is essential in order to perform a stable grasp on an object. Previous collision algorithms allowing for grasping have been non-penetrative which make pressure calculations difficult if not impossible. This method makes soft contact grasping in simulations possible. A simulator with simulated tactile sensors offers new possibilities for studying robot grasping using tactile sensors. The flexibility and repeatability of a simulator can be used to a great advantage in grasping research.**

I. Introduction

The main goal of this work was to create a simulated tactile sensor element. That is to say a simulated tactile element with the same physical properties as a real tactile sensor element would have, compressibility, friction, etc. In order to create a model of the sensor dynamics three different areas need to be addressed: tactile sensor construction, modeling soft contacts and friction modeling. All these areas are combined to make a physical model of a tactile sensor element. The sensor element type itself is universal and can be used to model any kind of a tactile sensor but the aim of this work was to simulate a Weiss robotics tactile sensor commonly used in grasping. A model was created that enables the calculation of surface pressure as well as the holding torque around the contact surface and the stick-slip phenomenon. Due to the discrete nature of simulation these phenomenon are very difficult to model correctly.

Research on using tactile sensors in grasping has been an active topic for some years now [1] obtaining some good results and new methods. Simulation has some great advantages over using real hardware when researching grasping. For example changing or repeating some configuration with the real simulator can be very time consuming. In the simulator these situations are easily and efficiently solved. Changing the configuration of the system can be as easy as just changing a number from the configuration file. The simulation model also is not limited to existing hardware. For example the positioning of the tactile sensors as well as the type can easily be changed unlike in the real hardware. Another great possibility of using a simulator is the availability of different information. The real hardware needs always some sort of an actual sensor for detecting things. The simulator on the other hand has all the information related to the virtual world available for use by just knowing what to query from the simulation. For example the slippage in a tactile sensor can be queried from the sensor model because it exists in the tactile sensor friction model. This availability of information allows for construction on non-real sensor types. For example an approach vector from an end effector base that measures the distance to the nearest object is easy to create, but in real hardware this might be impossible for some end effectors. These non-real sensors offer great possibilities for researching new ways of studying grasping without being restricted by existing hardware. It might even offer new ideas for constructing new types of hardware sensors.

The related work is discussed in Section II. Section III presents the proposed model of the sensors. The application of the model to robot grasping is explained with a use case in Section IV. The results of the implementation are shown in Section V and the conclusion and future work are discussed in Section VI.

II. Previous Work

Robotic manipulators have been researched widely using real hardware. Only in recent years the use of dynamic simulations has been researched in a wider scope. Some simulations environments like GraspIt! [2], [3] have also been developed for the purposes of robotic grasping.

A. Collisions

Contact models can be divided into three different categories: Rigid body assumption for collisions is used in the analytical [4] and in the impulse methods [5], [6] while continuous contact models are used in the penalty methods [7]–[9]. In this context rigid body assumption means non-penetrative or colliding contact in which the exact impact moment is solved after which the surfaces are prevented from penetrating each other. In the impulse based approach contacts between bodies are considered as a collision at a specific point in time without the need of solve contact forces meaning that the change in the object velocities is applied directly to the bodies over one time-step. The method is fast and easy to implement but problem arises with

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 215821, and by Fundació Caixa-Castelló (P1-1A2006-11).

S. Moisio and P. Korkealaakso are with the Centre of Computational Engineering ja Integrated Design (CEID) at the Department of Machine Technology, Lappeenranta University of Technology, Finland {smoisio,korkeala}@lut.fi

B. León and A. Morales are with the Robotic Intelligence Laboratory at the Department of Computer Science and Engineering, Universitat Jaume I, 12006 Castellón, Spain. {len,morales}@uji.es

steady contacts in static configurations. Analytical methods are based on the use of constraints to handle contacts. In contrast to impulse based methods the method is stable in steady contacts but, however, due to simultaneous solving of all contacts it is also computationally expensive. Penalty methods are consequently called penetrative or soft contacts (also non-colliding contacts) because they allow for small penetrations in the colliding objects. Consequently, contact forces are obtained using a temporal nonlinear springdamper element at the contact point. Based on the elasticity of the bodies in contact, the parameters of spring-damper element can be defined using the Herzian contact theory [10].

Analytical and impulse methods gives accurate description for contacts and are often used when there is not allowed any interpenetration between contacting bodies. These methods also allows longer timesteps compared to penalty methods with stiff springs. However these methods leads complicated equation especially in the case of multiple contact points and contacts with friction. Furthermore, in the case of mechatronic machines such as robots, the machine dynamics requires the use of small time steps making penalty methods more suitable especially for real time applications. It is also important to note that rigid body assumption does not account small deformations during collisions, instead there occurs instantaneous changes in velocities. For this reason, continuous contact models gives more accurate description of contact forces during contact period.

Conventional penalty methods use only deepest contact point for contact forces. [11] used geometry based approach in order to find exact contact areas of the polygons applying contact forces to multiple points. However, in the most cases the algorithm is not efficient enough for real time simulation and it is highly dependent on the body geometry construction. The developed method solves the contact forces in each of the sensor cells forming a contact surface. One of the advantages in using penalty methods is the straight forward applicability for solving surface pressures from contacts due to the fact that the objects are allowed to form a real contact surface. In non-penetrative contacts the surface has to be formed using guess or assumptions since the objects are not allowed penetrate and hence to form a real contact surface. This in turn complicates many different calculations for example holding torque around the contact area. For this reason a penalty method was chosen to be used in the developed sensor model.

B. Grasping in simulation

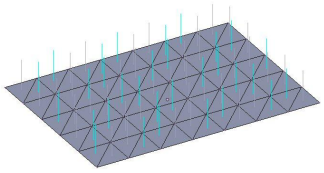
Grasping in robotics using simulated tactile sensors is a new field of research. Some research has been done but they are methods that are derived from existing non-penetrating contact models such as in [12]. In general robotic grasping simulations usually have been using kinematics instead of dynamics. This can be due to the fact that the simulations community has not co-operated extensively enough with the robotic grasping community and the simple fact that simulated grasping is a very difficult problem. The most common simulation method for robotic grasping simulations has been the impulse method (GraspIt!, ODE, Bullet, etc.). The impulse method is a very effective method for simulating structures that form open kinematic chains such as robotic manipulators usually are. The drawback of using impulse methods for solving the constraints between the bodies is that the accuracy of the joint constraints is dependent on the mass ratio of the two objects. This means that if the robotic manipulator has very light grippers attached to a heavy wrist the joints connecting the bodies can suffer from instability. This in turn increases the difficulty of modeling grasping. This mass ratio dependency is also a problem when grasping different objects using the impulse methods [5], [6]. When a very light (Barrett hand fingertip is approximately 50 g) object tries to collide with a very heavy object (3 kg payload) the mass ratio becomes already problematic in impulse methods. Using a penalty method this mass dependency can be avoided but the solution becomes sensible to variables such as the time-step size due to the stiffness of the system.

III. TACTILE SENSOR MODEL

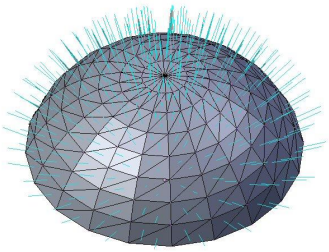
The tactile sensor element is formed based on a triangularized geometry. This was done so that differently shaped sensor elements could be easily defined. For example a finger tip with a tactile sensor is not flat and therefore it would be difficult to describe in order to form the tactile sensor array to encompass the finger tip. A geometry can be formed directly based on the finger tip geometry. In Figure 1 are presented two different variations of a tactile pad element array. One 1(a) being a simple grid and the other a spherical surface 1(b). The image represents the construction of a tactile sensor array. The blue lines represent the normal directions of different triangles. The tactile sensor element array is constructed using the vertices from the sensor geometry. This means that for each vertex the sum of all normals of the triangles connected to it is calculated and used as a normal direction to the sensor element. The sensor element maximum penetration needs to be defined in order to determine the parameters for the sensor. It is also used to place the beginning of a vector pointing in the normal direction to the vertex. This vector in turn is used to calculate the intersection against all possible targets.

The tactile element and its direction vector are then used to determine collision against all objects the tactile sensor is supposed to hit. The contact information (relative velocity, penetration, position, etc.) from this possible contact point is then used to calculate the force in a single tactile element. This force in turn is then applied to the body the tactile sensor is attached as well as the body the tactile sensor is hitting.

So for example in the case of a 6*8 tactile sensor array one would draw a 5*7 grid (such as in image 1(a)) having 6*8 vertices to represent the centers of the tactile elements. These elements are then used to calculate the forces in the tactile pad which are also used in the simulation in order to grasp the object. This information is also used for the feedback



(a) A simple grid



(b) A spherical surface

Fig. 1. Tactile Sensor Geometries

from the tactile sensor element identical to the actual physical sensor.

A. Soft contact model

Briefly, the kinematics of contact point between two bodies a and b can be described using knowledge of the states and geometries of the bodies. The normal of the force vector is obtained from the tactile sensor collisiont model. This normal vector is then used to define the force as well as the tangential plane for the friction. The contact force in the normal direction of the plane for colliding bodies can be written as spring-damper element:

$$F_n = -(kd + cv_{rn})n \tag{1}$$

where k and c are spring and damping coefficients respectively. v_{rn} is the relative velocity between the contact points, d is the penetration distance and n is the penertation distance which are both obtained from the collision detection.

The resulting moments of contact can be written:

$$M_{cont} = \widetilde{p}F_n \tag{2}$$

where $\widetilde{p}$ is skew-symmetric matrix of contact point p defined in global coordinates with respect to local coordinate system.

The contact normal force is then used in order to solve for the friction forces in the contact tangential plane. The friction forces are then added to the force element and the force is applied to the sensor and target bodies.

B. Friction model

In the grasping simulations, it is essential to have a proper description of friction. In order for the friction algorithm to perform in a stable manner the contact normal force algorithm must also be very stable. Otherwise the friction force becomes unstable as well. Another aspect in addition to the friction is the contact points obtained from the collision algorithm. In order to allow for a proper holding torque for example several contact force points need to be generated. In the conventional methods that use deepest contact point for contact forces, the friction is also acting in one point. Due to this fact, the models are not accounting torque induced by the contacting area. However, the problem can be avoided by formulating a proper contact surface such as in the tatile sensor elements. In this study the friction forces are evaluated using LuGre friction model [13] which accounts both static and sliding phenomenas based on bristle deflection interpretation. Accordingly, the LuGre model captures the dynamic behavior of the contact surface using first order differential equation for bristle deflections as follows:

$$\dot{z} = \dot{x} - \sigma_0 \frac{|v|}{g(v)} z \tag{3}$$

where z is bristle deflection, σ_0 is stiffness coefficent and $\dot{x}$ is relative velocity of the contacting surfaces. In Eq. (4) g(v) is used to capture Stribeck effect in order to describe stick-slip phenomena as follows:

$$g(v) = \alpha_0 + \alpha_1^{-\left(\frac{\dot{x}}{\dot{x}_0}\right)^2} \tag{4}$$

$\dot{x}_0$ is the Stribeck velocity. Parameters α_0 and α_1 are defined as follows:

$$\alpha_0 = F_n \mu_d \tag{5}$$

$$\alpha_1 = F_n(\mu_s - \mu_d) \tag{6}$$

where F_n is contact force in direction of the normal of the contact surface, μ_s and μ_d are the static and dynamic friction coefficients, respectively. Using state variables of friction and adding viscous term, the friction force can be written as follows:

$$F = \sigma_0 z + \sigma_1 \dot{z} + c\dot{x} \tag{7}$$

IV. Application to Robot Grasping

A tactile sensor can have several applications to improve robot grasping [14]. First of all, it can use the haptic information to position the hand correctly and detect the contact points when the robot hand is grasping an object. This contact information can be used to evaluate the grasp quality, if a model of the object is available. Tactile sensors can also be used to explore a new object and build its model before grasping it or for object identification. Additionally, it can be used by the controller during the grasping of an object in order to detect the slip between the object and the robot fingers.

Developing a model of a tactile sensor then allows the simulator to take advantage of all these possibilities and it has a wide range of uses for robot grasping. For example when searching for appropriate grasps for an object, a simulator can easily be used to run several approaches to test the different grasps. The simulator can also be used to

easily change the robot configuration. If the research requires testing using sensors that the real robot does not have, it can be done easily on a simulator by just simply defining the sensor to the simulation model. The same principle applies to using the tactile sensor in simulations. If for example one needed to know the best places for placing tactile sensors on a robot it could easily be tested on a simulator.

Also, the simulated tactile sensors have certain advantages over the real ones. First of all, it is always ideal and consistent. No manufacturing faults or drift (unless especially modeled). Secondly, all the information used in the simulation is available for use in the controller. For example measuring the slippage in a real tactile system is difficult, but in a simulated sensor that information is available.

Giving all these advantages, the model of a tactile sensor explained in Section III was implemented and it is detailed in the following sections.

A. Simulated use case scenario

A simple example of a robot hand trying to grasp an object was used as a test case scenario for the simulated tactile sensor. The robot hand has tactile sensors attached to each finger. The following steps contain the basic flow of the use case:

- The robot hand starts static, in a predefined position, with the fingers open.
- An object is placed between the fingers.
- The fingers start closing while the tactile sensors are being read.
- When the readings indicate that the fingers are touching the object, they stop closing.
- The fingers then, are opened to the start position.

This example is demonstrated modeling the platform of the Lappeenranta University of Technology, consisting on a Melfa RV-3SB robot arm with a Shunck PG70 parallel jaw gripper which is shown in Figure 2(a). The arm was not considered in order to concentrate only on the gripper, which was fixed to a specified position.

Each finger of the gripper has attached a Weiss tactile sensor shown in Figure 2(b). These sensors are resistive tactile sensors, which main components are a common electrode and sensing electrodes arranged as a matrix. This matrix measures the change on the resistivity according to the applied load, returning an image of the applied pressure profile [15].

The purpose of this use case is to demonstrate and validate the use of the tactile sensor for a basic grasp activity. In the next sections, the implementation of the use case is explained in detail.

B. OpenRAVE implementation

The simulated model of the tactile sensor was implemented using OpenRAVE [16], a planning architecture developed at the Carnegie Mellon University Robotics Institute. It has been designed to be an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It allows the user to

(a) Shunck PG70 gripper

(b) Weiss Tactile Sensor (DSA 9205)

Fig. 2. Robot and tactile sensor simulated

easily extend its functionality developing their own custom plugins, such a new controllers, sensors, planners and physics engines.

Using this architecture, the model of the Shunck PG70 parallel jaw gripper was implemented and a new plugin for tactile sensors was created, which are described in detail in the following sections.

1) Environment and robot definition: The environment used for the use case consists on a table, the robot hand and a box between its fingers.

OpenRAVE uses XML to store all robot and scene descriptions. The PG70 parallel jaw gripper was modeled using three iv files, one for the base and one for each finger. An snapshot of the OpenRAVE scene showing the PG70 gripper model with the tactile sensors can be seen in Figure 3.

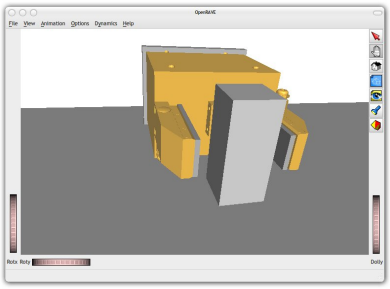


Fig. 3. OpenRAVE model of the PG70 gripper with the Weiss tactile sensors.

2) Tactile Sensor plugin: The model of each tactile sensor was implemented in a new Sensor plugin.

This plugin defines the parameters of the sensor stored in the Tactile Sensor Geometry, the Tactile data that it returns and the implementation of the base interface for sensors including the initialization and uptate at each time step.

3) Tactile Sensor geometry: The plugin allows the user to parameterize the sensor with specific values described in the Tactile Sensor geometry. A new XML reader was created to read this values from the xml file.

As explained in the previous sections, the model of the sensor is based in a mesh which defines the vertices where the forces are calculated. In OpenRAVE, each sensor is usually attached to a link in order to move the sensor in conjunction with the robot. For the tactile sensor this link should contain the mesh geometry needed for the sensor definition. The xml tag to use this definition is $<$ shapeBaseOnGeom $>$, which needs to be defined "true". The false value will be used to define the sensor geometry based on a plane mesh matrix defined with the number of cells in x and y directions.

The tactile sensor geometry should also contain the thickness of the sensor and the parameters used to calculate the friction force (F) described in Table I.

TABLE I

FRICTION FORCE PARAMETERS.

parameter	description
sigma0	Stiffness coefficent
xdot0	Stribeck velocity
mu_s	Static friction coefficient
mu_d	Dynamic friction coefficient
K	Spring coefficient
C	Damping coefficient

The following is an example of the OpenRAVE XML code needed to attach a tactile sensor to a robot:

```
< AttachedSensor >
< link > object1 < /link >
< sensorname = "T1" type = "SimTactileSensor" >
 < shapeBaseOnGeom > true < /shapeBaseOnGeom >
 < sigma0 > 20000 < /sigma0 >
 < xdot0 > 0.5 < /xdot0 >
 < mus > 0.6 < /mus >
 < mud > 0.3 < /mud >
 < K > 1000 < /K >
 < C > 100 < /C >
< /sensor >
< /AttachedSensor >
```

4) Tactile Sensor data: The data returned for the tactile sensor is the same structure as the one defined for the real sensor. In addition to the sensor type and the size of the tactile array, it consist of a vector of doubles containing the values of the forces calculated in each vertex of the sensor mesh and the value of the combined force.

5) Tactile Sensor plugin implementation: When OpenRAVE finds the XML code to attach a tactile sensor, it creates a new instance of the Simulated tactile sensor plugin with the parameters specified. It then waits until all the objects and robots in the scene are created and finds the link which the sensor is attached to. This link contains the mesh which is used to store the triangles and vertices needed to define the sensor.

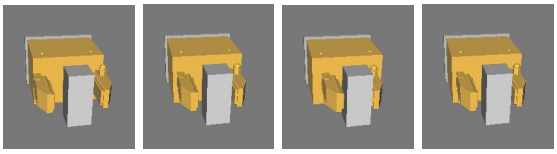
Now that the sensor is created, it gets all the objects in the environment excluding the ones that belongs to the robot. These objects are stored as they are the ones that can collide with the sensor. At the moment, all objects in the environment are considered in the calculation but this process can be optimized if only the objects that are going to be grasped by the robot are specified in the XML file.

After this, each time step, when the sensor is updated, it gets the positions and velocities of the sensor and collision links and it checks if they are colliding. Finally, it calculates the friction forces and updates the tactile data structure. This data can be read calling the GetSensorData method, for example by the controller.

C. Results

The tactile sensor was implemented in OpenRAVE using the plugin architecture. It was loaded at runtime and it successfully create the sensor, loaded the collision objects and performed the use case with the help of the controller.

The sequence of the use case execution is shown in Figure 4.



(a) Open (b) Closing (c) Stop (d) Opening

Fig. 4. Robot and tactile sensor simulated

The controller was checking the tactile readings when closing the gripper until they showed that the sensor was touching the box. They presented a meaningful behavior, given zero when the gripper was closing and positive values when it approached to the box. As future work, the readings obtained with this simulated model are going to be compared with the ones gotten with the real sensors executing the same example, in order to calibrate and validate the model of the tactile sensor.

V. CONCLUSION AND FUTURE WORK

This paper presented an approach to model a tactile sensor using soft contacts. Using OpenRAVE, a new Tactile sensor plugin was developed and tested using a simple grasp use case.

Our priority for future work is the validation of the model and simulation system using a real robot. The system will be verified using the PG70 system used at the Lappeenranta University of Technology. The robot controller will be used to manipulate both the real and the simulated robot. The results will then be verified and validated using a case where

a simple grasp action is taken using the PG70. After the validation the same system can be added to the other robot models in the OpenRave. The tactile sensor element will also be improved to be more efficient. Also some other features need to be developed for the sensor element like creating the sensor element using parameters.

This work has been developed within the GRASP [17] project funded by the European Commission.

REFERENCES

[1] J. Felip and A. Morales, "Robust sensor-based grasp primitive for a three-finger robot hand," in *IEEE/RSJ International. Conference on Intelligent Robots and Systems*, Oct. 2009.

[2] A. Miller and H. Christensen, "Implementation of multi-rigid-body dynamics within a robotic grasping simulator," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, Sept. 2003, pp. 2262–2268 vol.2.

[3] A. Miller and P. Allen, "Graspit!: A versatile simulator for robotic grasping," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, Dec. 2004.

[4] D. Baraff, "Analytical methods for dynamic simulation of non-penetrating rigid bodies," *SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 223–232, 1989.

[5] B. V. Mirtich, "Impulse-based dynamic simulation of rigid body systems," Ph.D. dissertation, 1996.

[6] P. Kraus and V. Kumar, "Compliant contact models for rigid body collisions," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 2, Apr 1997, pp. 1382–1387 vol.2.

[7] M. Moore and J. Wilhelms, "Collision detection and response for computer animationr3," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1988, pp. 289–298.

[8] C. F. R. E. Hunt, K. H., "Coefficient of restitution interpreted as damping in vibroimpact," *Transactions of the ASME Journal of Applied Mechanicss*, vol. Series E, no. 42, pp. 440–445, 1975.

[9] Y. A. KHULIEF and A. A. SHABANA, "A continuous force model for the impact analysis of flexible multibody systems," *Mechanism and Machine Theory*, no. 22, p. 213224, 1975.

[10] K. L. Johnson, "Contact mechanics," in *Cambridge University Press*. University Press, 1985, p. 452.

[11] G. Hippmann, "An algorithm for compliant contact between complexly shaped bodies," *Multibody System Dynamics*, vol. 12, no. 4, pp. 345–362, December 2004. [Online]. Available: http://www.springerlink.com/content/w1743724r0674033/

[12] J. Tegin and J. Wikander, "Simulating tactile sensors in robotic grasping," in *Proceedings of The Third Swedish Workshop on Autonomous Robotics*, vol. 1, Stockholm, Sweden, Sept 2005.

[13] C. Canudas de Wit, H. Olsson, K. Astrom, and P. Lischinsky, "A new model for control of systems with friction," *Automatic Control, IEEE Transactions on*, vol. 40, no. 3, pp. 419–425, Mar 1995.

[14] D. Göger, N. Gorges, and H. Wörn, "Tactile sensing for an anthropomorphic robotic hand: hardware and signal processing," in *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 2972–2978.

[15] K. Weiss and H. Worn, "The working principle of resistive tactile sensor cells," in *Mechatronics and Automation, 2005 IEEE International Conference*, vol. 1, July-1 Aug. 2005, pp. 471–476 Vol. 1.

[16] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[17] "GRASP. emergence of cognitive grasping through introspection, emulation and surprise," http://www.grasp-project.eu.